Geodynamic benchmarking tests in HPC

Rebecca Farrington[1], Louis Moresi[1], Steve Quenette[2], Robert Turnbull[1], Patrick Sunter[2]

[1]Scool of Mathematical Sciences
Monash University
Clayton, VIC 3800, Australia

[2] Victorian Partnership for Advanced Computing,
Carlton South, Victoria 3053, Australia

June 10, 2005

Abstract

The increase in large science focused computational frameworks has raised many issues involving the ability to maintain accurate scientific benchmarks throughout the ongoing evolution of the code. These science based tests allow not only developers access to the latest updates, but the science users aswell. It is these scientific tests required for geodynamic code benchmarking in a HPC environment that are investigated. The importance of benchmarking in computational science, for both quality assurance and reliability, is discussed and a case study for thermochemical convection modelling is presented. The implementation of automated testing for science units is described with particular attention to the problems arising from science tests compared to traditional computational tests.

Keywords: Benchmarking, High Performance Computing, Geodynamics

1 Introduction

Our group is in the process of developmenting a general-purpose, parallel, solid-mechanics / fluid dynamics Lagrangian-Integration-Point finite element code primarily for use in geodynamics research. The project is multi-institutional and multi-disciplinary. It involves the simultaneous development of three codes: 1) a supporting framework code, 2) a generalized solver for handling governing mathematical equations, and 3) a code aimed towards modeling geological materials with complex constitutive behaviours. The codes are community-developed and have a tiered structure of dependency, so modifications made by developers at the low-level need to be seamlessly integrated by those working higher up at the science application level.

In a research environment the end-use-cases for the code are continually being refined as new models are run and new discoveries are made. We have adopted a very flexible approach to the code development with a short cycle of implementation, checking, release and redesign. The operation of the code is as modular as the inherent cross-coupling of the physical processes will allow. The code has to run within the environment of various parallel supercomputers which are far from homogeneous.

This leads to an interesting environment for testing the code. Due to this short cycle we must not only ensure the code is bug-free during this process of evolution, we must also continually test the science. These tests have to distinguish between total failure, acceptable numerical error and "important" changes in behaviour even if they are within error. The standard form of a numerical benchmark for scientific code may not specify test results for all these requirements.

Due to the number of developers, size of the code and continual evolution, these tests must also be performed on an ongoing basis. We are currently relying on manual benchmark tests for high-level geodynamic models. The logical step forward is for automatic regression testing. This will provide users and developers an up-to-date assessment on the real health of the code. However these tests can be difficult to automate, particularily distinguishing between "failure", "acceptable error" and "important behavioural changes".

Section 2 discusses the scientific context of the modeling code, the implementation details, and the development environment. Section 3 details the importance of benchmarking for numerical sciences and the issues surrounding the choice of benchmark problem. Section 4 discusses the individual benchmarks required for thermochemical convection. Section 5 deals with implementing automated benchmark tests into the framework.

2 Problem Description

2.1 Geodynamics

In computational geodynamcis we are concerned with the stresses and corresponding motions on the surface and within terrestrial planets. Due to the long time scales involved we can approximate these movements as those of a fluid. We are then able to draw upon the large wealth of knowledge in the field of fluid dynamics.

In order to model, for example, mantle dynamics we use the principles of conservation of mass, energy and momentum as well as assuming an incompressible, boussinesq fluid to model thermal convection. This leads to the governing equations

$$\begin{aligned} \nabla \cdot \mathbf{v} &= 0 \\ \frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T &= Q(T) + \kappa \nabla^2 T \\ \eta \nabla^2 \mathbf{v} &= \nabla p + g \rho \hat{z} \end{aligned} \tag{1}$$

It is usual to work with a non-dimensional system of equations to improve numerical accuracy since we solve for very small velocities (cm/yr) driven by very large stresses ($10^8 - 10^9 Pa$) acting on fluids with very large viscosity ($10^{19-23} Pas$). Once rescaled, the equations can then be solved efficiently using a particle-in-cell finite element method (7) (8).

2.2 Code Description

The code used for our geodynamic modeling is essentially a parallel, particle-in-cell finite element code with a modular structure designed to encourage rapid prototyping of new scientific modeling concepts. A plugin infrastructure allows for highly complex mechanical models to be implemented efficiently (and safely from the point of view of other users of the code). This is achieved by the addition of either numerical or science applications at various entry points which are called during the code's execution. Modules cannot be firewalled from each other or prevented from interacting, as the physical process which one module implements is often strongly physically coupled to processes implemented in other modules. Each plugin needs to work both in isolation and interacting as part of a larger system.

Underworld is the specialised geodynamics modeling part of our code. It utilises the plugin environment to build upon a science-neutral finite element modeling framework. It decribes the initial and boundary conditions, the material properties, and rheological laws specific to geodynamics, along with specialised processing of the computed solution to match geophysical observables.

The Particle-In-Cellerator (PICellerator) is a separate package that implements the Particle-In-Cell intergration schemes for the science neutral finite element framework. Underworld assumes this integration scheme in the implementation of its material-history-dependent rheological laws.

StgFEM is a framework for developing science netural finite element codes. It is broken into two sub-frameworks, one for creating finite element discretisations and one for creating systems of linear equations. It utilises the linear solver package PETSc.

StGermain is a fundamental framework for developing extensible scientific codes and frameworks. It provides the implementation for plugins and associated infrastructure. It is motivated by the need for better implicit software engineering in scientific codes.

Snark is the software that provides a general solver capability given a set of governing equations, but does so using the more low-level infrastructure made available by StGermain (6), StgFEM and/or PICellerator. Examples from this toolchain are basin extension modelling, mantle convection, slab subduction, plumes, etc.

gLucifer (10) is the visualisation framework used with and developed for StGermain. It provides interactive and background rendering for serial and parallel applications running on local and remote machines. That is, images and movies are created during run time, allowing for timely analysis of results. This visual output can be a key method for determining if the code output is valid.

2.3 Development Team

Our development team consists of a broad range of loosely interacting scientist and programers across a number of organisations. While this group is an organised entity, development is allowed, and encouraged, to proceed in an organic fashion. This is particularly prevalent for the development of the science applications. While each plugin is developed individually, it is also available to the wider community and therefore can interact with many other modules within the code, in situations not always anticipated by the original author. It is important to ensure each developer is aware of this possibility when designing new applications, ensuring they are not only free from software bugs, but scientifically accurate as a stand alone product and as a dynamic, interacting piece of the larger framework.

2.4 Testing responsibility

The tests for a given plugin should be the responsibility of the individual developer &/or team. Each plugin, before it can be integrated into a stable release of the code, has to be accompanied by a test-case and the expected results. To assist developers of other plugins to interact safely and predictably with a given plugin, it should be possible to run in a "do nothing" mode by setting particular combinations of parameters (that is to say, the plugin loads, and executes all of its code, but the physical processes have no effect with this particular combination of settings). Where possible, parameter combinations which reproduce the behaviour of other plugins should be given (for example: a visco-elastic deformation module should be able to reproduce viscous behaviour if the elastic modulus is infinitely large).

3 Benchmarking

3.1 Why benchmark codes ?

The classical way to verify physical modeling codes is to see how well they work when compared with simple analytic solutions, experimental observations and scaling results, and other codes or versions of the same code. This process is known as benchmarking.

Carefully chosen, relatively simple benchmark problems allow us to gain confidence that the results of more complex problems are reliable. Benchmark problems are an important component of the peer review assessment of scientific work based upon modeling. Benchmark problems are also used to develop rules of thumb to determine the minimum resolution required by a given modeling algorithm to resolve more realistic problems. In 3D problems, the computational effort required increases dramatically with any increase in resolution; it is important in HPC to be able to estimate the resources required for a specific model.

One of the challenges in developing benchmarking problems is to design tests which are simple enough to have well-understood solutions, and which can be "incremented" to include new physical effects in turn. A good example comes from convection: the linear diffusion equation can be solved analytically. Advection is very hard to test in isolation as, in the absence of diffusion, it becomes a special case for which specialised advection schemes are needed. Advection-diffusion as a coupled problem is easier to solve numerically and more relevant to the actual problem, but harder to benchmark in 2D/3D due to the inherent non-linearity.

3.2 Bug detection versus error detection

It is important as a modeler to “know your code” as it can only ever provide an approximate solution to the mathematical formulation of the problem. Benchmarking helps in this characterization. Benchmarking looks at whole sub-systems of the code and is only possible once at least one component of the system of equations can be solved. Benchmarks are designed to detect inaccuracies in the implementation of the algorithm, incompatibilities between plugins, (in)appropriate choices of resolution of the discretisation, and (in)appropriate match between the chosen algorithm and the nature of the mathematical model equations. Benchmarks are not designed specifically to flush out bugs in the framework parts of the code but are sensitive to their existence.

3.3 Good Benchmarks

The ‘classic’ benchmark is appropriate for relatively simple models, where analytical solutions exist, or with only a small number of interacting components. This process becomes increasingly difficult with the complexity of the models. The ideal benchmark problem should be fast, with deviations from expected results appearing within the behaviour of the system during the early stages of the test.

It is important in scientific code to test the isolated physical components which may not be used individually within the working code. For example the thermal diffusion and advective heat flux, which are both components outlined in equation 1, are not seen individually in the model. It is therefore common for a given problem to require a series of tests to be completed, initially testing the individual units then tests of the larger system.

Ideally, a benchmark problem should be sensitive to algorithm change. The benchmark problem might be chosen in parameter space near a bifurcation, where small changes in the model can push to another part of the phase space. For example through a period doubling, from a time-independent to time-dependent condition, or past a catastrophic transition. The overall model results may still be within acceptable accuracy limits as defined by a pass/fail test, however, this change in behaviour might be significant for particular applications and should always be documented.

It is also worth mentioning that changes in model results should always be highlighted as part of a testing / benchmarking regime even if they indicate a reduction in error relative to simple models: reproducibility of model results is paramount. For example, a method for tracking compositional interfaces(5) which has commonly been used in geodynamics relies on correcting certain known properties of the chosen advection scheme to reduce numerical diffusion. A “helpful improvement” of the advection scheme to a more accurate, higher order scheme would immediately break this algorithm.

4 Case study: Thermochemical Convection

Thermochemical convection problems frequently arise in geodynamics including, for example, the behaviour of chemically buoyant plates overlying the lithosphere and the entrainment of the dense, low lying D” layer at the base of the mantle. The mathematical description of thermochemical convection is identical to that of thermal convection with the addition of a compositional buoyancy force and a compositional conservation equation (pure advection of material domains).

The non-dimensional equations governing thermochemcial convection are therefore given by

$$\begin{aligned}
\nabla \cdot \mathbf{v} &= 0 \\
\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T &= Q(T) + \kappa \nabla^2 T \\
\frac{\partial C}{\partial t} + \mathbf{v} \cdot \nabla C &= Q(C) + \kappa \nabla^2 C \\
\eta \nabla^2 \mathbf{v} &= \nabla p + (RaT - RbC)\hat{z}
\end{aligned} \tag{2}$$

where

$$Ra = \frac{g\alpha\Delta\rho d^3}{\kappa\eta}, Rb = \frac{g\beta\Delta\rho d^3}{\kappa\eta} \tag{3}$$

For thermochemcial convection many individual plugins and modules within the code must be used, including thermal diffusion-advection schemes, particles advection and integration schemes, and momentum equation solvers. In order to achieve accurate results, all of these modules must be routinely tested throughout the evolution of the code and compared to well established benchmark problems.

Some of the difficulty surrounding these high-level problems lies in identifying the interdependency of these modules. Positive outcomes for individual module tests will not necessary yield accurate thermochemical benchmarking results. Problems also arise in automation of these high-level benchmarking tests. Determining the accuracy of a thermal advection diffusion schemes can be fairly straightforward. However the testing increases in complexity as the simulations include momentum equation solvers with thermal buoyancy or chemical buoyancy with particle advection and integration, leading to the complete thermochemical convection.

Automated testing of high-level simulations is not as straight forward as achieving a single number result within a specified accuracy. For these simulations there is no ‘correct’ numerical solution. It is therefore often more important to produce a trend than a single number for comparison. It is these tests which can not only be complicated to develop, but time consuming to perform during the ongoing development of the code.

4.1 Thermal advection-diffusion schemes

In order to solve for the energy equation used in infinite prandtl number, boussinesq fluid solutions we must implement an advection-diffusion solver. To test the accuracy of our implemented solver, results are benchmarked to the published numerical examples of Brooks and Hughes (1982) (2), who implemented a streamline upwind, petrov-galerkin formulation for convecting flows.

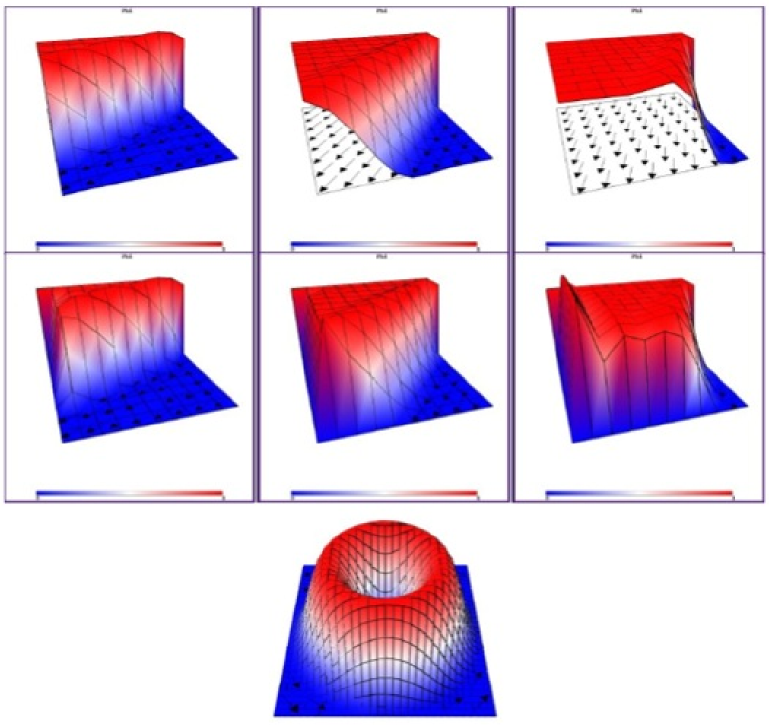

Figure 1: Results for the advection-diffusion scheme benchmark, skewed advection with homogeneous natural (top) and general (middle) outflow boundary conditions and the advection of a cosine hill as detailed in (2)

These simulations solve the linear advection-diffusion equation for a range of problems, as detailed in section 3.4.2 of (2). These include skewed advection for homogenous, natural and essential, boundary conditions and the advection of a cosine hill in a rotating flow field. Test results for these simulations are shown in figure 1. The results are compared for each nodal value, with the deviation to the published figures tested to ensure a specified accuracy.

4.2 Momentum solver

The momentum solver is used for both the continuity and momentum equation, to solve for stokes flow. To test the accuracy of the scheme, the common stokes flow problem of driven cavity flow is used, as outlined in Hughes (1987) (3). This simulation solves stokes flow within a box with free-slip boundary conditions, $V_x = 0$, along the bottom and sides, with $V_x = 1$ along the upper boundary, as shown in figure 2.

The resulting Nusselt number and Vrms trends for the implemented scheme can then be compared to those of previously calculated results.

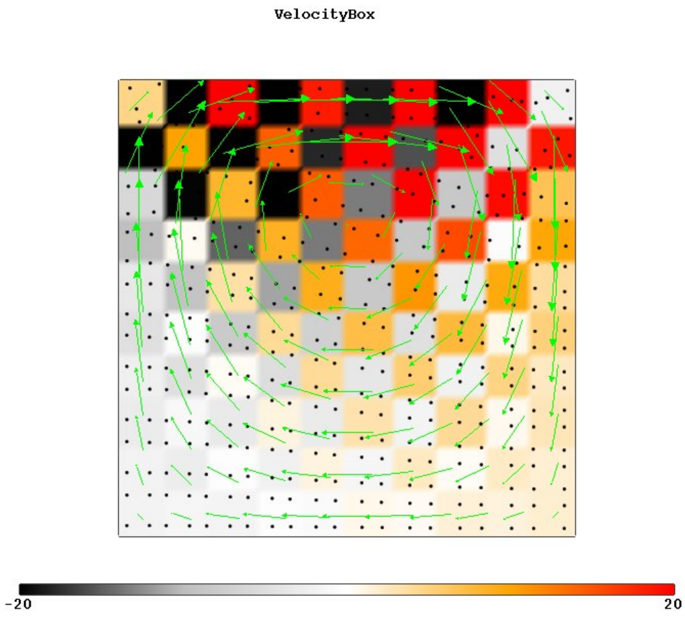

Figure 2: Benchmark problem results for the momentum solver. A lid driven convection problem with particles as outlined in (3)

4.3 Thermal convection benchmarks

Thermal convection, with instabilities driven by thermal density gradients, requires both energy and momentum solvers, and thermal advection diffusion routines. The momentum equation must also be solved for a thermal buoyancy forcing term. The benchmark standard for thermal isoviscous mantle convection codes is outlined in case 1 of Blankenbach et al (1989) (1).

Simulations are run for a range of Rayleigh numbers in a box of aspect ratio 1. The resulting steady state Nusselt number, shown in figure 3, and Vrms values are used to determine accuracy.

While this is a long standing benchmark, both the Nusselt number and Vrms values are global measures, averaged across the entire simulation. This measure therefore smoothes out local incon-

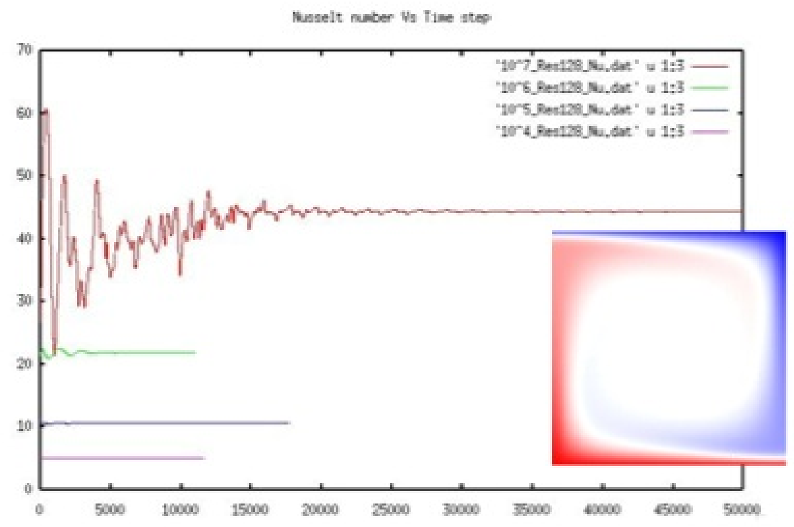

Figure 3: Benchmarking results for thermal convection, a convection cell in a box of dimension 1×1 for Rayleigh numbers of 10^4 to 10^7 as detailed in (1). The resulting Nusselt number trends are shown.

sistencies which may arise within the system. The time to steady state also increases with Rayleigh number, a measure of the time dependence within the system, as shown for Rayleigh of 10^7 in figure 3.

4.4 Particle advection schemes

In order to model materials with different properties we include particles in the simulation. These particles are assigned material properties, including density and viscosity. The particles are moved through the velocity field using advection schemes.

The accuracy of the particle advection scheme can be assessed using van Keken et al (1997) (4), by testing the tracing of passive markers (Appendix C of (4)). The accuracy is tested by the deviation in the particles position after one complete cycle of a steady state flow, as shown in figure 4.

While results for this test are easily obtained and tested accurately, it is tested within a constant velocity field and therefore does not test the scheme for accuracy with time. To access the true accuracy of the advection scheme a benchmark which tests for the accuracy of time to the 2nd order is required.

The test available for the individual solution schemes are relatively straight forward. The momentum solvers, thermal advection diffusion and particle advection schemes can be tested accurately to ensure correct simulation results. However complexities arise when these schemes are brought together with particle integration methods to model geodynamic problems like that of Rayleigh-Taylor

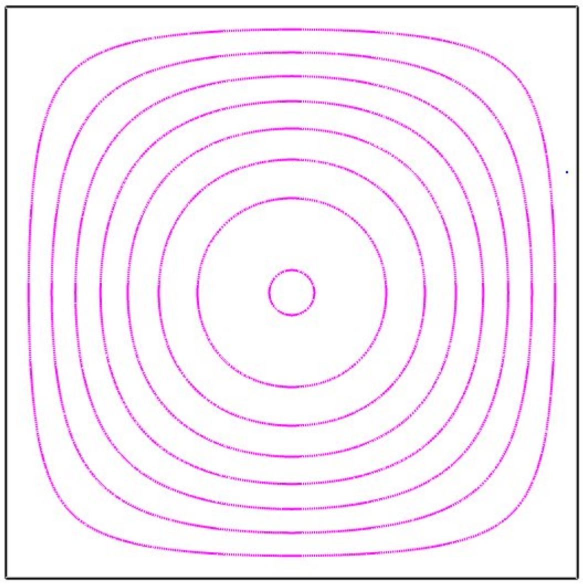

Figure 4: Benchmarking result for the particle advection scheme. The advection of tracer particles through a constant velocity field as detailed in (4)

instabilities.

4.5 Rayleigh-Taylor convection and particle integration schemes

The instabilities within Rayleigh-Taylor convection are caused by compositional density variations, with the forcing term applied to the momentum equation being compositional buoyancy. The particle advection and integration schemes therefore need to be tested for accurate modelling.

There are many different integration schemes available to apply the weightings to specific particles with respect to the local coordinates and material properties. The simulations shown in figure 4 are produced using approximately 20 particles per element which are initially spread uniformly throughout the domain.

The benchmark used to test the Rayleigh-Taylor simulations is outlined in van Keken et al (1997)(4), with our results shown in figure 5. The variations between these solutions, a result of differing resolution, grow with time. Hence, after a relatively short period the solutions diverge, with distinct instabilities within the boundary layers of figure 5(a) growing at a faster rate than those of figure 5(b).

How do we measure, in an automated way, the appearance and individual behaviour of rising plumes during the course of a numerical model experiment? The quantifiable benchmark value for the Rayleigh-Taylor instabilities is the maximum Vrms value, coinciding with the arrival of the first plume at the mid-point of the upper boundary. The difference between this value for the 64×64 and

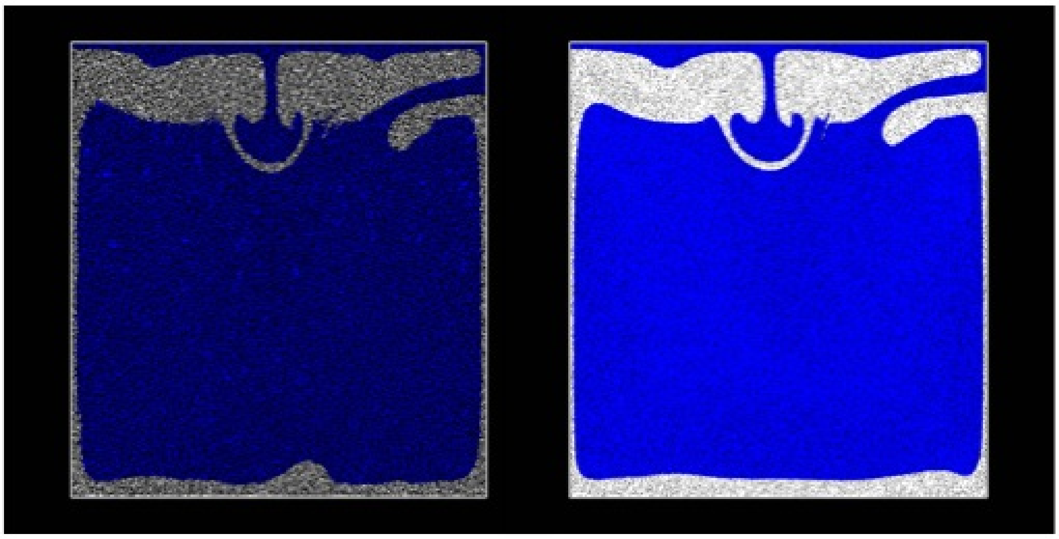

Figure 5: Benchmarking results for Rayleigh-Taylor convection. The motion of an initially low, lying buoyant layer as detailed in (4), for a resolution of (a) 64×64 (left), (b) 128×128 (right).

the 128×128 simulation is only 2.5%. The simulations do however display dynamic differences, with instability growth and plume formation occurring in the 64×64 simulation after the maximum Vrms value is reached, which do not occur in the 128×128 simulation.

4.6 Thermochemical benchmarks

Modelling thermochemical convection requires an application incorporating all of the previously mentioned equations and solution methods. This introduces interdepencies at all levels, thereby increasing the complexity of the system remarkably.

The benchmark used for these simulations is again outlined in van Keken et al (1997) (4). Our simluation results shown in figure 6 match those published in figure 8 for $t = 0.01$ and figure 12a of (4). The entrainment rate of the dense fluid into the overlying fluid is also used as a measurable in this benchmark, however the published results begin to diverge very early, at the point where entrainment begins.

Unlike thermal convection, that converges towards the single solution, the solutions diverge once particles are involved. We therefore have the complexity of models increasing with possible benchmark measurements decreasing. Quantifying individual routines for accuracy can be straight-forward, however measuring their interactions is not.

How do we then attempt to automate these benchmarkings into an ever evolving framework? En-

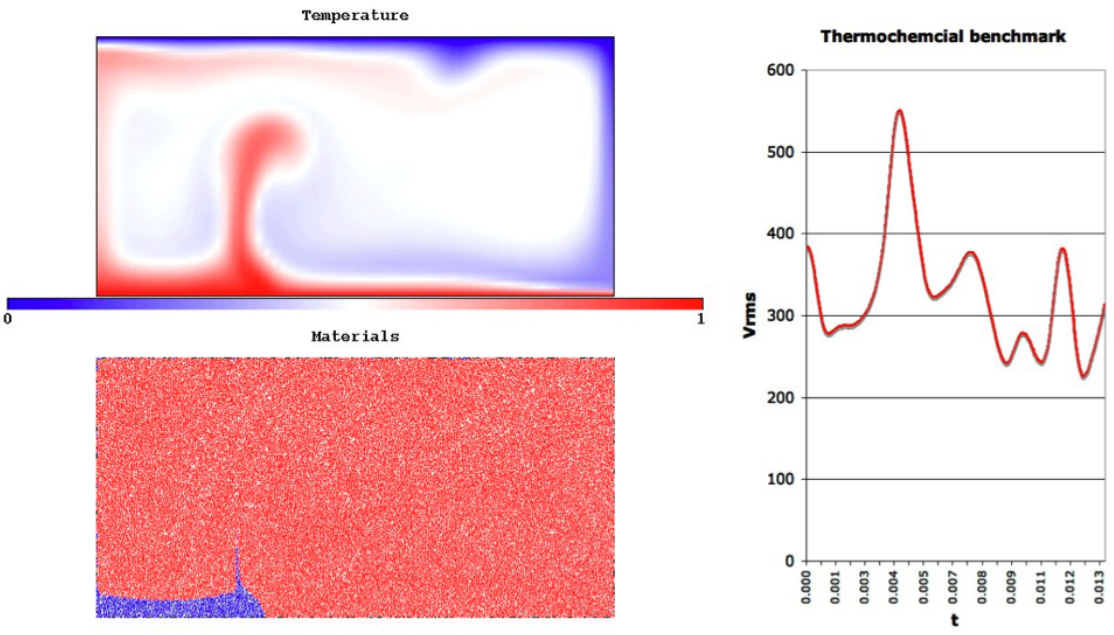

Figure 6: Benchmarking results for thermochemical convection. The entrainment of a low, lying dense layer by thermal convection, as detailed in (4), with the (a) temperature field (top left) (b) particle field (bottom left) and (c) Vrms trend (right).

suring many different groups working on a range of geodynamic problems have access to the most up-to-date and accurate code.

5 Automated Testing of Benchmark Solutions

The aim of automated science tests is to achieve benchmarking of both interacting and isolated science components as unit tests. This is a shift from the current perception of benchmarking as an operation on a complete code to a action on a specific unit within the code. It requires that test creation is seemingly effortless to the science implementer, a computational scientist not a programmer. It also requires the ability to break the science into appropriately sized units, as demonstrated above.

To achieve a solution satisfactory to a both scientists and programmers, benchmarking is implemented into three phases: the modular/unit-based code, the ability to have unit tests and finally the development of science testing tools.

5.1 Modular/Unit-based coding

Object oriented programming is typically shunned in scientific codes, especially in HPC environments where there is a perceived loss of performance associated with the encapsulation of data and methods. However, there also tends to be confusion between the Object Oriented approach and predominant languages that support it. Course-grain Object Orientation does not severely affect performance with respect to the ability to maintain code(9), and languages such as C can have interfaces created to enable the methodology.

StGermain enables the detangling of ‘spaghetti’ code which was once typical in scientific applications into course-grained ideas. It provides three mechanisms for describing objects: Classes, Components and Plugins. Classes suit the fine grain objects that have a role within the infrastructure, but typically not suited to scientific concepts. Components are used to represent objects that can be configured to interact with others at run time. They are typically of course-grain concepts, such as a mesh or a linear algebra solver, etc, which in turn interact to form the system that is the code. Plugins are used to introduce new Component types to the system, or to change the configuration or behaviour of a system.

The plugins are typical of scientific units. For example, a plugin used to modify rheology adds to the material properties and modifies the rheological law, but is not a class per se. Classes, Components and Plugins are then considered units and are subjected to unit tests.

In the thermochemical case study we outlined the key science units needed for thermochemical convection. The advection diffusion schemes and momentum solvers are special cases of a system of linear equations abstract component of StgFEM. The algorithms that implement these pieces of science have their own established benchmarks to prove the algorithm works, which in turn is used as the unit test to that component. The thermal convection benchmark is applicable to the system when both advection diffusion schemes and momentum solver components are in use. In this case the unit test is more like a system test or benchmark.

On the other hand, in the Rayleigh-Taylor instability model, the given particle integration and advection schemes are implemented via plugins. They customise the system into a particular configuration. With each configuration having their own unit tests for specific behaviour. The thermochemcial benchmark is a system test of the given configuration.

5.2 Enabling unit tests

The codes' build system needs to be developed to support testing. There is a clear benefit in automatically building the tests with the code, and disabling any ability to switch this feature off. For the developer, this links ascertaining confidence in the supporting code with the development process. The computational scientist's work flow evolves from "changing the code, building the code and running the code" to "changing the code, developing tests, building the code and tests, optionally running the tests, and then running the code". Building the tests ensure that interface changes are trapped early, which in turn reduces the changes of code forks, replication of implementations and their tests. Where the development is done by a community of developers, then automation of the regression test becomes important.

Some infrastructure is required to build simple unit tests. StGermain provides an elementary set of features: selective printing, fire-walling and difference checking. Output streams are toggled on or off per unit at the user's discretion. A test may hardwire a unit's output to be on, where it would be otherwise disabled during production runs. Firewalls are effectively a print on assertion combination, thus simplifying the interface to a "check" of state within a unit. The build system itself is capable of comparing whether the output is identical to an expected outcome, enabling rapid testing without requiring any additional code.

Some of the initial science unit tests are already included in the testing suite, including the advection-diffusion schemes and momentum equation solvers. These regression tests are currently conducted automatically on a daily basis. We are left with the science tests, the benchmarks for Thermal, Rayleigh-Taylor, Thermochemical convection and the other geodynamic science tests as required by the user base.

5.3 Development of science level unit tests

At some point, the actual items within a unit being tested begin to have scientific meaning. For example, in thermochemical convection Nusselt number trends, temperature and particle fields may be subjected to comparison with an expected result. There are many reasons why these results may numerically differ but the scientific judgement considers them equivalent. One reason may be numerical accuracy of the hardware, another may be the use of another linear algebra solver.

Typically a scientist will subjectively assess the results via a visualisation in a suitable representation. In an automated system, this is not practical, and some mathematical operation is needed to reduce the judgement to one number within a tolerance. In simple cases convergence tests are suitable and implementation of convergence tests methods typically utilised by linear algebra solvers is possible. In the case of a high-level science test, these automated recursive testing becomes more complicated.

Testing the code manually using benchmarking problems is very accurate. We can easily ascertain the accuracy of many of the more complicated tests both visually and quantitatively through the Nusselt number or Vrms values. It is a combination of the quantitative results and the visual appearance of the simulation at specific times which provides the benchmark. The visual check gives us confidence in these global measurements. These visual checks are more difficult to automate than the calculated measures. This is particularly true when particles are involved. How do we measure the motion of individual plumes? To have a completely automated benchmarking test suite, we would need to implement some type of 'face' recognition software. A fix for this issue is to maintain the manual visual check through automated outputs which can be verified as required.

Simulations for thermal convection at high Rayleigh number or to resolve thermochemical convection require long periods of time to accurately produce results. Determining the resolution required for solutions to be resolved is also important. It is therefore preferable to run each science test for a range of resolutions and thus detail the required resolution as this may also change with the evolution of the code. These time increases are an important factor to consider when automating these problems as they will be produced on a daily basis, and the cumulative time for these tests quickly becomes significant.

It is also important to note that science tests should be run on a range of platforms. The resulting accuracy calculated, resolution required and visual checks also need to be published throughout the developer and user communities. Thus providing all parties with up-to-date information on the accuracy, resolution and versions required for the models. For example, if the accuracy of the particle advection scheme becomes a problem with the current version, users simulating thermochemical convection may need to revert to an older version. However, those requiring only thermal convection can remain on 'the bleeding edge'.

6 Conclusion

The automation of science based testing within our framework will not only ensure that all members of the development and user community have access to the most accurate code at all times, it will also raise the importance of the science tests to the level of traditional system tests. It is no longer acceptable to simply have a working code, it must also be able to solve many, varied geodynamical problems to a world class accuracy. While this has been possible for a long while, the short cycle of implementation and large base of end users has made it difficult to ensure all science aspects of the code are operational on demand. Implementing automated science testing will ensure all participants, both developers and users, are aware of the status and version required for the end product which is, after all, the science.

7 Acknowledgements

We would like to acknowledge the support of the Australian Computational Earth Systems Simulator (ACcESS), the Australian Partnership for Advanced Computing (APAC), the Victorian Partnership for Advance Computing (VPAC) and Monash Cluster Computing (MC^2).

Code documentation can be found at

- Underworld at http://www.mcc.monash.edu.au/twiki/view/Codes/UnderWorld
- PICellerator at https://csd.vpac.org/twiki/bin/view/PICellerator/WebHome
- StgFEM at https://csd.vpac.org/twiki/bin/view/CSD/StgFEM
- StGermain at https://csd.vpac.org/twiki/bin/view/Stgermain
- Snark at https://csd.vpac.org/twiki/bin/view//Snark.WebHome
- gLucifer http://www.mcc.monash.edu.au/twiki/view/Codes/Lucifer

References

[1] B. Blankenbach, F. Busse, U. Christensen, L. Crespes, D. Gunkel, U. Hansen, H. Harder, G. Jarvis, K. Koch, G. Marquart, P. Moore, D. andOlson, H. Schmeling, and T. Schnaubelt, *A benchmark comparison for mantle convection codes* Geophysical Journal International, 98:23-38, 1989.

[2] A. Brooks & T. Hughes, *Streamline upwind Petrov-Galerkin formulations for convection dominated flows with paricular emphasis on the incompressible Navier-Stokes equation*, Comput. Methods Apll. Mech. Eng. **25**, (1982) pp 199-259.

[3] T. J. R. Hughes. *The Finite Element Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1987.

[4] van Keken, P. E. and King, S. D. and Schmeling, U. R. and Christensen, U. R. and Neumeister, D. and Doin, M.-P., *A comparison of methods for the modeling of thermochemical convection*, Journal of Geophysical Research **102**, October 1997, B10, pp 22477-22496

[5] A. Lenardic and W.M. Kaula. *A Numerical Treatment of Geodynamic Viscous Flow Problems Involving the Advection of Material Interfaces* Journal of Geophysical Research, **98(B5)** pp 8243-8260, 1993.

[6] L. Moresi and D. May and J. Freeman and B. Appelbe *Mantle convection modeling with viscoelastic/brittle lithosphere: Numerical and computational methodology* Computational Science - Iccs 2003, Pt Iii, Proceedings **2659** pp 781-787, 2003.

[7] L. Moresi, F. Dufour, and H. B. Muhlhaus. *Mantle convection modeling with viscoelastic/brittle lithosphere: Numerical methodology and plate tectonic modeling* Pure And Applied Geophysics, **159(10)** pp 2335-2356, August 2002.

[8] L. Moresi, F. Dufour, and H. B. Muhlhaus. *A Lagrangian integration point finite element method for large deformation modeling of viscoelastic geomaterials* Journal Of Computational Physics, **184** pp 476-497, 2003.

[9] S. M. Quenette, B. F. Appelbe, M. Gurnis, L. J. Hodkinson, L. Moresi, and P. D. Sunter *An investigation into design for code maintainability in HPC* , ANZIAM Journal (in review).

[10] G. Watson and D.R. Stegman and C. Duboz and L. Moresi *gLucifer: Next-generation visualization framework* AGU 2004 Fall Meeting Abstracts, 2004.