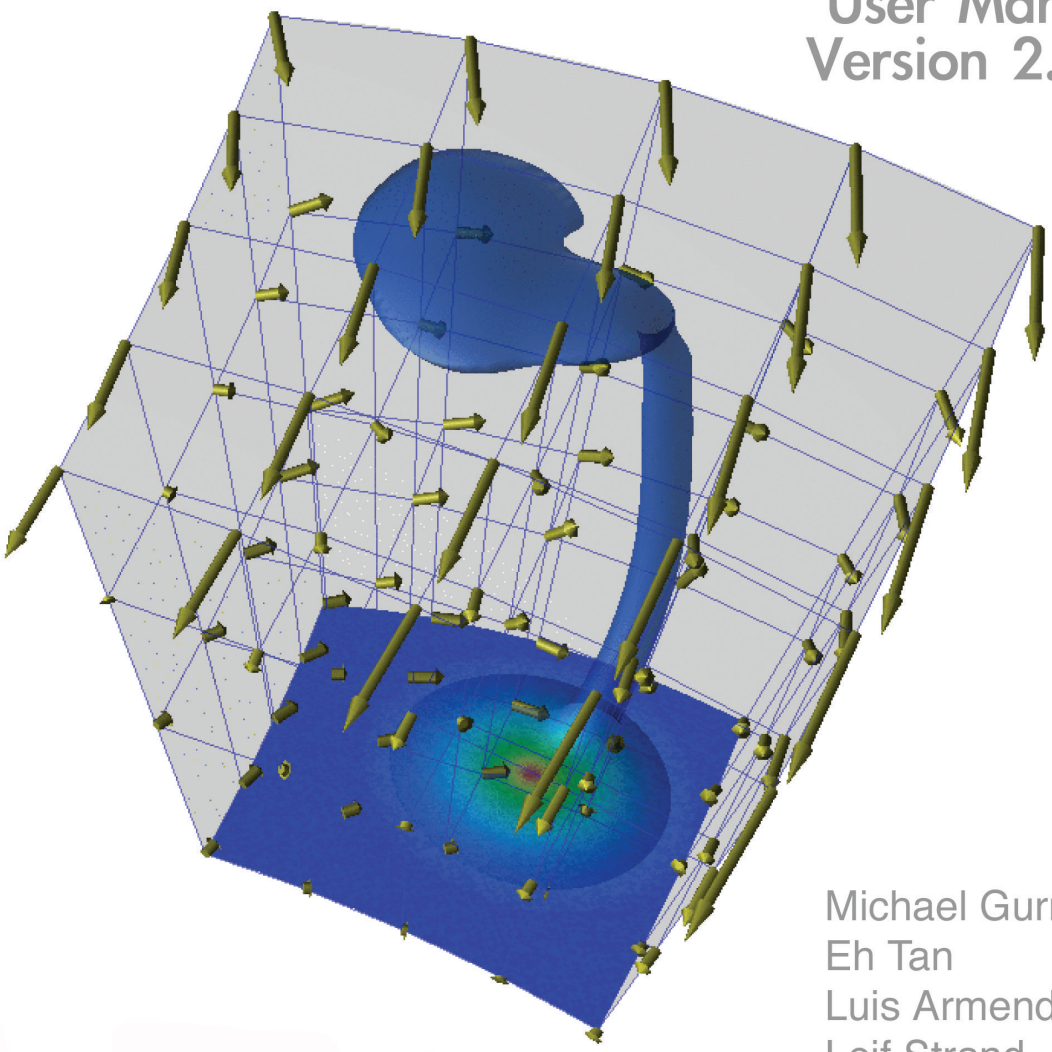
COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

CitComS

User Manual
Version 2.2.1



Michael Gurnis
Eh Tan
Luis Armendariz
Leif Strand
Cassie Ferguson
Susan Kientz

www.geodynamics.org

CitComS User Manual

© California Institute of Technology
Version 2.2.1

March 26, 2007

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 a "cookbook" 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/documentation/UserExperience/Conceptual/APStyleGuide/AppleStyleGuide2003.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 CitComS.py on a variety of different platforms. CitComS.py has run on shared memory computers (Sun, Hewlett-Packard, SGI, and IBM), commercial distributed memory machines (Intel and Cray/SGI), and clusters (including machines on the NSF TeraGrid).

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 finite element package and the Python bindings for the framework were donated to CIG in July of 2005. A number of individuals have contributed a significant portion of their careers toward the development of CitComS.py and Pyre. 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 CitComS development team asks that you cite both of the following:

- Zhong, S., M.T. Zuber, L.N. Moresi, and M. Gurnis (2000), The role of temperature-dependent viscosity and surface plates in spherical shell models of mantle convection. *J. Geophys. Res., 105*, 11,063-11,082
- Tan, E., E. Choi, P. Thoutireddy, M. Gurnis, and M. Aivazis (2006), GeoFramework: Coupling multiple models of mantle convection within a computational framework, *Geochem., Geophys., Geosyst. 7,* Q06001, doi:10.1029/2005GC001155.

Additionally, if you are using tracers in CitComS, please cite the following:

- McNamara, A.K., and S. Zhong (2004), Thermochemical structures within a spherical mantle: Superplumes or Piles?, *J. Geophys. Res.*, 109, B07402, doi:10.1029/2003JB002847.

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 (`www.geodynamics.org`).

Support

Pyre development is funded by the U.S. Dept. of Energy's Advanced Simulation and Computing program (`www.sandia.gov/NNSA/ASC`) and the National Science Foundation's (`www.nsf.gov`) Information Technology Research (ITR) program (grant #0205653). Continued support of CitComS.py is made possible under NSF EAR-0406751.

Conventions

Throughout this documentation CitComS.py refers to the Pyre-compatible version of CitComS unless specifically stated otherwise. 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

CitComS 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 97). 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 CitComS

CitComS is a finite element code written in C that solves for thermal convection within a spherical shell. It has two variants, `CitcomSFull` and `CitcomSRegional`; the first solves for problems within a full spherical domain, and the second, for a restricted domain of a full sphere. 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. CitComS.py allows easy use of either one of these two geometries by simply changing command line options.

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.py. 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.py 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. There are versions of CitCom which solve for these classes of forward and backward problems. Variable viscosity, including temperature-, pressure-, position-, composition-, and stress-dependent viscosity are all possible, although they may not be fully implemented in the current version.

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

CitCom (for **C**alifornia **I**nstitute of **T**echnology **Co**nvection in the **M**antle) was originally written in the early 1990s by Louis Moresi. Although the code for three-dimensional problems was incorporated from its inception, early versions of the software only solved for time-dependent convection problems within two-dimensional Cartesian domains. Moresi's original code turned out to be incredibly modular and easily

extensible. Consequently, the fundamental finite element infrastructure which Louis wrote is still in place and forms the basis for much of the code contained in the present release.

In the mid-1990s Moresi wrote versions of the code that solved the equations within three-dimensional Cartesian domains. Then Shijie Zhong successfully parallelized CitCom using message passing routines on a limited release Intel supercomputer. Zhong then created a spherical version of the code which he named CitComS. Lijie Han then created a regional version of CitComS as well as an alternate version of message passing for an arbitrarily large number of processors. Clint Conrad created the first Beowulf implementations of the code, then Conrad and Eh Tan re-coded the message passing of the fully spherical version so that problems run on arbitrarily large numbers of processors could also be solved. A plethora of different versions of CitCom exist both on computers at the California Institute of Technology and around the world.

Consequently, by 2002, there were so many different versions of the code that some rationalization was in order. The software was migrated into a version control system and Eh Tan and Eun-seo Choi created a version of CitComS that generates either a fully spherical or regional model, `CitcomSFull` and `CitcomSRegional` respectively. CitComS was released to the community through the former GeoFramework project as version 1.0 and 1.1.

By 2004, in order to increase the functionality of CitComS, the developers began to reengineer the code into an object-oriented environment specifically so it could work with a Python-based modeling framework called Pyre. This release of the software, now named CitComS.py, is essentially the product of those reengineering efforts. Eh Tan was the principal developer of CitComS.py, with considerable help from Eun-seo Choi, Puru Thoutireddy, and Michael Aivazis.

CitComS is one component of a larger collection of software encompassed by the former GeoFramework project, a collaboration between the Center for Advanced Computing Research (CACR) (`www.cacr.caltech.edu`) and the Seismological Laboratory (`www.gps.caltech.edu/seismo`), both at Caltech, and the Victorian Partnership for Advanced Computing (`www.vpac.org`) in Australia. The GeoFramework project developed a suite of tools to model multi-scale deformation for Earth science problems. This effort was motivated by the need to understand interactions between the long-term evolution of plate tectonics and shorter term processes such as the evolution of faults during and between earthquakes. During 2005 and 2006 much of the remaining software developed by GeoFramework was released under a GPL license and made available from CIG (`www.geodynamics.org`).

The second major release of CitComS (2.0) incorporated the software framework Pyre, free surface modeling methods, and stress boundary conditions on the top and bottom surfaces. In the summer of 2005, as part of the 2.0.1 release, CIG replaced the old build procedure with the GNU Build System. A subsequent release, version 2.0.2, could compile and run on 64-bit systems.

The third major release of CitComS (2.1) incorporated new features and functionality, the most important being the use of HDF5 (a parallel version of the Hierarchical Data Format). The HDF5 format allows you to deal with the massive data output created for production runs (see Chapter 4 on page 39). This version accepted `.cfg` files on input, which are easier to create and read.

Other improvements included the incorporation of geoid calculations that had been left out of earlier releases, as well as new scripts to allow results to be visualized with MayaVi2 (`svn.enthought.com/enthought/wiki/MayaVi`) in addition to Generic Mapping Tools (GMT) (`gmt.soest.hawaii.edu`) and OpenDX (`www.opendx.org`). Instructions were provided on using this version as a preinstalled package on some of the NSF TeraGrid sites.

This release of CitComS (2.2) incorporates the ability of tracing particles in the flow. The tracer code was developed by Allen McNamara and Shijie Zhong in 2004 and donated to CIG in early 2007. The tracer code has a wide range of applications in the mantle convection. It can be used in tracing the trajectory of passive particles, in delineating the top boundary of subducted slabs to define the low viscosity wedges, or in tracking the evolution of the chemical composition field.

1.3 About Pyre

Pyre is an object-oriented environment capable of specifying and launching numerical simulations on multiple platforms, including Beowulf class parallel computers and grid computing systems. Pyre allows the binding of multiple components such as solid and fluid models used in Earth science simulations, and different

meshers. The Pyre framework enables the elegant setup, modification and launching of massively parallel three-dimensional solver applications.

Pyre is a framework, a combination of software and design philosophy that promotes the reuse of code. In their canonical software design book, *Design Patterns*, Erich Gamma *et al* condense the concept of a framework concept down to, "When you use a framework, you reuse the main body and write the code it calls." In the context of frameworks and object-oriented programming, Pyre can be thought of as a collection of classes and the way their instances interact. Programming applications based on Pyre will look similar to those written in any other object-oriented language. The Pyre framework contains a subset of parts that make up the overall framework. Each of those parts is designed to solve a specific problem.

The framework approach to computation offers many advantages. It permits the exchange of codes and promotes the reuse of standardized software while preserving efficiency. Frameworks are also an efficient way to handle changes in computer architecture. They present programmers and scientists with a unified and well-defined task and allow for shared costs of the housekeeping aspects of software development. They provide greater institutional continuity to model development than piecemeal approaches.

The Pyre framework incorporates features aimed at enabling the scientific non-expert to perform tasks easily without hindering the expert. Target features for end users allow complete and intuitive simulation specification, reasonable defaults, consistency checks of input, good diagnostics, easy access to remote facilities, and status monitoring. Target features for developers include easy access to user input, a shorter development cycle, and good debugging support.

1.4 Pyre and CitComS.py

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.py and to fine-tune CitComS.py simulations. Future versions of this documentation will cover coupled simulations generated via an "exchanger" module.

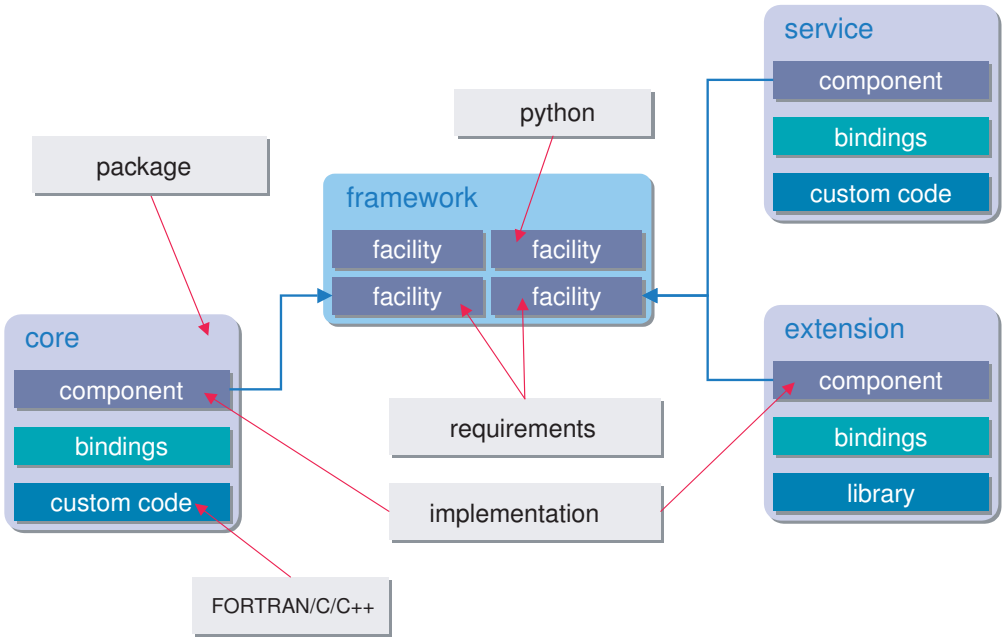


Figure 1.1: Pyre Architecture. The integration framework is a set of cooperating abstract services.

Developers have created Pyre classes for CitComS.py 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.py 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.py with other solvers.

This version of CitComS.py "attaches" to Pyre via the use of bindings. They are included with CitComS.py, eliminating the need for users to write or alter them.

1.5 Governing Equations

With CitComS, the mantle is treated as an incompressible viscous spherical shell. With these assumptions, thermal convection is governed by the equations for conservation of mass, momentum, and energy:

$$u_{i,i} = 0 \tag{1.1}$$

$$-P_{,i} + (\eta u_{i,j} + \eta u_{j,i})_{,j} + \delta\rho g\delta_{ir} = 0 \tag{1.2}$$

$$T_{,t} + u_i T_{,i} = \kappa T_{,ii} + H \tag{1.3}$$

where u is the velocity, P is the dynamic pressure, $\delta\rho$ is the density anomaly, g is the gravitational acceleration, η is the viscosity, T is the temperature, κ is the thermal diffusivity, 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. Without phase transitions and composition variation, the density anomalies are:

$$\delta\rho = -\alpha\rho_0(T - T_0) \tag{1.4}$$

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

$$x_i = R_0 x_i^{'} \tag{1.5}$$

$$u_i = \frac{\kappa}{R_0} u_i^{'} \tag{1.6}$$

$$T = \Delta T T' + T_0 \tag{1.7}$$

$$t = \frac{R_0^2}{\kappa} t^{'} \tag{1.8}$$

$$\gamma = \frac{H R_0^2}{\kappa \Delta T} \tag{1.9}$$

$$\eta = \eta_0 \eta^{'} \tag{1.10}$$

$$P = \frac{\eta_0 \kappa}{R_0^2} P^{'} \tag{1.11}$$

where $\rho_0$is the reference density, R_0 is the radius of the Earth, η_0 is a reference viscosity, and ΔT is the superadiabatic temperature drop from the core-mantle boundary (CMB) to the surface. Dropping the primes, the equations become:

$$u_{i,i} = 0 \tag{1.12}$$

$$-P_{,i} + (\eta u_{i,j} + \eta u_{j,i})_{,j} + RaT\delta_{ir} = 0 \tag{1.13}$$

$$T_{,t} + u_i T_{,i} = T_{,ii} + \gamma \tag{1.14}$$

where Ra, a Rayleigh number, is defined as:

$$Ra = \frac{\rho_0 g \alpha \Delta T R_0^3}{\eta_0 \kappa} \tag{1.15}$$

This is not the usual definition of the Raleigh number that is based on layer thickness, not R_0. So for mantle convection problems where R_0 is about twice the layer thickness, our Ra are about a factor of 8 larger than by the usual definition.

If there is a phase change, equation 1.13 is modified to:

$$-P_{,i} + (\eta u_{i,j} + \eta u_{j,i})_{,j} + (RaT + Rab\Gamma - RacC)\delta_{ir} = 0 \tag{1.16}$$

$$\Gamma = \frac{1}{2}\left(1 + \tanh\left(\frac{1 - r - d_{ph} - s(T - T_{ph})}{w_{ph}}\right)\right) \tag{1.17}$$

where C is the composition, d_{ph} and T_{ph} are the ambient depth and temperature of a phase change, s is the Clapeyron slope of a phase change, and w_{ph} is the width of a phase transition. The phase-change Rayleigh number, Rab, and the chemical Rayleigh number, Rac, are defined as:

$$Rab = Ra\frac{\delta\rho_{ph}}{\rho_0} \tag{1.18}$$

$$Rac = Ra\frac{\delta\rho_{ch}}{\rho_0} \tag{1.19}$$

where $\delta\rho_{ph}$ is the density jump across a phase change, and $\delta\rho_{ch}$ is the density difference between the compositions.

1.6 Numerical Methods

The governing equations are solved with the finite element method [8]. CitComS employs an Uzawa algorithm to solve the momentum equation coupled with the incompressibility constraints [3, 9]. The energy equation is solved with a Petrov-Galerkin method [10]. 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, such as dynamic topography, in incompressible Stokes' flow [8]. The discrete form of equations 1.12 and 1.13 may be written in the following matrix form [5]:

$$\mathbf{B}^T u = 0 \tag{1.20}$$

$$\mathbf{A}u + \mathbf{B}p = f \tag{1.21}$$

where $\mathbf{A}$ is the "stiffness" matrix, u is a vector of unknown velocities; $\mathbf{B}$ is the discrete gradient operator, 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 $\mathbf{A}$, $\mathbf{B}$, and f are obtained using a standard finite element formulation, see [5] for the explicit entries. Equation 1.21 can be transformed by premultiplying by $\mathbf{B}^T\mathbf{A}^{-1}$ and using equation 1.20 to eliminate the velocity unknowns:

$$\mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}p = \mathbf{B}^T\mathbf{A}^{-1}f \tag{1.22}$$

This equation is solved using the Uzawa algorithm, an established method for solving the minimization of a dual function [11], which simultaneously yields the velocity field. A conjugate gradient scheme for this iteration is described by [9, 12], and forms the basis for the technique used in CitComS.

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. For a regional mesh 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. 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. The caps are approximately square in map view so that the edges of the square are orientated diagonally with respect to latitude and longitude. The four corners of the domain are connected by great circles (Figure 1.2). One would normally associate at least one processor with one cap. However, CitComS can further automatically generate meshes with domain decomposition such that additional processors are used to divide caps uniformly along the two edges of the caps (Figure 1.2) as well as in radius.

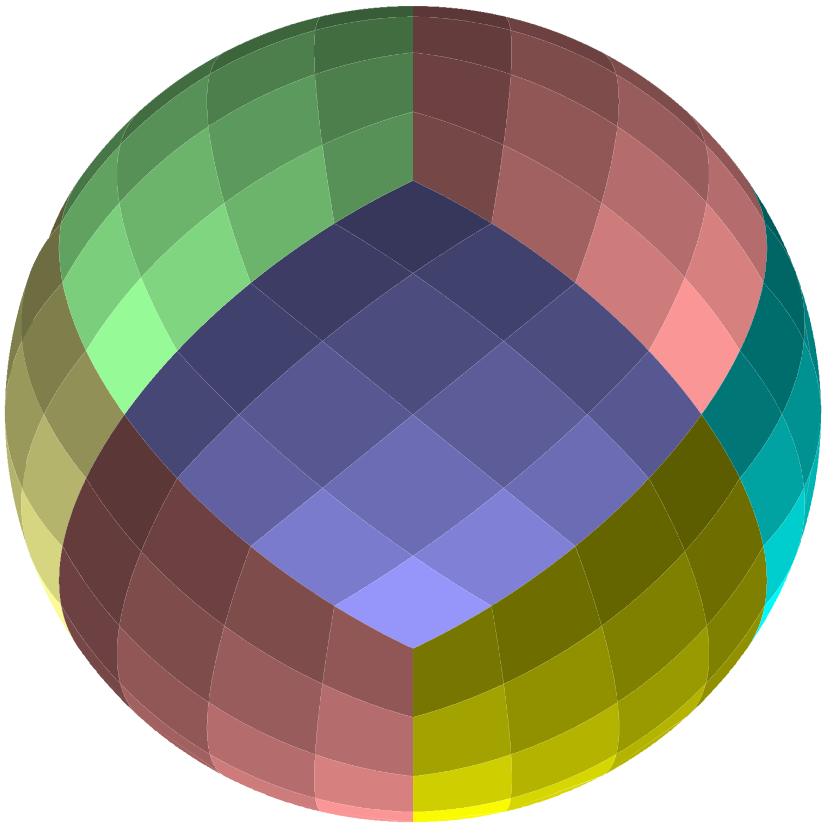


Figure 1.2: Orthographic projection of processors from a full CitComS mesh in which there are 16 processors in map view for each cap. The CitComS cap is shown as distinct colors while the processor domains within the caps are indicated by the intensity of the color. This example was produced for a run with 2 processors in radius such that the total number of processors was $12 \times 16 \times 2 = 384$.

Chapter 2

Installation and Getting Help

2.1 Introduction

To install CitComS.py, you follow the procedure that is commonly used with other open source software packages. First, download the source package (in the form of a compressed `tar` file) available at the Geodynamics Software Packages web page (`www.geodynamics.org/cig/software/packages`). After unpacking the source, you run a prepackaged shell script to configure CitComS for your system. Finally, you use the `make` utility to build and install CitComS from source.

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 2.10 later in this chapter.

CitComS.py has been tested on Linux, Mac OS X, and several NSF TeraGrid platforms.

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 (`www.geodynamics.org/cig/lists`).

2.3 System Requirements

Installation of CitComS.py requires the following:

- A C compiler
- An MPI library
- Python 2.3 or greater (Python 2.4 or greater is required on 64-bit machines), including header files

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. This process requires only a C compiler and an MPI library. For more information, see Section 2.9 later in this chapter.

Optionally, CitComS can be configured to use a parallel HDF5 library. See Section 2.7 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`). Installing MPICH from source involves walking through the standard GNU build procedure (`configure && make && make install`).

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
```

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

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

```
$ python -V
Python 2.3.4
```

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 include files, 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
$ 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`).

2.4 Downloading and Unpacking Source

Download CitComS.py from the Geodynamics website (`www.geodynamics.org`). Click the software tab at the top of the page. Then click Software Packages, then Mantle Convection. Once you click the CitComS link, download the source archive and unpack it using the `tar` command:

```
$ tar xzf CitcomS-2.2.1.tar.gz
```

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

```
$ gunzip -c CitcomS-2.2.1.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-2.2.1
   ```

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.

2.5.0.1 Installing to a Secondary Location

Optionally, after building CitComS, you can install it in a secondary location using the `make install` command.

Note: The following is not recommended if you plan on modifying the CitComS source.

By default, CitComS is configured to install under `/usr/local`, which requires that you become `root` before issuing the `make install` command.

```
$ su
Password:
# make install
```

To install as an ordinary user instead of `root`, give `configure` the `--prefix` option, specifying a directory to which you have write access:

```
$ cd CitcomS-2.2.1
$ ./configure --prefix=$HOME/cig
$ make
$ make install
```

The above commands 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. Most 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.

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.

CitComS v2.1 requires Pythia v0.8.1.x, which `configure` downloads directly from CIG. (Pythia v0.8.1.x is distinct from Pythia v0.8, which you may have installed previously.) Pythia v0.8.1.x is pure Python, and resides in a file or directory with the extension `.egg` (e.g., `pythia-0.8.1.0.egg`). The `configure` script may download other `.egg` packages, in addition to Pythia. These are Python packages which are required (either directly or indirectly) by CitComS.py.

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/python2.3/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. See Section 2.6.3 for details and examples.
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.

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 HDF5 Configuration (Optional)

For writing its output in binary format, CitComS requires parallel HDF5 (PHDF5). In turn, PHDF5 requires an MPI compiler 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.

By default, CitComS will attempt to auto-detect your PHDF5 installation, and will disable HDF5 support if it is not found. You may explicitly 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 --prefix=$HOME/cig
```

2.7.1 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.

2.7.1.1 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.

2.7.1.2 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.

2.7.1.3 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`).

2.7.1.4 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.

2.8 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.

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 (`www.geodynamics.org/cig/software/packages/cs/pythia`). For more information on batch system configuration as it pertains to running CitComS, see Section 3.6.2 on page 36.

2.9 Installing without Pyre

To build just the CitComS tools (or "drivers") from the legacy C code, give `configure` the `--without-pyre` option:

```
$ ./configure --without-pyre
```

The only system requirements for this configuration are an MPI library and a C compiler. The `make` command will build two command-line tools, `CitcomSFull` and `CitcomSRegional`, for running the full solver and the regional solver, respectively.

2.10 Installing from the Software Repository

The CitComS source code is available via a Subversion server at the Geodynamics website (`www.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.

2.10.1 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 v2.2.1 source package was created with Autoconf 2.59, Automake 1.9.2, and Libtool 1.5.6.

2.10.2 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.

2.10.3 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.py

3.1 Using CitComS without Pyre

When you build CitComS, two binary executables, `CitcomSRegional` and `CitcomSFull`, are placed under the `bin` directory. These programs do not use Python or the Pyre framework (even if CitComS was configured to use 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 73.

3.2 Using CitComS with Pyre

If you build CitComS.py 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 6.2 on page 53 for an example).

On input, CitComS.py needs numerous parameters to be specified (see Appendix A on page 73 for a full list). All parameters have sensible default values. Since you will likely want to specify the parameters of your CitComS.py 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.py have names which are identical to the parameters for the old CitComS, which are described in Appendix A on page 73.

3.3 Changing Parameters

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

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 further here. But if you are interested in using `.pml` files, note that `.pml` files and `.cfg` files can be used interchangeably; in the following discussion, a file with a `.pml` extension can be substituted anywhere a `.cfg` file can be used.

3.3.4 Specification and Placement of Configuration Files

Configuration files may be specified on the command line:

```
$ citcoms example.cfg
```

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. Configuration files given on the command line override all others. 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 Coordinate System and Mesh

In general, CitComS uses meshes that are regular, although considerable flexibility exists for grid refinement in the regional models. 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 3.1 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.

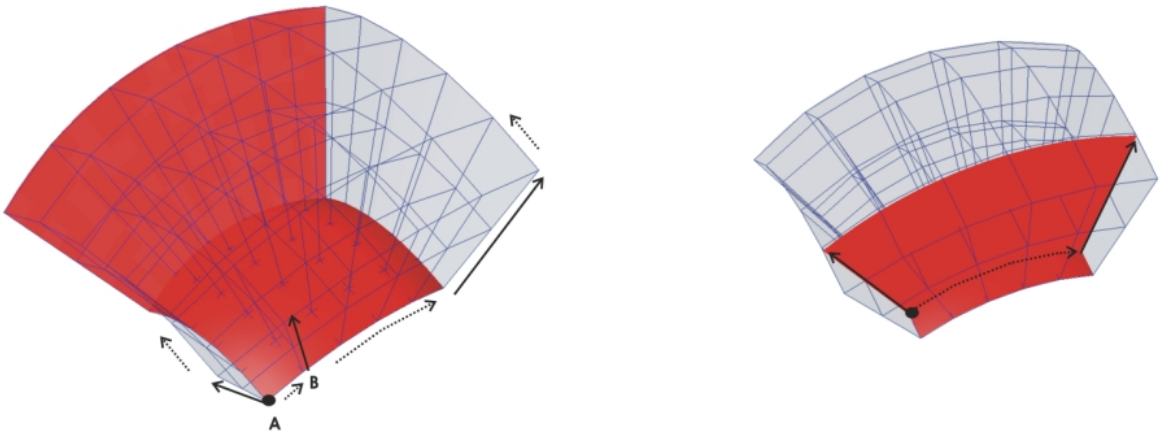


Figure 3.1: Global Node Numbering. Left: Global node numbering starts at the base of arrow A (theta_min, fi_min, radius_inner), and advances from 1 at the base to *nodez* at the tip. Upon reaching the tip, numbering continues from the base of arrow B (*nodez*+1) to its tip (2 *nodez*), and so on for 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.

3.5 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 × 17 × 9 nodal points. 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.py 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 the previous example: `example0.cfg`. You can run the model using:

```
$ citcoms example0.cfg
```

3.5.0.1 Example: Uniprocessor, `example0.cfg`

```
[CitcomS]
steps = 5
```

```
[CitcomS.controller]
monitoringFrequency = 1

[CitcomS.solver]
datafile = example0

[CitcomS.solver.mesher]
nprocx = 1
nprocy = 1
nodex =  9
nodey =  9
nodez =  9
```

3.6 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.py has been extensively used on both environments, using up to several hundred processors. How to run a multiprocessor CitComS.py 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.

3.6.0.2 Example: Multiprocessor, `example1.cfg`

```
[CitcomS]
steps = 71

[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 71 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
```

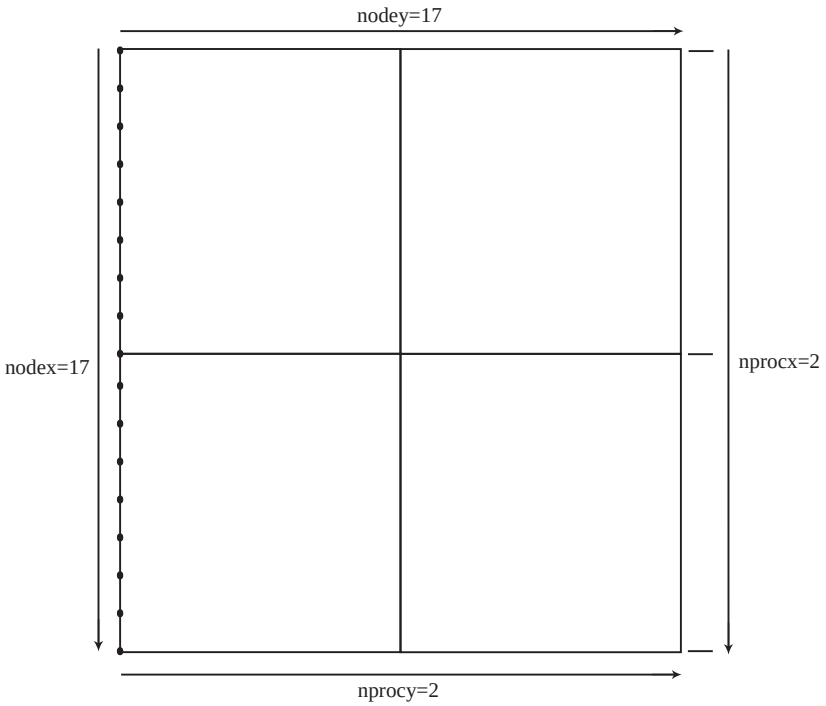


Figure 3.2: Computational Domain. Map view on the configuration of the top layer of the computational nodes and the processors.

3.6.1 Output Directories and Output Formats

CitComS.py potentially generates a large number of ASCII files. This means that you will have to organize your directories carefully when running CitComS.py 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.py 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.py 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 last choice is the most powerful one. Instead of writing many ASCII files, CitComS.py can write its results into a single HDF5 (Hierarchical Data Format) file per timestep. 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.py with the parallel HDF5 library if you haven't done so already (see Section 2.7 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 39 for more information on how to work with the HDF5 output.

3.6.2 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.py 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.)

3.6.2.1 Running without a Batch System

On a cluster without a batch system, you need to explicitly specify the machines on which 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.py has run. Save the machine file as it will be useful in the postprocessing step.

3.6.2.2 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
```

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.

3.6.3 Monitoring Your Jobs

Once launched, CitComS.py 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. The log file contains the convergence progress of the computation and, if an error occurs, debugging output. The time file 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 93. 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 45).

3.7 Using CitComS.py 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. The TeraGrid takes its name from two concepts from high-end computing: "Tera" is the metric prefix for "trillions" as in teraflops (trillions of calculations per second) and terabytes (trillions of bytes) and reflects the scale of the computing power provided by the TeraGrid; the "Grid" portion of TeraGrid reflects the idea of harnessing and using distributed computers, data storage systems, networks, and other resources as if they were a single massive system. In other words, Grid computing uses software technologies to allow researchers to create "virtual supercomputers" far larger than individual hardware components. 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.py has already been installed and tested on several NSF TeraGrid platforms, includes NCSA, SDSC and TACC. To use CitComS.py 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 (`www.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 45). The CitComS.py 2.1 release 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.

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.

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, although in future versions of CitComS this step may become unnecessary for supported file systems.

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 10). 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.4.9 on page 94 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 C.4 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 C.4.2 on page 93 for restart purpose 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 for using the PyTables extension module to access the data contained in the CitComS HDF5 files. In 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:

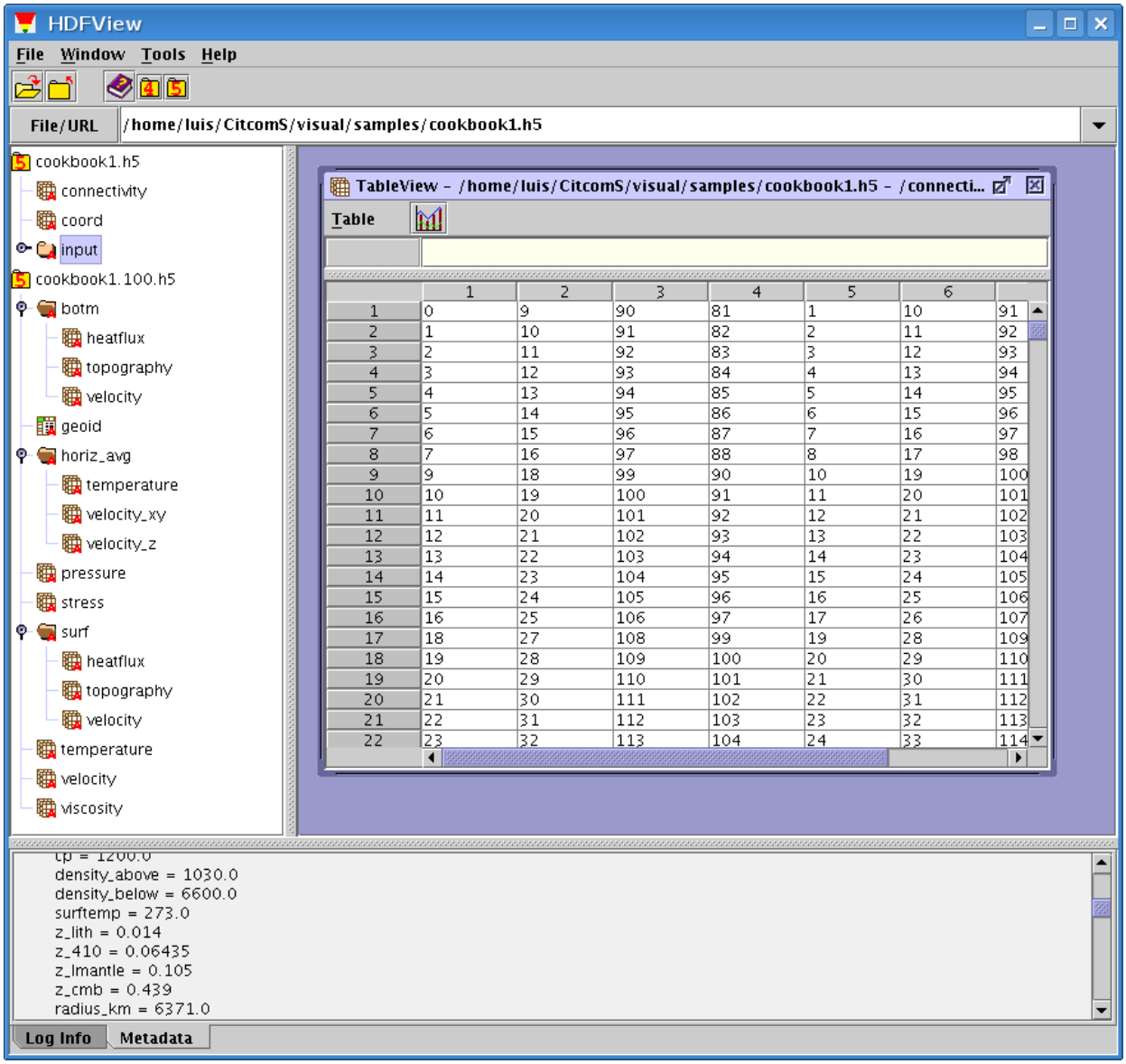
```
import tables

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 written 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 6.2 on page 53.



	1	2	3	4	5	6	
1	0	9	90	81	1	10	91
2	1	10	91	82	2	11	92
3	2	11	92	83	3	12	93
4	3	12	93	84	4	13	94
5	4	13	94	85	5	14	95
6	5	14	95	86	6	15	96
7	6	15	96	87	7	16	97
8	7	16	97	88	8	17	98
9	9	18	99	90	10	19	100
10	10	19	100	91	11	20	101
11	11	20	101	92	12	21	102
12	12	21	102	93	13	22	103
13	13	22	103	94	14	23	104
14	14	23	104	95	15	24	105
15	15	24	105	96	16	25	106
16	16	25	106	97	17	26	107
17	18	27	108	99	19	28	109
18	19	28	109	100	20	29	110
19	20	29	110	101	21	30	111
20	21	30	111	102	22	31	112
21	22	31	112	103	23	32	113
22	23	32	113	104	24	33	114

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.py, you should have a series of output files (potentially spread throughout the file systems of your Beowulf 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 49. Scripts using GMT commands to plot 2D cross sections of temperature field are also provided; see Section 5.8 on page 50.

5.2 Postprocessing on a Beowulf Cluster

Generally, the results from your CitComS.py 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 48 for more details.

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` field 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.py 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.py need to be postprocessed before you can perform the visualization. The script `autocombine.py` can postprocess (retrieve and combine) CitComS.py output; it will retrieve CitComS.py 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.py 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 `batchpaste.sh` and replace 'rsh' with 'ssh' in the script.

Once `autocombine.py` has run, you will have with 2 files (or 24 files for the full spherical version of CitComS.py) 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 on page 93. The latter file is the OpenDX header for the data.

5.3 Postprocessing in a Non-Cluster Environment

If you run CitComS.py 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.py 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.py.

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.

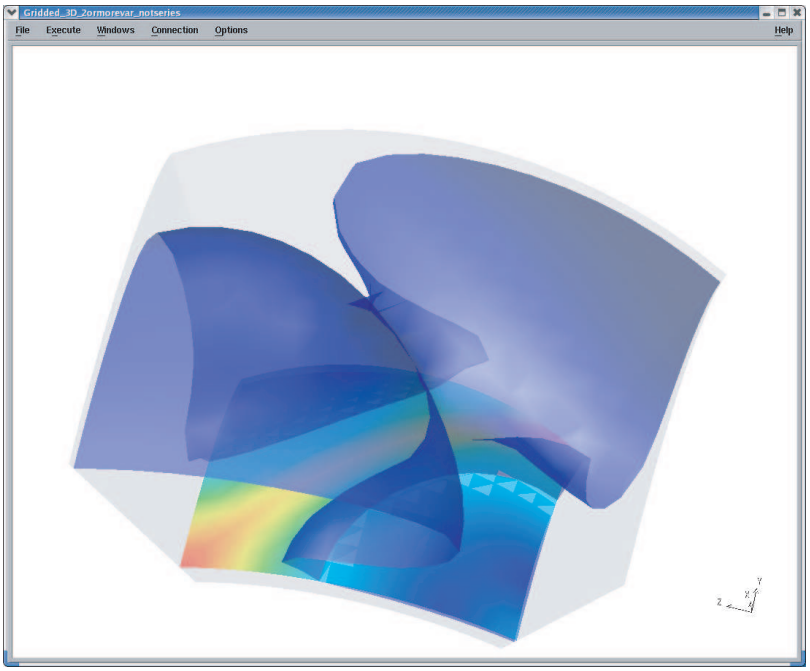


Figure 5.1: Regional Model Visualized with OpenDX. A snapshot of an upwelling (blue isosurface) with a slice of the temperature field (bisecting plane).

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.
- **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.py 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.
- **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 2.7.1.4 on page 28). 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 5.2).

3. Place the module in the work space and rewire the network as shown in Figure 5.2.

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.

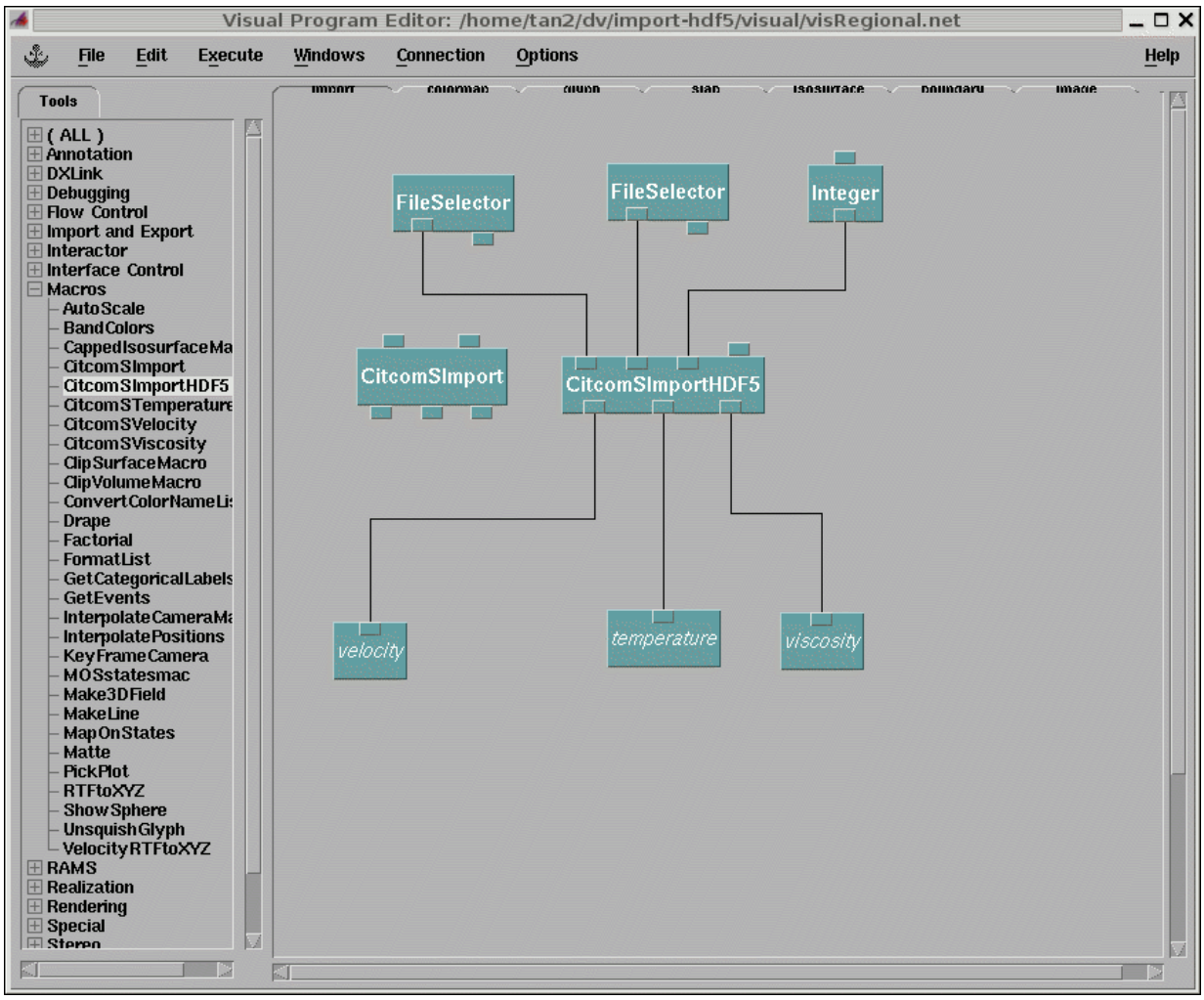


Figure 5.2: How to import CitComS.py HDF5 data. The CitcomSImportHDF5 module is highlighted in the Tools panel.

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, you will need to install PyTables (see Section 2.7.1.2 on page 28) and PyVTK (see PyVTK website (`cens.ioc.ee/projects/pyvtk`) for instructions) before using the script.

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 5.3. 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`).

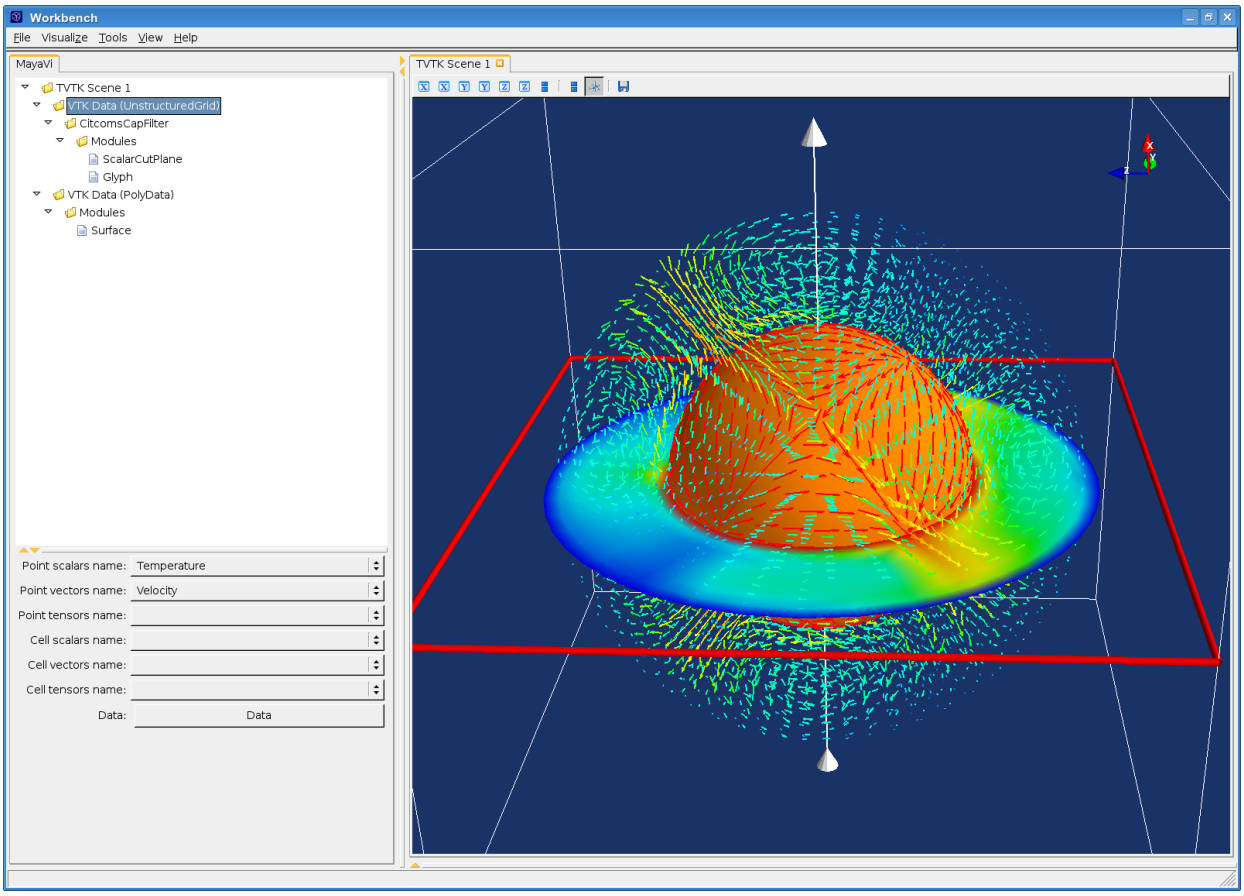


Figure 5.3: Example MayaVi2 visualization of `cookbook1.100.h5`. Here we display the velocity field as vectors, as well a slice of the temperature field. Note the visualization pipeline in the tree view on the left panel.

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.

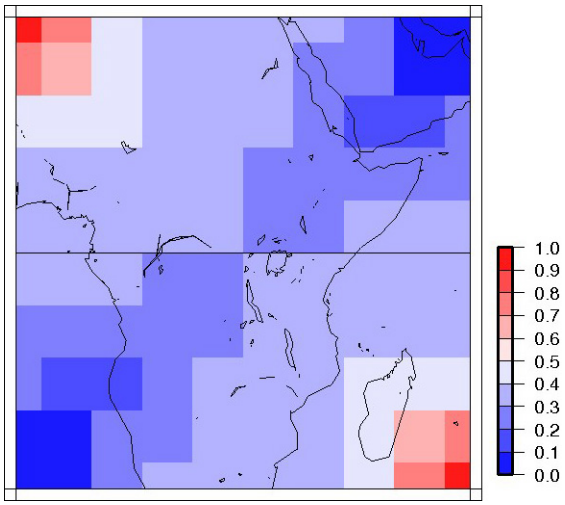


Figure 5.4: Temperature field generated by `plot_layer.py`. This image is the mid-layer (layer 5) of the sample data `visual/samples/regtest.cap00.100`. The resolution of the image is poor because the data has only 9×9×9 nodes.

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.

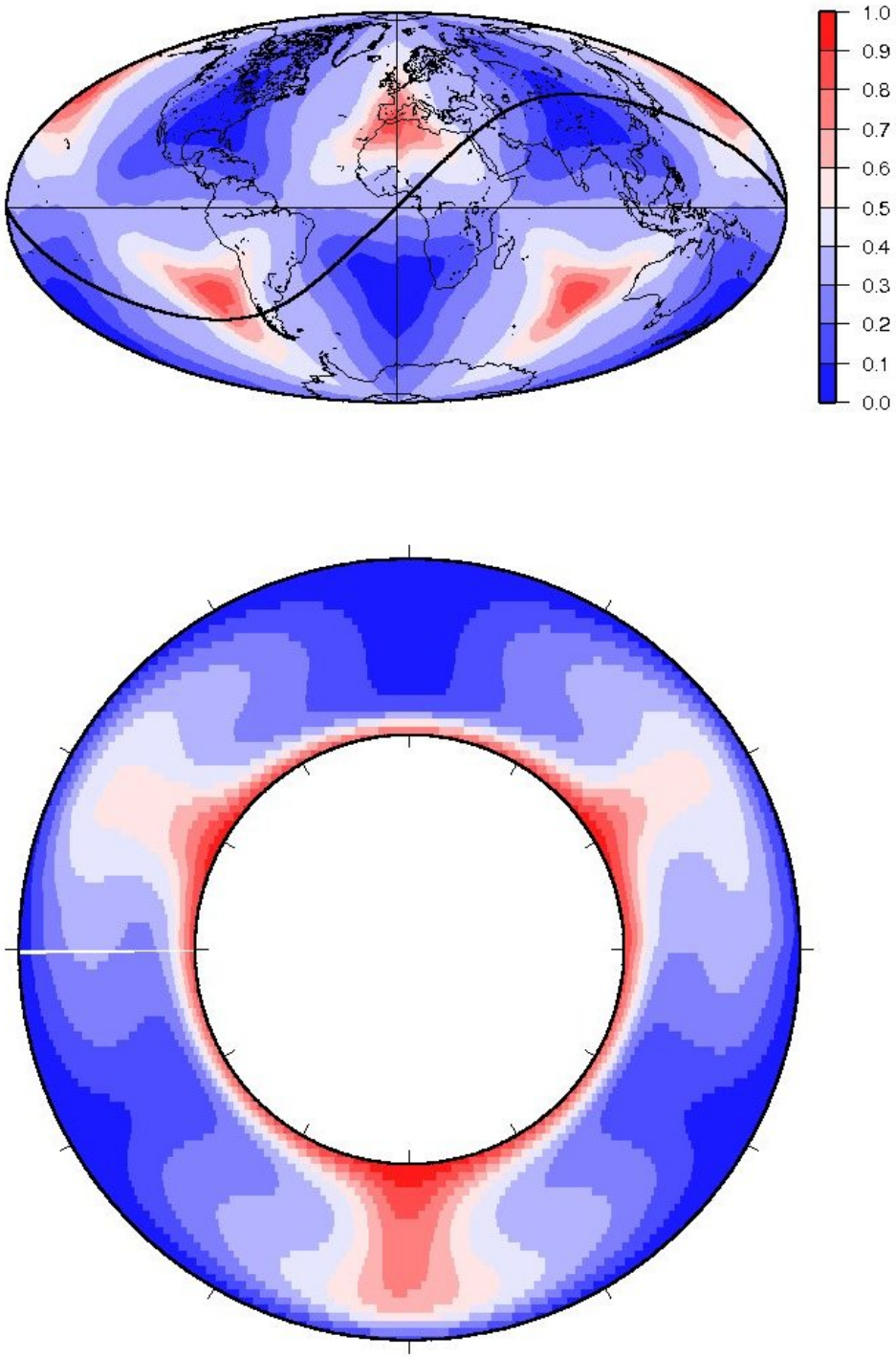


Figure 5.5: Temperature fields generated by `plot_annulus.py`. The top panel is the horizontal cross section at the mid layer in Hammer-Aitoff projection. The thick line marks the surface track of the radial cross section. The bottom panel is the radial cross section. The cross section starts at 0° latitude and 0° longitude and has an azimuth of 45°. The image is from the sample data `visual/samples/fulltest.cap*.100`. There is always a gap on the left (starting point) of the cross section. Users familiar with GMT are welcome to contribute a fix to the script.

Chapter 6

Cookbooks

6.1 Introduction

These cookbook examples are meant to serve as a guide to some of the types of problems CitComS.py can solve. 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 3.6 on page 34 for more information).

6.2 Cookbook 1: Global Model

6.2.1 Problem

This example solves for thermal convection within a full spherical shell domain. The full spherical version of CitComS.py is designed to run on a cluster that decomposes the spherical shell into 12 equal "caps" and then distributes the calculation for caps onto separate processors. To run CitComS.py with the full solver parameter set, it is recommended that you have a minimum of 12 processors available on your cluster.

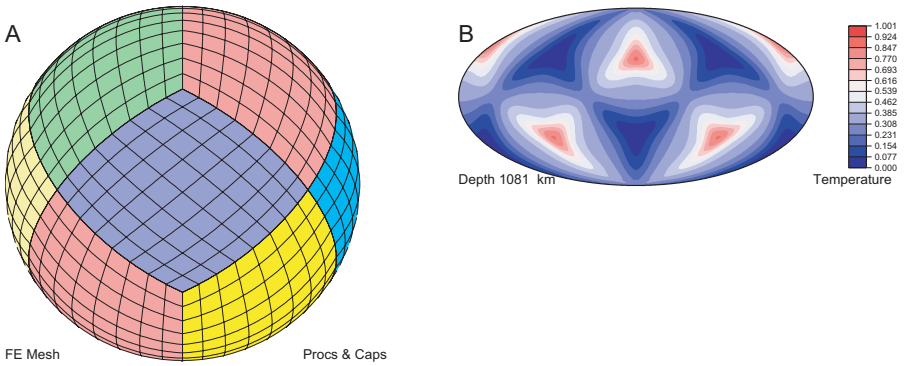


Figure 6.1: Global Model "Caps." Left (A): Three-dimensional perspective image showing seven of the 12 spherical caps used in a full CitComS.py run. Right (B): The temperature field at 1081 km depth from a Cookbook 1 run.

6.2.2 Solution

You will use `cookbook1.cfg`. The first set of parameters specifies that you are going to use the full spherical version of the solver. The default is to use the regional spherical version, so this must be set.

```
solver = full
```

The second set of parameters specifies the number of time steps (101), how often full outputs of the computation are created (25), and the prefix of output filenames (cookbook1).

```
steps = 101
monitoringFrequency = 25
datafile = cookbook1
```

The `solver.ic` facility controls the temperature field for the initial conditions. The last set of parameters includes the number of perturbations to the initial temperature (1), the number of nodal lines of the perturbation in the longitudinal direction, e.g., the spherical harmonic order (3), the number of nodal lines in the latitudinal direction, e.g., the spherical harmonic degree (2), which layer the pertubation is on (5), and the amplitude of the perturbation (0.05). Note that although the number of perturbations is assigned here as `num_perturbations=1`, that is actually the default value.

```
num_perturbations = 1
perturbl = 3
perturbm = 2
perturblayer = 5
perturbmag = 0.05
```

This example is executed by typing

```
$ citcoms cookbook1.cfg
```

6.2.2.1 Example: Global Model, `cookbook1.cfg`

```
[CitcomS]
solver = full
steps = 101                 ; number of time steps

[CitcomS.controller]
monitoringFrequency = 25    ; how often outputs are created

[CitcomS.solver]
datafile = cookbook1        ; prefix of output filenames

[CitcomS.solver.ic]
num_perturbations = 1
perturbl = 3
perturbm = 2
perturblayer = 5
perturbmag = 0.05
```

Once you have run the model, you can visualize the results using OpenDX, described in the previous chapter. When you invoke "Edit Visual Program," select `visFull.net`.

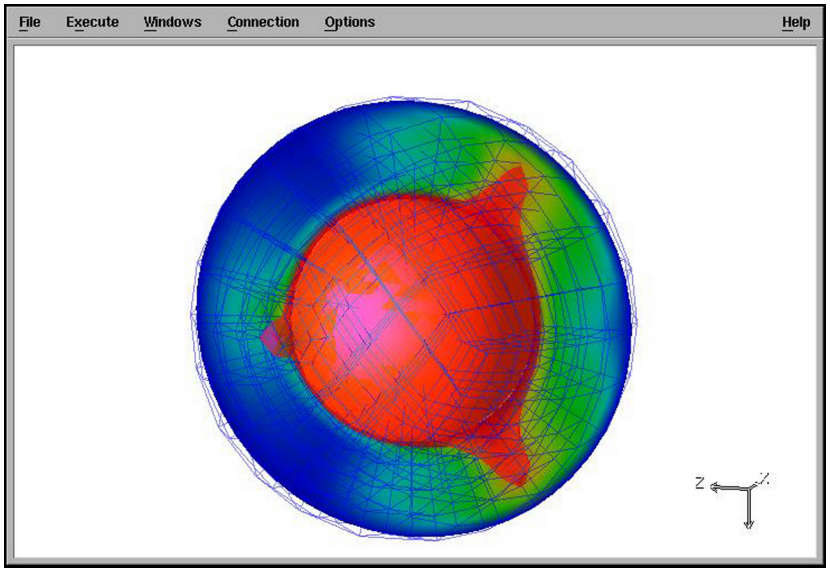


Figure 6.2: Cookbook 1: Global Model. This image, created by OpenDX, depicts a slice through a spherical model of convection, with warmer colors indicating upwelling and the cooler colors showing downwelling.

6.2.3 Discussion

You have generated a simple example of the full CitComS.py model, with minimal parameter alterations. With a default Rayleigh number of 10^5 and perturbation on the initial temperature field which has a degree-3 and an order-2 pattern, after 100 time steps, the convection pattern remains relatively steady.

As a side note, it is not required that 12 processors, with one spherical cap per processor, be used. As an end-member possibility, for example, 12 different jobs could be run on a single computer (`n001` in this example) by invoking:

```
$ citcoms cookbook1.cfg --launcher.nodegen="n%03d" \
--launcher.nodelist=[1,1,1,1,1,1,1,1,1,1,1,1]
```

This is not particularly efficient, but it does illustrate the flexibility of both `mpi` and Pyre.

6.3 Cookbook 2: Velocity Boundary Conditions

6.3.1 Problem

This example solves for thermal convection with velocity boundary conditions imposed on the top surface within a given region of a sphere. This requires using the regional version of CitComS.py.

6.3.2 Solution

You will use `cookbook2.cfg`. The parameters specify the number of time steps (61), the prefix of the output filenames (`cookbook2`), and how often outputs will be created (30).

```
steps = 61
monitoringFrequency = 30
datafile = cookbook2
```

The `solver.mesher` facility has several properties involved in the generation of the computational mesh. Continue to use the default values for the physical portion of the domain in which you are interested. However, try modifying the layout of the mesh as shown.

```
nprocx =  2
nprocy =  2
nodex  = 17
nodey  = 17
nodez  =  9
```

The `solver.bc` facility allows you to impose a uniform velocity across the top surface. You have a velocity which is purely in the colatitude direction with a non-dimensional velocity of 100 (see Equation 1.6 on page 19 for how to dimensionalize it).

```
topvbc    =   1
topvbxval = 100
```

In addition, the initial temperature field has a linear conductive profile. The amplitude of initial temperature perturbation is set to zero using the `solver.ic` facility.

```
num_perturbations = 1
perturbmag        = 0.0
```

6.3.2.1 Example: Velocity Boundary Conditions, `cookbook2.cfg`

```
[CitcomS]
steps = 61                  ; number of time steps

[CitcomS.controller]
monitoringFrequency = 30    ; how often outputs are created

[CitcomS.solver]
datafile = cookbook2        ; prefix of output filenames

# Modify the layout of the mesh.
[CitcomS.solver.mesher]
nprocx =  2
nprocy =  2
nodex  = 17
nodey  = 17
nodez  =  9

# Impose a uniform velocity across the top surface.
[CitcomS.solver.bc]
topvbc    =   1
topvbxval = 100

# Modify the layout of the mesh.
[CitcomS.solver.mesher]
nprocx =  2
nprocy =  2
nodex  = 17
nodey  = 17
nodez  =  9

# Impose a uniform velocity across the top surface.
[CitcomS.solver.bc]
topvbc    =   1
topvbxval = 100
```

```
topvbyval =   0

# In addition, set the initial temperature perturbation to zero.
[CitcomS.solver.ic]
num_perturbations = 1
perturbmag        = 0.0
```

6.3.3 Discussion

Using OpenDX to visualize the results (Figure 6.3), this model allows you to create a plate-driven convection in which there is a thermal upwelling on one wall, a thermal downwelling on another, and uniform horizontal velocity across the top. The downwelling is not exactly subduction because the default boundary conditions are close to zero shear stress on the boundaries. This means that there is a symmetrical downwelling in a vertical domain on the other side.

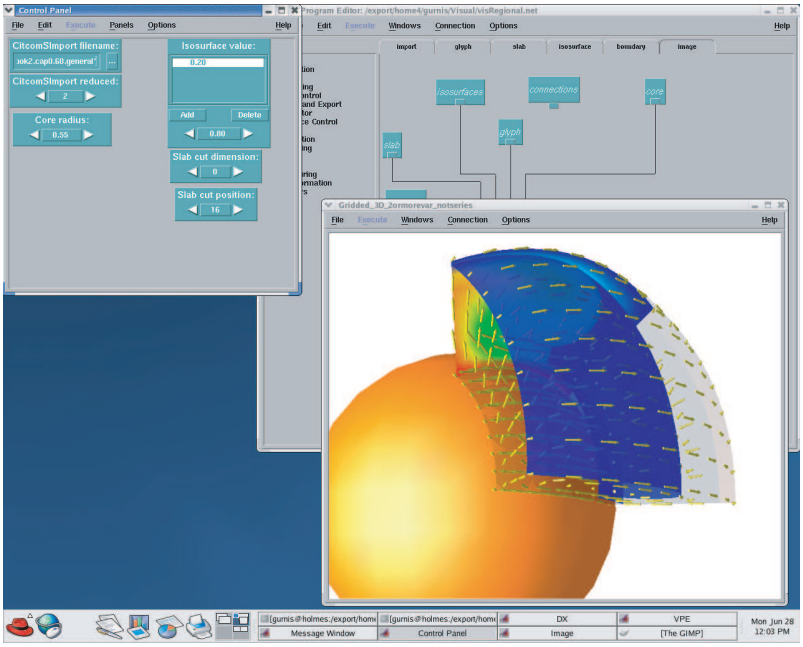


Figure 6.3: Cookbook 2: Velocity Boundary Conditions. This model, visualized with OpenDX, highlights a region of the sphere and the heated upwellings (warm colors), downwellings (cool colors), and the velocities (yellow arrows).

6.4 Cookbook 3: Temperature-Dependent Viscosity

6.4.1 Problem

A common problem in geophysics is the exploration of natural convection in the presence of variable viscosity, including temperature-dependent or stress-dependent viscosity.

6.4.2 Solution

You will use `cookbook3.cfg`. The parameters specify the number of time steps (200), how often outputs will be created (25), the prefix of the output filenames (`cookbook3`), and the Rayleigh number (10^6).

```
steps = 200
```

```
monitoringFrequency = 25
datafile = cookbook3
rayleigh = 1e6
```

The `solver.visc` facility assigns the viscosities. The parameters specify whether the viscosity should be updated at every time step (`on`), the number of viscous layers (4), the viscosity of each layer (1,1,1,1), whether the viscosity is temperature dependent (`on`), the activation energy of each layer (0.2,0.2,0.2,0.2), the temperature offset of each layer (0,0,0,0), whether to apply the minimum cutoff (`on`), the value of the minimum cutoff (1.0), whether to apply the maximum cutoff (`on`), and the value of the maximum cutoff (100.0).

```
VISC_UPDATE = on
num_mat = 4
visc0 = 1,1,1,1
TDEPV = on
viscE = 0.2,0.2,0.2,0.2
viscT = 0,0,0,0
VMIN = on
visc_min = 1.0
VMAX = on
visc_max = 100.0
```

The range of the layers are determined by the following parameters: layer 1 extends from the top surface to a depth of `solver.const.z_lith` (default to 0.014, or 90 km dimensionally); layer 2 is below layer 1 and extends to a depth of `solver.const.z_410` (default to 0.06435, or 410 km dimensionally); layer 3 is below layer 2 and extends to a depth of `solver.const.z_lmantle` (default to 0.105, or 670 km dimensionally); layer 4 is below layer 3 and extends to the bottom. The temperature dependence of the viscosity of each layer is determined by:

$$visc = visc0 \times exp\left(\frac{viscE}{T + viscT} - \frac{viscE}{1 + viscT}\right) \tag{6.1}$$

6.4.2.1 Example: Temperature-Dependent Viscosity, `cookbook3.cfg`

```
[CitcomS]
steps = 200                 ; number of time steps

[CitcomS.controller]
monitoringFrequency = 25    ; how often outputs are created

[CitcomS.solver]
datafile = cookbook3        ; prefix of output filenames
rayleigh = 1e6              ; Rayleigh number

# Modify the layout of the mesh.
[CitcomS.solver.mesher]
nprocx =  2
nprocy =  2
nodex  = 17
nodey  = 17
nodez  =  9

# Assign the viscosities.
[CitcomS.solver.visc]
VISC_UPDATE = on
```

```
num_mat = 4
visc0 = 1,1,1,1
TDEPV = on
viscE = 0.2,0.2,0.2,0.2
viscT = 0,0,0,0
VMIN = on
visc_min = 1.0
VMAX = on
visc_max = 100.0
```

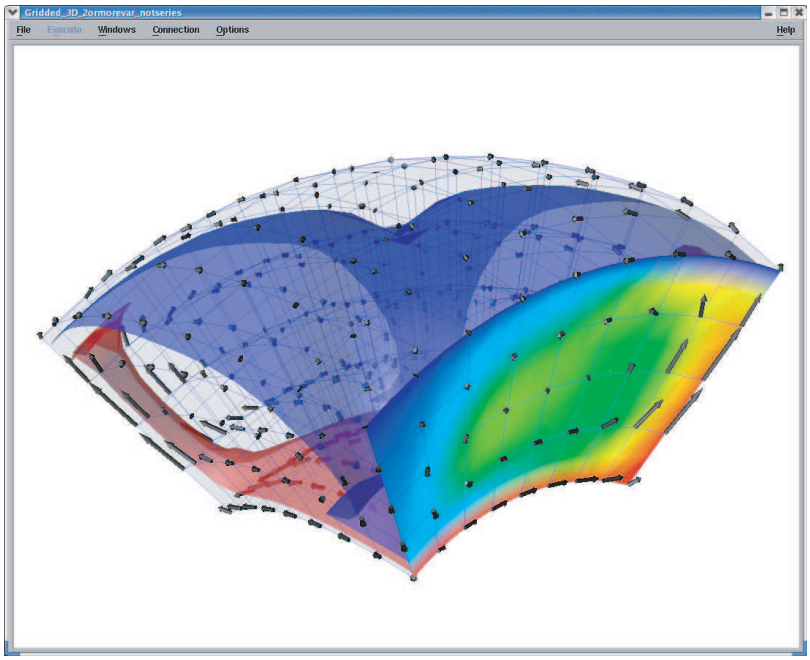


Figure 6.4: Cookbook 3: Temperature-Dependent Viscosity. This model, visualized with OpenDX, highlights a region that features both the heated upwelling (warm colors) and the even distribution of the velocities (arrows).

6.5 Cookbook 4: Regionally Refined Meshes

6.5.1 Problem

Frequently for convection problems, particularly those with variable viscosity, there are features with strong gradients in temperature or viscosity that can be better resolved with refined meshes. For example, for the problem just studied in Cookbook 3, temperature-dependent viscosity, a higher resolution is required for a narrow hot upwelling while a lower resolution is sufficient for the wider cold downwelling.

6.5.2 Solution

The parameter that controls whether a mesh is to be uniform or refined is `solver.mesher.coor`. Set it to `on` (`solver.mesher.coor=on`) in order to read the uneven mesh point coordinates from an input file (specified by `coor_file`). The format of the coordinate input file is described in Appendix B.2 on page 89.

```
coor = on
coor_file = coor.dat
```

The computational domain is bounded in colatitude between radian 1 and 2, in latitude between radian 0 and 1, and in radius between 0.55 and 1.

```
theta_min = 1
theta_max = 2
fi_min = 0
fi_max = 1
radius_inner = 0.55
radius_outer = 1
```

6.5.2.1 Example: Regionally Refined Meshes, `cookbook4.cfg`

```
[CitcomS]
steps = 250                 ; number of time steps

[CitcomS.controller]
monitoringFrequency = 10    ; how often outputs are created

[CitcomS.solver]
rayleigh = 1e6              ; Rayleigh number
datafile = cookbook4        ; prefix of output filenames

[CitcomS.solver.ic]
num_perturbations = 1
perturbmag = 0.05
perturblayer = 10

# Perturb the initial temperature gradient in the longitudinal
# direction.
perturbl = 1
perturbm = 0

[CitcomS.solver.mesher]
# Read uneven mesh point coordinates from 'coor.dat'.
coor = on
coor_file = coor.dat

nprocx = 4
nprocy = 2
nprocz = 2
nodex = 33
nodey = 17
nodez = 17
theta_min = 1
theta_max = 2
fi_min = 0
fi_max = 1
radius_inner = 0.55
radius_outer = 1

[CitcomS.solver.visc]
VISC_UPDATE = on
num_mat = 4
visc0 = 1,1,1,1
TDEPV = on
```

```
viscE = 0.2,0.2,0.2,0.2
viscT = 0,0,0,0
VMIN = on
visc_min = 1.0
VMAX = on
visc_max = 100.0
```

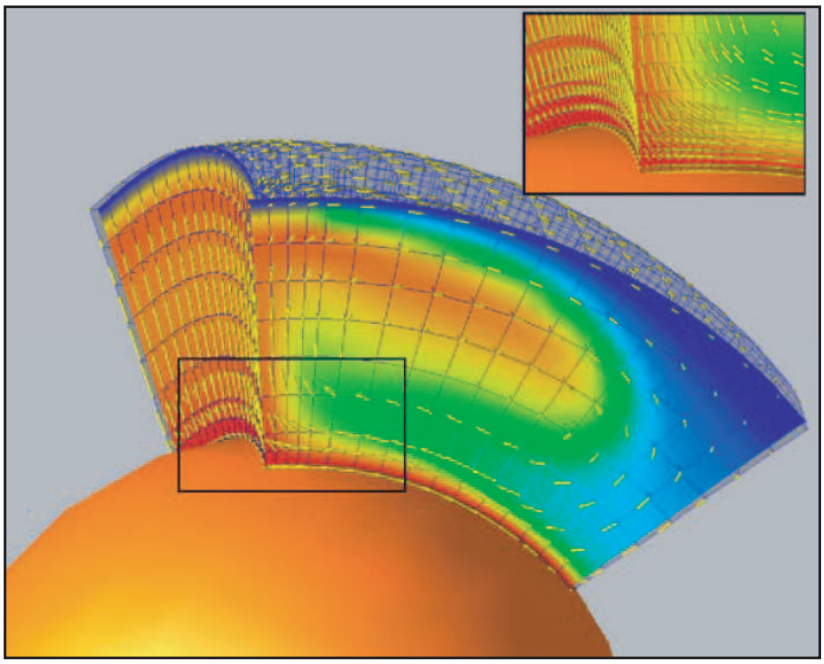


Figure 6.5: Cookbook 4: Regionally Refined Mesher. This model shows the temperature (upwelling – warm colors, downwelling – cool colors) and the uneven distribution of the velocities (yellow arrows). Note the refined mesh for the left and bottom regions of the model (inset).

6.5.3 Discussion

The resulting model is like a 2D model, but extends in the longitudinal direction. To set up this type of model, the initial temperature gradient was perturbed only in the longitudinal direction, by setting the parameters `perturbl=1` and `perturbm=0`. The model results show a thin thermal upwelling on one wall and a wide thermal downwelling on the opposite wall. Also, at the bottom of the model a thin layer of hot material can be shown. Note the higher resolution in the narrow regions with hot material and the lower resolution for the wide thermal downwelling region.

6.6 Cookbook 5: Subduction Models with Trench Rollback

6.6.1 Problem

A common issue to address is the problem arising when the position of the oceanic trench is not constant in time (trench rollback) for a subduction zone. In addition, the trench rollback speed may vary in time, which can be caused, for example, by a rotation pole jump.

6.6.2 Solution

In order to introduce in a convection model a trench rollback you have to prepare the velocity files. The following scenario is proposed for this cookbook: there are two plates centered along the equator, with

two different Euler poles (see Figure 6.6). Initially, both plates have Euler poles situated far apart, so the velocities are approximately constant (about 5 cm/yr for the left plate and about 2 cm/yr for the right plate). After 30 Ma, a pole jump occurred for the right slab only, so the Euler pole moved closer.

Since we impose a top velocity boundary in our model, the following parameters should be turned on:

```
topvbc = 1
file_vbcs = on
```

The following two lines specify the location of the velocity files and the starting age for the model:

```
start_age = 55
vel_bound_file = ./velocity/bvel.dat
```

Since the starting age is set to 55 Ma, there will be 57 velocity files, one for each Ma (`vel.dat0, vel.dat1, ...vel.dat55, vel.dat56`). The format of the velocity file is described in Appendix B.3 on page 90. Note that the velocity file is in a unit of cm/yr.

When `restart=on` is set, the initial temperature field is read from setup files. The files have the same naming scheme and format as the CitComS velo output, as described in Appendix C.4.2 on page 93. The location and prefix of the file is specified in `datadir_old` and `datafile_old`, and the time step of the velo file in `solition_cycles_init`. In this example, a processor of MPI rank `n` will read file `./restart_files/cookbook5.velo.n.0`

```
datadir_old = ./restart_files
datafile_old = cookbook5
restart = on
solution_cycles_init = 0
```

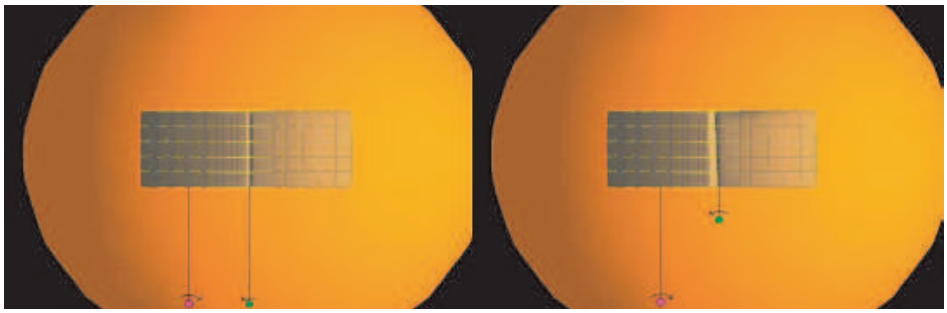


Figure 6.6: Cookbook 5: Left (A): Initially, both plates have Euler poles (magenta and green dots) situated far apart from the plate, so the velocities are approximately constant (about 5 cm/yr for the left plate and about 2 cm/yr for the right plate). Right (B): After 30 Ma a pole jump occurred for the right slab only, so the Euler pole moved closer.

6.6.2.1 Example: Subduction Models with Trench Rollback, `cookbook5.cfg`

```
[CitcomS]
steps = 1000                ; number of time steps

[CitcomS.controller]
monitoringFrequency = 10    ; how often outputs are created

[CitcomS.solver]
datafile = cookbook5
datadir_old = ./restart_files
datafile_old = cookbook5
```

```
rayleigh = 4.07e+08

[CitcomS.solver.bc]
topvbc = 1

[CitcomS.solver.param]
file_vbcs = on
start_age = 55

# Since the starting age is set to 55 Ma, there will be 56 velocity
# files, one for each Ma (bvel.dat0, bvel.dat1, ... bvel.dat56).
vel_bound_file = ./velocity/bvel.dat

[CitcomS.solver.ic]
restart = on
solution_cycles_init = 0

[CitcomS.solver.mesher]
coor = on
coor_file = ./coor.dat
nprocx = 2
nprocy = 8
nprocz = 4
nodex = 17
nodey = 65
nodez = 33
theta_min = 1.47
theta_max = 1.67
fi_min = 0
fi_max = 0.5
radius_inner = 0.7

[CitcomS.solver.visc]
num_mat = 4
visc0 = 100,0.003,1,2
TDEPV = on
viscE = 24,24,24,24
viscT = 0.182,0.182,0.182,0.182
VMIN = on
visc_min = 0.01
VMAX = on
visc_max = 100.0
```

6.6.3 Discussion

The results for this problem are presented in Figure 6.7. Since the two Euler poles are kept fixed for the first 30 Ma, the shape of the subducting slab will be the same along the trench. At 25 Ma, the Euler pole for the right plate jumps toward the plate. Therefore, the velocity along the trench varies from about 1 cm/yr to about 2.5 cm/yr. This will produce in time a flat slab at one side of the model, and a steep slab at the other side.

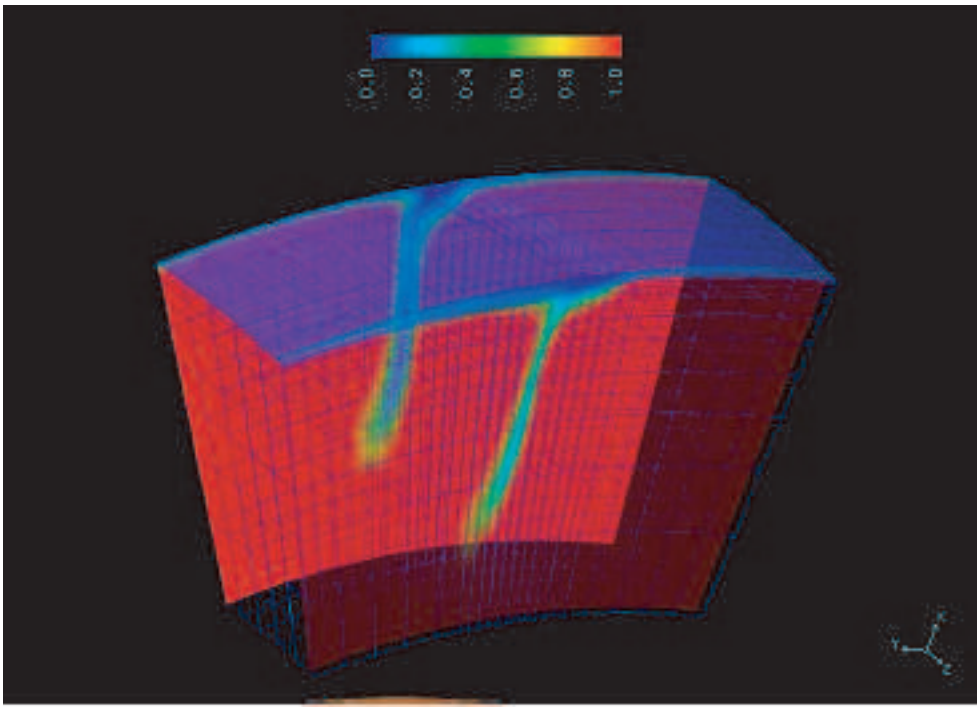


Figure 6.7: Cookbook 5: The pole jump after 30 Ma produced a flat slab on one side (fore side) and a steep slab on the other side (back side).

6.7 Cookbook 6: Pseudo-Free-Surface Formulation

6.7.1 Problem

Free-slip boundary conditions are typically applied on the top surface of mantle convection models, and the dynamic topography is obtained by assuming that the normal stress on the top surface is instantaneously compensated by the deformed surface. This type of boundary condition works well for long-wavelength topography, but a free-surface formulation becomes necessary in cases where intermediate to short wavelength topography is of interest or the lithosphere has a very high effective viscosity. Another situation where free-surface formulation is desired is when two Pyre solvers, Snac and CitCom.py, are coupled.

The basic algorithm [15, 16] consists of four steps.

1. On the nodes of the top surface, compute the topography increment by integrating normal velocity over time.
2. Calculate the normal traction on the top surface based on the accumulated topography up to the current time step, and add it to the forcing term in the matrix version of the momentum equation.
3. Update the velocity field with the changed forcing term.
4. If the velocity field has not converged yet, repeat steps 1 to 3.

6.7.2 Solution

To verify that the above algorithm works, you will run two different CitComS.py models with each boundary condition (BC) (free-slip and pseudo-free-surface) and compare the topography computed accordingly. The scripts in `cookbook6.cfg` will include the following parameters.

- Domain size: $45° \times 45° \times 1200\,\text{km}$

- Mesh size: $61 \times 61 \times 25$ with mesh refinement (coord.dat)
- Boundary Conditions (BC): free-slip/free-surface for top; free-slip for all the other surfaces
- Initial Conditions (IC): spherical hot blob with diameter of $\frac{1}{4}$ of radial dimension is placed at the center of the domain

Specific options:

- `solver.tsolver.fixed_timestep = 7.770000e-10` (= ~1000 yrs)
- `solver.bc.pseudo_free_surf = on`

6.7.3 Discussion

The plots of the topography profile are consistent with results described in [15]. In the graphs in Figure 6.8, the solid lines indicate free-slip and the dashed lines indicate free-surface.

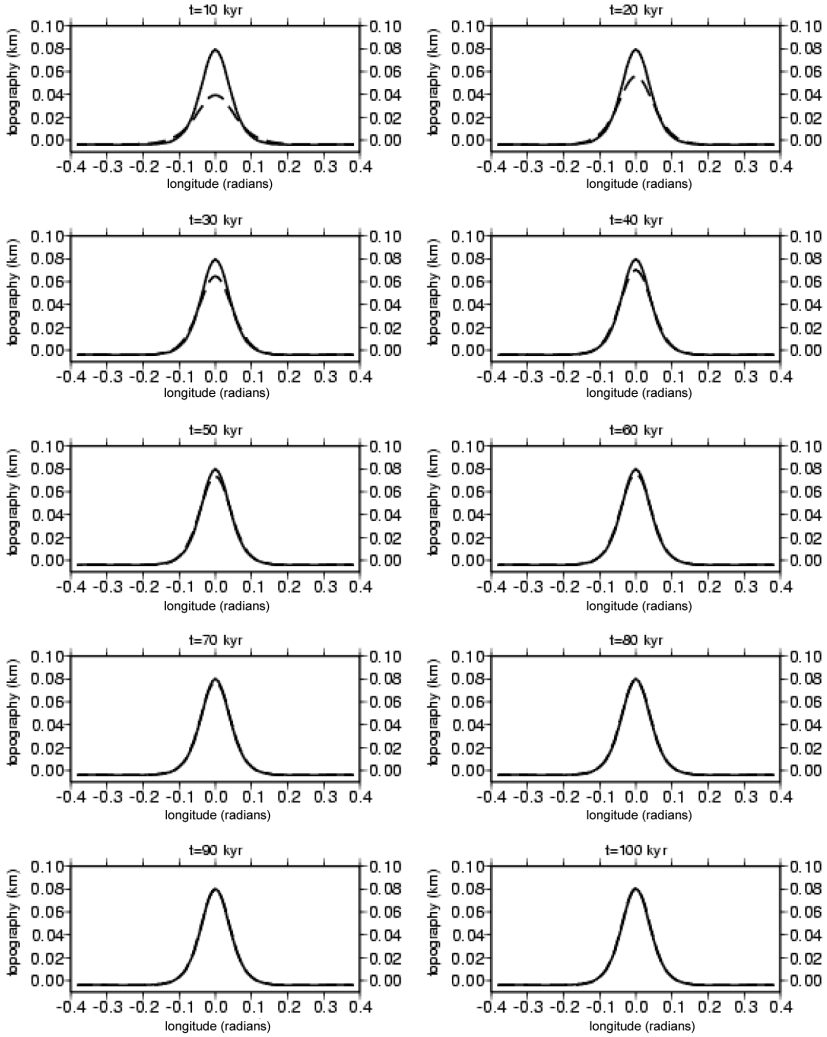


Figure 6.8: Cookbook 6: Graphs of topography profiles.

6.8 Cookbook 7: Thermo-Chemical Convection

6.8.1 Problem

This example solves for thermo-chemical convection within a full spherical shell domain. Composition heterogenity exists in the Earth mantle. The density anomalies due to the composition heterogenity, as well as due to the thermal heterogeneity, drive the convection flow.

6.8.2 Solution

You will use `cookbook7.cfg`. Most of the parameters you have encountered in previous cookbooks. The initial condition is the same as in Cookbook 1, and the viscosity law is the same as in Cookbook 3.

A large number of ASCII files will be output. In order to keep the current directory tidy, the output files can be put in a subdirectory `output/` by specifying

```
datadir = output
```

Since you are interested in the composition field, the compostion output should be enabled and some output files disabled:

```
output_optional = tracer,comp_nd
```

The most important parameters are in the `CitcomS.solver.tracer` section. Turn on the tracer module:

```
tracer = 1
```

The number of tracers and their initial location must be set. Here, you specify `tracer_ic_method=0`. The tracers will be generated pseudo-randomly automatically, with a total number equal to

`tracers_per_element`×(total number of finite elements).

If `tracer_ic_method=1`, all processors will read the location (and flavors; see next paragraph) of the tracers from the same file specified by `tracer_file`. If `tracer_ic_method=2` or `restart=on` (in the `CitcomS.solver.ic` section), each processor will read the location (and flavors; see next paragraph) of the tracers from the file specified by `datafile_old` (in the `CitcomS.solver` section) and `solution_cycles_init` (in the `CitcomS.solver.ic` section).

```
tracer_ic_method = 0
tracers_per_element = 20
tracer_file = tracer.dat
```

Each tracer can have a "flavor" attached to it. The meaning of the flavor depends on the application. Here, the flavor indicates the chemical species of the tracer. There are two flavors of tracers (`tracer_flavors=2`). A tracer of flavor 0 is of the "normal," or ambient, chemical composition, while a tracer of flavor 1 is of the "anomalous" chemical composition. Because the tracers are automatically generated, you need to specify how to assign the flavor to each tracer. If `ic_method_for_flavors=0`, tracers above `z_interface` are assigned to flavor 0, and tracers below `z_interface` to flavor 1.

```
tracer_flavors = 2
ic_method_for_flavors = 0
z_interface = 0.7
```

The thermo-chemical convection module is turned on by

```
chemical_buoyancy = 1
```

The composition field is determined by the ratio method (`buoy_type=1`) [2]. The density anomaly of the anomalous chemical composition is 0.4.

```
buoy_type = 1
buoyancy_ratio = 0.4
```

The code keeps track of the sum of the bulk composition. In a perfect world, the sum of the bulk composition would not change with time. But due to numerical issues, the sum of the bulk composition tends to decrease with time. If `tracer_ic_method=2` or `restart=on` (in the `CitcomS.solver.ic` section), the code will read in the location of the tracers, as well as the initial sum of the bulk composition, from the previous run. One can reset the initial sum of the bulk composition with the current sum by turning on this parameter:

```
reset_initial_composition = 0
```

The regular grid is an auxiliary grid that helps locate the tracers [1]. The optimal grid spacing of the regular grid depends on the size of the CitComS mesh. A general rule of thumb is that the grid spacing of the regular grid should be less than $18/(nodex \times nprocx)$. These parameters are used only in the full spherical version.

```
regular_grid_deltheta = 1.0
regular_grid_delphi = 1.0
```

If you encounter any error, look at the end of the tracer log files (`cookbook7.tracer_log.*`) for error messages. Note that this code does not work if `nprocx` or `nprocy` is greater than 1 in the full spherical version. The regional spherical version does not have this constraint.

6.8.2.1 Example: Thermo-Chemical Convection, `cookbook7.cfg`

```
[CitcomS]
solver = full
steps = 31                  ; number of time steps


[CitcomS.controller]
monitoringFrequency = 10    ; how often outputs are created


[CitcomS.solver]
datadir = output            ; path to output directory
datafile = cookbook7        ; prefix of output filenames
rayleigh = 1e7


[CitcomS.solver.ic]
num_perturbations = 1
perturbl = 3
perturbm = 2
perturblayer = 5
perturbmag = 0.05


[CitcomS.solver.output]
output_optional = tracer,comp_nd


[CitcomS.solver.tracer]
```

```
tracer = 1
tracer_ic_method = 0
tracers_per_element = 20
tracer_file = tracer.dat

tracer_flavors = 2
ic_method_for_flavors = 0
z_interface = 0.7

chemical_buoyancy = 1
buoy_type = 1
buoyancy_ratio = 0.4
reset_initial_composition = 0

regular_grid_deltheta = 1.0
regular_grid_delphi = 1.0


# Assign the viscosities.
[CitcomS.solver.visc]
VISC_UPDATE = on
num_mat = 4
visc0 = 1,1,1,1
TDEPV = on
viscE = 0.2,0.2,0.2,0.2
viscT = 0,0,0,0
VMIN = on
visc_min = 1.0
VMAX = on
```

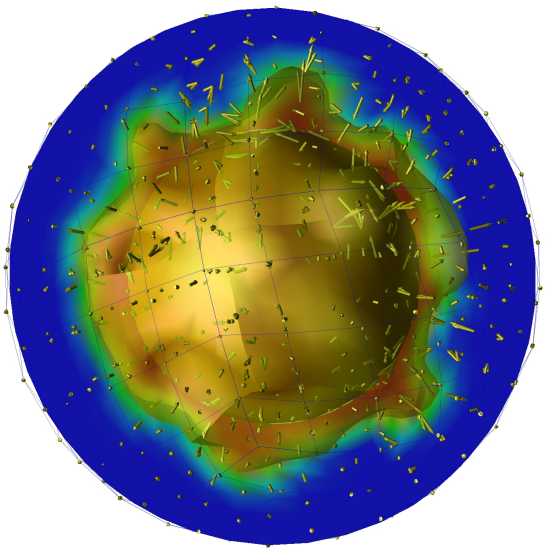


Figure 6.9: Cookbook 7: The composition and velocity field at the 20th step. The arrows are the velocity vectors. The composition field is shown in an isosurface of 0.7 and in a cross section.

6.8.3 Discussion

The results for this problem are presented in Figure 6.9. The buoyancy ratio in this model is too low to stabilize the chemical layer. A few thermo-chemical plumes are rising from the lower mantle, especially the ones at the 4, 11, and 12 o'clock directions. The resolution of this model is fairly low. The composition isosurface is slightly discontinuous across the cap boundary. A model of higher resolution will not have this kind of artifact.

Part III

Appendices

Appendix A

Input Parameters for CitComS.py

A.1 Input Parameters Grouped by Functionality

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

A.1.1 Parameters that Control Input Files

`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.
`coor=off` `coor_file="coor.dat"`	If `coor` is set to `off`, then there will be uniform mesh in the latitudinal, longitudinal, and radial directions. If you wish to have a regular, but uneven, spacing between elements, `coor` should be set to `on`. Then the coordinate is reading from the file specified by `coor_file`.
`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.

`tracer=off` `tracer_file=tracer.dat`	This controls the tracer particles which are advected passively by the flow. This part of the code has not been made parallel and must be used with caution with the present implementation.

A.1.2 Parameters that Control Output Files

`output_format=ascii`	Choose the format and layout of the output files. Can be either `ascii` or `hdf5`.
`output_optional="surf,botm,tracer,comp_el"`	Choose additional output, including `surf`, `botm`, `geoid`, `stress`, `pressure`, `connectivity`, `horiz_avg`, `tracer`, `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=10` (in Pyre version)	Controls the interval between output files. CitComS.py 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.
`output_ll_max=20`	This parameter controls the maximum degree of spherical harmonics coefficients for geoid output.

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.
	
	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 following constraints must be satisfied:
	
	$\text{nodex} = 1 + \text{nprocx} \times \text{mgunitx} \times 2^{levels-1}$
	$\text{nodey} = 1 + \text{nprocy} \times \text{mgunity} \times 2^{levels-1}$
	$\text{nodez} = 1 + \text{nprocz} \times \text{mgunitz} \times 2^{levels-1}$
	
	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.py.

`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, given below, 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.

A.1.5 Restarting the Code

`restart=off`	If `restart` is `on`, each processor will read its initial temperature field from a velo file.
`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 temperature field form file `regtest.velo.5.0` in this case.

`zero_elapsed_time=on`	If `zero_elapsed_time` is `on`, the initial time is set to zero. If it is `off` and `restart` is `on`, the initial time is read in from previous output files.

A.1.6 Run Length

`minstep=1`
`maxtotstep=1000000`
(only in non-Pyre 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 non-Pyre version.

A.1.7 Initial Conditions

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

`num_perturbations=1`
`perturbmag=0.05`
`perturbl=1`
`perturbm=1`
`perturblayer=5`
	If `tic_method=0`. The initial temperature is a linear temperature gradient with some perturbations, and `num_perturbations` specifies the number of perturbations. The amplitude of the perturbations is specified in the list of real numbers by `perturbmag`. In a full spherical model, `perturbl` and `perturbm` specify the shape of the perturbations in spherical harmonic degree and order. In a regional model, `perturbl` and `preturbm` specify the number of nodal lines in longitudinal and latitudinal directions. `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.
	
	If `tic_method=3`. The initial temperature is a linear temperature gradient with some perturbations. `num_perturbations` is fixed to 1 and `perturblayer` is ignored in this case. The perturbation is given by:
	
	$$mag \times \sin\left(\frac{(r-r_{in})\pi}{r_{out}-r_{in}}\right)\left(\sin(m\phi)+\cos(m\phi)\right)P_{lm}(\cos\theta)$$

`half_space_age=40`
`mantle_temp=1.0`
`blob_center`
`blob_radius=0.063`
`blob_dT=0.18`
	Used only if `tic_method=2` and in the regional model. This creates a top thermal boundary with a half-space cooling age specified by `half_space_age` in millions of years and a warm spherical blob. These parameters specify the temperature of the ambient mantle, the location and radius of the blob, and also the amplitude of temperature change in the blob relative to the ambient mantle temperautre. The location of the blob is default to the center of the computational domain.

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.
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 heatflux (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.
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 $\eta = \eta_0 \times \exp(\eta_E(\eta_T - T))$ When `rheol`=2, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(-\frac{T}{\eta_T})$ When `rheol`=3, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(\frac{\eta_E}{T+\eta_T} - \frac{\eta_E}{1+\eta_T})$ When `rheol`=4, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(\frac{\eta_E+\eta_Z(1-z)}{T+\eta_T} - \frac{\eta_E+\eta_Z(1-z)}{1+\eta_T})$

`viscE=1,1,1,1`
`viscT=1,1,1,1`
`viscZ=1,1,1,1`
	Parameters defining viscosity law (η_E, η_T, and η_Z 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=0.02`	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.
`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`.
`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`
	Multigrid parameters. `mg_cycle`=1 for V cycle and 2 for W cycle. `down_heavy` and `up_heavy` are the smoothing factors for downward/upward smoothing. `vlowstep` and `vhighstep` are the number of smoothing passes at lowest/highest levels.

`piterations=1000`	Maximum iterations of the outer loop for the momentum solver.
`accuracy=1.0e-6` `tole_compressibility=1.0e-7`	Convergence criterion for the momentum solver.
`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=on`	Filter out the overshoots and undershoots of the temperature field or not.
`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 `restart` or `post_p` is `on`, 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 `restart=on`.

`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. Tracers above `z_interface` are of flavor 0, otherwise, of flavor 1.
`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 locating 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 [2]. (No other methods are implemented yet.)
`buoyancy_ratio=1.0`	The ratio of chemical density anomaly to the reference density.
`reset_initial_composition=off`	Whether to reset the initial bulk composition or use the restarted value. Used when `tracer_ic_method=2`.

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`
`surftemp=273`
	Various dimensional information in SI units.

A.1.20 Required Information

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

A.2 CitComS.py Facilities and Properties

This section lists the facilities and properties in the Pyre version of CitComS.py. Most of the properties have names which are identical to the parameters used in the non-Pyre version of CitComS.py 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, including the 0th time step.
`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.py, an MPI application, is executed on multiple processors. It is the equivalent to `mpirun` in MPI.

`dry=off`	If on, print the `mpirun` command line and exit 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]`

A.2.3 `scheduler`

The scheduler facility controls how CitComS.py submits jobs to a batch scheduler.

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

A.2.4 `job`

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

`name`	The name for 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=10`	The time step interval between disk output. It replaces the `storage_spacing` parameter in the (old) CitComS input file.

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 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.
visc	A facility for viscosity parameters.
datadir="."	
datafile="regtest"	
datadir_old="."	
datafile_old="regtest"	
rayleigh=100000	
Q0=0	
stokes_flow_only=off	
verbose=off	
see_convergence=on	

A.2.7 solver.mesher

nproc_surf=1 or 12	This parameter is set by the solver facility. Users do not need to change it.
nprocx=1	
nprocy=1	
nprocz=1	
coor=off	
coor_file="coor.dat"	
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=on	
finetunedt=0.9	
fixed_timestep=0.0	
inputdiffusivity=1	
adv_sub_iterations=2	

A.2.9 solver.vsolver

Solver=cgrad	
node_assemble=on	
precond=on	
accuracy=1e-06	
tole_compressibility=1e-07	
mg_cycle=1	
down_heavy=3	
up_heavy=3	
vlowstep=1000	
vhighstep=3	
piterations=1000	
aug_lagr=on	
aug_number=2000	

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	
surftemp=273.0	
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	
num_perturbations=1	
perturbl=1	
perturbm=1	
perturblayer=5	
perturbmag=0.05	

A.2.13 solver.output

output_format="ascii-local"	
output_optional="surf,botm"	
output_ll_max=20	
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

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	
mantle_temp=1.0	
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	
regular_grid_deltheta=1.0	
regular_grid_delphi=1.0	
chemical_buoyancy=on	
buoy_type=1	
buoyancy_ratio=1.0	
reset_initial_composition=off	

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,	
SDEPV=off	
sdepv_misfit=0.02	
sdepv_expt=1,1,1,1,	
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 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.

The journal facility is not used in the uncoupled CitComS solver. 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.CitComSExchanger`. Try running the scripts with the following options:

```
--journal.debug.Exchanger=on --journal.debug.CitComSExchanger=on
```

Appendix B

CitComS.py Input File Format

B.1 Introduction

CitComS.py 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 `bold` 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.py, 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 full spherical version of CitComS.py, 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, just 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.py, 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]
```

Appendix C

CitComS.py Output File Format

C.1 Introduction

The format of the output files of CitComS.py 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

The command `autocombine.py` produces 1 cap file for regional CitComS.py and 12 cap files for full CitComS.py, e.g., `test-case.cap00.10`. 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 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.4 ASCII Output

C.4.1 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.4.2 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.4.3 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.4.4 Material Output (`test-case.mat.0`)

In this version of CitComS.py, 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.4.5 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 heatflux vel_colat vel_lon
```

C.4.6 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:

```
Sxx Syy Szz Sxy Sxz Syz
```

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

$$\begin{bmatrix} Sxx & Sxy & Sxz \\ Syx & Syy & Syz \\ Szx & Szy & Szz \end{bmatrix} \tag{C.1}$$

C.4.7 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.4.8 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.4.9 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	sine coefficients of total geoid (S_{lm})
col 4	cosine coefficients of total geoid (C_{lm})
col 5	sine coefficients of the geoid due to internal density variation
col 6	cosine coefficents of the geoid due to internal density variation
col 7	sine coefficients of the geoid due to surface topography
col 8	cosine coefficients of the geoid due to surface topography

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.4.10 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.4.11 Composition Output (`test-case.comp_el.0.10` and `test-case.comp_nd.0.1`)

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 line of the file is a header. The fourth field in the header is the bulk sum of the current composition. The fifth field in the header is the bulk sum of the initial composition. The rest of the file is column-based, where the meaning of the only columns is:

```
composition
```

C.5 HDF5 Output (`test-case.h5`)

The format and layout of HDF5 output is described in Section 4.4 on page 41.

Appendix D

License

GNU GENERAL PUBLIC LICENSE Version 2, June 1991. Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program" below refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification.") Each licensee is addressed as "you."

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version," you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

[1] Moresi, L., M. Gurnis, and S. Zhong (2000), Plate tectonics and convection in the Earth's mantle: Toward a numerical simulation, *Comp. Sci. Engin., 2*, 22-33.

[2] Moresi, L.N., and V.S. Solomatov (1995), Numerical investigation of 2D convection with extremely large viscosity variations, *Phys. Fluid, 7*, 2,154-2,162.

[3] Moresi, L.N., and M. Gurnis (1996), Constraints on the lateral strength of slabs from three-dimensional dynamic flow models, *Earth Planet. Sci. Lett., 138*, 15-28.

[4] Zhong, S., M. Gurnis, and L. Moresi (1998), The role of faults, nonlinear rheology, and viscosity structure in generating plates from instantaneous mantle flow models, *J. Geophys. Res., 103*, 15,255-15,268.

[5] Zhong, S., M.T. Zuber, L.N. Moresi, and M. Gurnis (2000), The role of temperature-dependent viscosity and surface plates in spherical shell models of mantle convection, *J. Geophys. Res., 105*, 11,063-11,082.

[6] Tan, E., M. Gurnis, and L. Han (2002), Slabs in the lower mantle and their modulation of plume formation, *Geochem. Geophys. Geosys., 3*, 1067.

[7] Conrad, C.P., and M. Gurnis (2003), Seismic tomography, surface uplift, and the breakup of Gondwanaland: Integrating mantle convection backwards in time, *Geochem. Geophys. Geosys., 4(3)*, 1031, doi:10.1029/2001GC000299.

[8] Hughes, T.J.R. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc.; 1987. 672 p.

[9] Ramage, A., and A.J. Wathen (1994), Iterative solution techniques for the Stokes and Navier-Stokes equations, *Int. J. Numer. Methods. Fluids, 19*, 67-83.

[10] Brooks, A.N. *A Petrov-Galerkin Finite Element Formulation for Convecton Dominated Flows.* Unpublished doctoral thesis, California Institute of Technology, Pasadena, CA, 1981.

[11] Cahouet, J., and J.-P. Chabard (1988), Some fast 3D finite element solvers for the generalized Stokes problem*, Int. J. Numer. Methods. Fluids, 8*, 869-895.

[12] Atanga, J., and D. Silvester (1992), Iterative methods for stabilized mixed velocity-pressure finite elements, *Int. J. Numer. Methods. Fluids, 14*, 71-81.

[13] Hager, B.H., and R.J. O'Connell (1981), A simple global model of plate dynamics and mantle convection, *J. Geophys. Res., 86*, 4,843-4,867.

[14] Tan, E., E. Choi, P. Thoutireddy, M. Gurnis, and M. Aivazis (2006), GeoFramework: Coupling multiple models of mantle convection within a computational framework, *Geochem., Geophys., Geosyst. 7,* Q06001, doi:10.1029/2005GC001155.

[15] Zhong, S., M. Gurnis, and L. Moresi (1996), Free-surface formulation of mantle convection–I. Basic theory and appication to plumes*, Geophys. J. Int.*, *127*, 708-718.

[16] Zhong, S., A. Paulson, and J. Wahr (2003), Three-dimensional finite-element modeling of Earth's viscoelastic deformation: effects of lateral variations in lithospheric thickness, *Geophys. J. Int., 155*, 679-695.

[1] McNamara, A.K., and S. Zhong (2004), Thermochemical structures within a spherical mantle: Superplumes or Piles? *J. Geophys. Res.*, 109, B07402, doi:10.1029/2003JB002847.

[2] Tackley, P.J., and S.D. King (2003), Testing the tracer ratio method for modeling active compositional fields in mantle convection simulations, *Geochem. Geophys. Geosyst., 4,* 8302, doi:10.1029/2001GC000214.