The Devil truly is in the detail. A cautionary note on computational determinism: Implications for structural geology numerical codes and interpretation of their results

Stuart Hardy[1]

Abstract

The widespread use of numerical modeling codes in structural geology, particularly in forward modeling of crustal deformation, has brought valuable insight into many processes such as fault initiation and propagation, fold growth, and orogenic wedge development. Whether commercial, open source, or the result of an individual researcher's endeavor, such numerical codes increasingly rely on advanced compiler optimizations and/or parallelization techniques. This is done to maximize performance on today's multicore processors and thus achieve tractable model runtimes. However, such speed and productivity can come at a price. Many optimization and parallelization techniques used do indeed produce remarkable runtime speedups, but at the expense of determinism and reproducibility. Most researchers are only tangentially aware of these issues. I have aimed to explain why numerical model runs with identical sets of initial and boundary conditions can potentially produce different results and thus, sensu stricto, be nondeterministic. I have developed some solutions and guidelines for maintaining determinism and evaluated these issues using a discrete element code of upper crustal deformation. Implications for the interpretation and use of such model results in structural geology were also evaluated.

Introduction

Recently, high-performance computing (HPC) has effectively become accessible to the average researcher due to the rapid advances in desktop, multiprocessor/core, computing power (see e.g., McMillan, 2011). This has allowed a variety of problems to be tackled in runtimes that are, for the first time, measured in hours and days rather than weeks or months. Such HPC has been applied to several topics in structural geology, particularly forward modeling of deformation (e.g., Li et al., 2009; Hardy, 2013). The ability to reproduce such results goes a long way toward enhancing the fidelity of such simulations, and hence confidence in the scientific discoveries themselves. Such *determinism* lies at the philosophical base of much scientific thought and experimentation. A deterministic model is one in which every set of variable states is uniquely determined by parameters in the model and by sets of previous states of these variables. Therefore, deterministic models perform the same way for a given set of initial and boundary conditions and are thus *reproducible*. The wider scientific world is now very well aware of complexity and chaos in large, multicomponent, systems, whereby subtle differences in input parameters produce very different results (see Kellert, 1993). However, if no input parameters or boundary conditions are altered, the basic expectation is still that a given numerical model, run repeatedly, will produce identical results. This applies whether the system under consideration is a global climate model, a physics-based multibody simulation/game, or a mechanical model of deformation in the brittle upper crust. Unfortunately, this is not necessarily the case in many numerical codes. This short note aims to discuss this issue, illustrate it using several examples from a discrete element numerical model of upper crustal deformation, and propose some generic guidelines and possible solutions aimed at achieving determinism and reproducibility. The implications for the interpretation and use of such model results in structural geology are also discussed.

Before continuing, the differences among accuracy, precision, reproducibility, and performance will be emphasized. Accuracy of numerical codes refers to the calculation of results that are "close" to the result of an exact, analytical calculation (if that exists). This is often measured in fractional error or sometimes units in the last place. Clearly, the objective is the most accurate numerical calculation possible given appropriate constraints. Precision of a numerical quantity is a measure of the detail in which the quantity is expressed or stored. The number of bits used to store a number will often cause some loss of accuracy. The error is then often magnified because subsequent computations are made to the data. In con-

[1]Institució Catalana de Recerca i Estudis Avançats, Barcelona, Spain and Universitat de Barcelona, Facultat de Geologia, Barcelona, Spain. E-mail: stuart.hardy@icrea.cat.

Manuscript received by the Editor 3 March 2015; revised manuscript received 13 May 2015; published online 24 July 2015. This paper appears in *Interpretation*, Vol. 3, No. 4 (November 2015); p. SAA29–SAA35, 3 FIGS., 1 TABLE.

trast, reproducibility concerns itself with the production of consistent results either on the same machine, on different machines, or on different operating systems. Thus, accuracy, per se, is not the issue, the objective is reproducible results for a given chosen, or imposed, computational accuracy. It is the desire for a numerical model that runs as fast as possible — performance — that has driven and availed itself of advanced compiler optimizations and parallelization techniques. What the average scientist aims to do is to run a given simulation more quickly than before or to investigate a problem that was previously not tractable. Unfortunately, accuracy, precision, reproducibility, and performance are often difficult to reconcile, if they are not in direct conflict.

Prior to the discussion of the main topic of this short paper, it is necessary to clarify/define some of the terminology typically encountered when discussing HPC numerical modeling as follows:

1) Model — The logical/mathematical description of a process, often also used to describe the computer program itself.
2) Code — The manner in which the model is expressed or implemented in a computer language.
3) Compiler — A program that transforms the code into a series of machine instructions or program.
4) Processor — The processing unit that undertakes the machine instructions.
5) Cores — Each processor may have more than one core, i.e., distinct processing unit.
6) Optimization — A series of steps or techniques used by the compiler to maximize model performance. These typically are grouped in various levels (0, 1, 2, ...), which are progressively more "aggressive" and lead to better performance.
7) Parallelization — An approach whereby the calculations or tasks inherent in a code are concurrently distributed between available processors on a machine or machines. There are various methods to do this, in this paper, OpenMP is used. Parallelization of numerical codes can result in major performance increases.
8) Threads — OpenMP is an implementation of multithreading, a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them. The threads then run concurrently with the runtime environment allocating threads to different processors.
9) Scheduling — How these threads or tasks are organized or coordinated.
10) Serial code — A numerical code in which all instructions are executed consecutively with no parallelization.

Computational (non)determinism

A first step, or premise, in the discussion that follows is that a serial, unoptimized, and/or unparallelized version of any given numerical code (be it coded in Fortran, C, C++, etc.) has been rigorously checked and verified, as far as is humanly possible. This typically involves a large amount of fairly mind-numbing effort, but in the end, it always pays off. That being the case, it is the author's experience over many years of developing and running a variety of different numerical models that several issues are becoming more frequent and pertinent. When a particular numerical model, after compiler optimization and/or parallelization, is run under strictly identical initial and boundary conditions, using exactly the same version of the code, sometimes, the following can be observed:

1) Repeated, identical model runs on the same machine can produce different results.
2) Repeated, identical model runs on the same machine but using different numbers of processors/threads can produce different results.
3) Repeated, identical model runs on the same operating system but on different machines with slightly different processors can produce different results.
4) Repeated, identical models runs on different operating systems with either identical or similar processors can produce different results.
5) Model runs with identical inputs on the same machine but with different levels of optimization can produce different results.
6) Codes compiled under a seemingly identical set of options, but using different compilers or versions of compilers, can produce different model results.

These observations are perplexing and clearly present a challenge for the end user of either open-source/commercial software or the scientific coder/researcher. A basic, implicit, expectation when running such codes is reproducibility of model results. Thus, results are not expected to be different. This does not necessarily imply that the results are strikingly different (although they can be) but rather that the results are formally not the same. From a structural geology point of view, the interest might lie in why a particular fault initiated or became inactive at a particular location. Thus, consistency in the model results is required; not subtle differences that are related to technological issues and have nothing to do with the scientific question in hand. These effects are not necessarily heisenbugs, genuine bugs, or errors in code/logic that appear/disappear inconsistently when trying to correct them (see Raymond, 1996), but they definitely have the same level of frustration associated with them.

The discussion herein will confine itself to a limited subset of possible HPC approaches, but the observations do apply generally. They are based on the author's experience of developing such codes, and there is no intention to favor one compiler/platform/system over any another. The results have been generated using the Intel ICC compiler with OpenMP on Intel x86 under MacOS X and Linux. Obviously, there are other available compilers (e.g., gcc, PGI) and parallelization technologies (MPI 2.0, Grand Central Dispatch, and the Pthread standards).

Causes of nondeterminism

The observations above are quite troubling in that they raise questions and doubts regarding the veracity of numerical modeling results. Here, an attempt will be made to explain, or at least allay some fears over, the sources of these issues.

First, it is clear that precision in any computer is only finite, and thus, error, loss of significant bits, and rounding are facts of life in numerical floating-point calculations (Butt, 2009). However, what is being discussed here is the origin of numerical differences that may arise when repeatedly running a model with identical initial and boundary conditions. This is a very complex issue.

Here, no particular compiler or operating system will be assumed; these issues are quite generic and are often obscured to the casual, experienced, or even professional user. However, the objective is to list/explain, based on the author's experience, the most common sources of nondeterminism in such numerical codes. Nondeterminism can arise as a result of one, or more, of the following issues:

- Reduction summation order in OpenMP — In parallelized numerical codes, a floating-point sum, typically in a loop, may be broken down into n parallel tasks or partial sums. The OpenMP standard does not specify the order in which such partial sums should be subsequently combined. The order and sequence of threads may change dynamically in a single run and between identical runs. Clearly, with finite precision, this leads to a situation in which, because the order in which the partial sums are added together to form the final sum is undefined, the value of the final sum may vary from run to run. Although these reductions are equivalent mathematically, they are not equivalent numerically with finite precision arithmetic. Thus, reproducibility depends on how a reduction is implemented internally. Clearly then, a sum that is broken down into, e.g., 8, 12, 16, or 48 tasks or threads will also generate subtly different answers. If these final sums influence the ongoing behavior of the numerical model, the accumulated differences may be subtle but large enough to cause different results over a long model run. This is usually the cause of observations (1) and (2) above. Similar issues regarding reductions also exist in MPI (e.g., using MPIAllReduce; see Balaji and Kimpe, 2013):
- The floating-point model used — What is meant here are the semantics of floating-point calculations used during optimization. In some cases, though not all, the default floating-point model used by a given compiler is unclear. Typically, it is a fast model that uses more aggressive optimizations when implementing floating-point calculations. These optimizations increase speed, but they may affect the accuracy or reproducibility of floating-point computations because they may not be value safe. By reassociation and changing of parentheses, e.g., $(x+y)+z \rightarrow x+(y+z)$, multiplication by a reciprocal, e.g., $\mu/\phi \rightarrow \mu \times (1/\phi)$, approximations of the square root function, fusing of operations, etc., the compiler may generate a model that produces different results from the unoptimized version of the same code. In addition, intermediate, or temporary, store precision is very important when doing floating-point calculations but is often undefined or uncontrolled. The variations induced by these issues are often small; however, their impact on the final result of a longer calculation may be amplified if the algorithm involves cancellations (small differences of large numbers). This is typically the cause of observations (3, 4, and 5) above.
- No standard, defined math library for C. As yet, no official standard specifies the accuracy of mathematical functions, such as log() or sin(), or how the results should be rounded. Different implementations of these functions may not have the same accuracy or be rounded in the same way. Math libraries may also contain function implementations that are optimized differently for different processors and operating systems. In addition, the value returned by a math library function may vary between one compiler release and another due to the algorithmic and optimization improvements. Finally, if vectorization is used in the optimization scheme yet another math library may be used. This is often the cause of observations (3 and 4) above.
- Different compilers and compiler build versions — Slightly different code can be generated by different versions of the same compiler and between different compilers, even when using seemingly identical build options. This causes observation (6) above.

Illustration of nondeterminism

The effect of lack of control over determinism on numerical results will now be illustrated by presenting a series of examples from a numerical model of upper crustal deformation *cdem2D*. This model contains approximately 7000 lines, and it is written in ANSI Standard C with OpenMP parallelization. It is a discrete element model that may include breakable elastic bonds, friction, and cohesion at element contacts (Hardy, 2011). It is compiled using the Intel Compiler, ICC, version 14.0.3, on MacOS X and Linux. The particular code under consideration is documented by Hardy (2011, 2013) and Botter et al. (2014). Results are identical with or without OpenMP parallelization enabled, and the program is 64-bit executable. Luckily, this code does not use reductions in any loops that influence model evolution, and as a result there is no nondeterminism arising from OpenMP thread scheduling issues. Therefore, the results will be used to highlight differences solely due to the use of different levels of optimization and floating-point semantics.

First, cdem2D will be used to examine brittle cover deformation above a basement normal fault dipping at 65°. The model represents a section (the cover above the basement) of the upper crust (6.25×2.77 km) using approximately 47,000 elements. The assemblage modeled here has no elastic bonding, but it does have Mohr-Coulomb friction with cohesion at element contacts. The average element radius is 9.5 m with a density of 2500 kg/m^3. The particular model scenario is chosen because it well illustrates the structural differences arising from purely technological/computational issues.

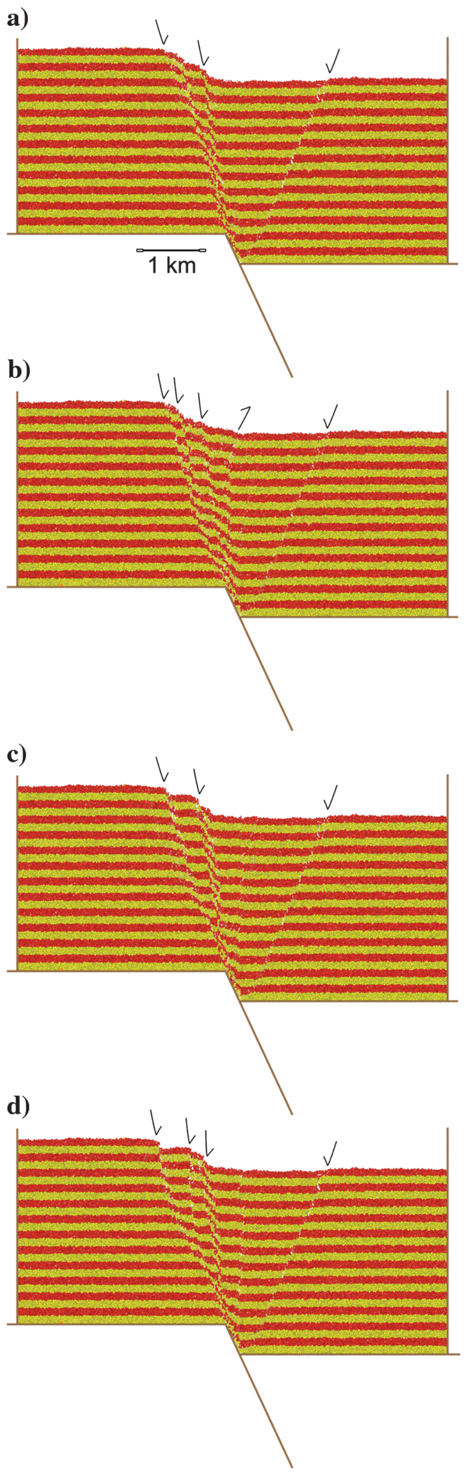

Figure 1. Results of a discrete element model of brittle cover deformation above a basement normal fault run with identical initial and boundary conditions but with different levels of optimization and compiler flags. (a) No optimization (O0), (b) level 1 optimization (O1), (c) level 2 optimization (O2), and (d) level 2 optimization, only value-safe optimizations, source code precision, and architecture consistency.

The results to be discussed were generated on a MacPro with a 12-core (24 thread) Xeon E5-2697 2.7 GHz processor running MacOS X Yosemite. This discussion will start by illustrating the results of a base model compiled with all compiler optimizations turned off, corresponding to the compiler switch/flag "O0." The deformation of the cover after 500 m of displacement on the basement fault is shown in Figure 1a. It can be observed that the basement fault passes upward into the cover as two main, closely spaced, synthetic, normal faults. In the upper cover, the footwall also contains a more subtle, smaller, normal fault. Because these cover faults are steeper than the basement fault, a clear antithetic fault has also developed in the hanging wall. The panel between the synthetic and antithetic faults is flat and undeformed. Fault offsets are clear, and fault zones are narrow. A model compiled with the level 1 compiler optimization enabled, corresponding to the compiler switch/flag "O1," leads to quite a different structural configuration (Figure 1b). The basement fault now links upward with three steep, synthetic, normal faults in the footwall. However, a prominent, steep, reverse fault has also now developed cutting the central panel above the basement fault. The antithetic fault in the hanging wall is still expressed strongly. Level 2 optimization of this code, corresponding to the compiler switch/flag "O2," leads to yet another, different structural configuration (Figure 1c). What can now be seen is a broad panel of strata between two prominent, synthetic footwall faults, a subtle vertical/reverse fault and, as before, the antithetic fault. Thus, each optimization level produces a different (yet geologically plausible) result. In a sense, each of these results is correct, in that it is exactly what the user (perhaps unwittingly) asked the compiler/system to do. However, whether they are desired is another issue.

These progressive differences are the result of different sets of value-unsafe optimizations, fast math libraries, and undefined floating-point models being used in the different levels of optimization (0, 1, and 2). Ideally, an approach that allows a definitive, consistent result would be desirable. When only value-safe optimizations are explicitly allowed, all intermediate values are required have the precision defined in the source code, and crossplatform consistency is insisted upon, a result that is consistent at all optimization levels and between MacOS X and Linux is produced (Figure 1d). For the Intel compiler, the set of switches used to produce this result were as follows (for the interested reader): icc -openmp -fp-model strict -fp-model source -fimf-arch-consistency=true -O2 cdem2D.c (set 1).

This result contains many of the characteristic features present in the other models (compare Figure 1a–1c with Figure 1d). However, the dominant footwall fault in the cover now has a marked listric character with consequent rotation of a panel of hanging-wall strata. This model result (Figure 1d) is produced at all levels of optimization on Linux and MacOS X, on 8-core (Intel Xeon X5570), 12-core (Intel Xeon X5650 and E5-2697), and 24-core (Intel Xeon E5-2697) machines. It is thus truly reproducible across all levels of optimization on three different processors and across two platforms. This result is perhaps no more geologically realistic than the others, but at least it is produced with no hidden compromises on precision and only with optimizations that are value safe. Thus, there is some certainty that the numerical model as described in the source code is being consistently honored. However, this consistency and reproducibility do come at a cost — Shown in Table 1 are runtimes for the models shown in Figure 1, and for comparison, the runtime without using OpenMP paralellization is also given. What can be seen is that increasing levels of optimization do produce remarkable speedups and that the penalty for insisting on only value-safe optimizations and architectural consistency is not too severe.

A second example is given in Figure 2 — Here, a simple caldera simulation has been run. All other model parameters are identical to the models shown in Figure 1, except that the model has a pistonlike subsiding block at its base. The piston block has subsided 1200 m to produce the observed deformation and caldera in the cover. In Figure 2a, the result obtained using only level 2 optimizations and no control over value safety, intermediate precision, or math libraries is shown, whereas in Figure 2b, the result obtained using the consistent set of compiler flags is shown (set 1 above). As before, it can be noted that although the results are similar, they are by no means the same. This is particularly marked on the right-hand margin of the caldera in which the style of faulting is quite different between the two models.

A final example, in a contractional setting, is presented in Figure 3. Here, the model setup and boundary conditions are quite distinct. In this example, the left-hand wall moves inward, toward the right, over a horizontal decollement, producing shortening in the cover. The model had initial dimensions of 25×2.7 km and an average particle radius of approximately 20 m. The base of the model has a coefficient of friction approximately a third of that of the internal, assembly, value. The total shortening produced is 3.5 km. All other model parameters are the same as in the other examples. As before, Figure 3a shows the result obtained using only level 2 optimizations and no control over value safety, intermediate precision, or math libraries, whereas Figure 3b shows the result obtained using the consistent set of compiler flags (set 1 above). The results are quite distinct in many minor details but particularly in the number of forethrusts developed: In Figure 3a, two forethrusts are seen, whereas in Figure 3b, three are developed. This is not a subtle difference in geometry, but a basic structural difference at this amount of shortening, And this difference is not due to any geologic or experimental change.

Table 1. A comparison of runtimes for the models shown in Figure 1 for a variety of different compilation options.

Compiler set used	Runtime
No OpenMP, O2 level optimizations	28 days
OpenMP, O0 optimizations	1 day, 5 h, 5 min
OpenMP, O1 optimizations	16 h, 4 min
OpenMP, O2 optimizations	8 h, 47 min
OpenMP, O2 optimizations, value safe, architectural consistency	10 h, 36 min

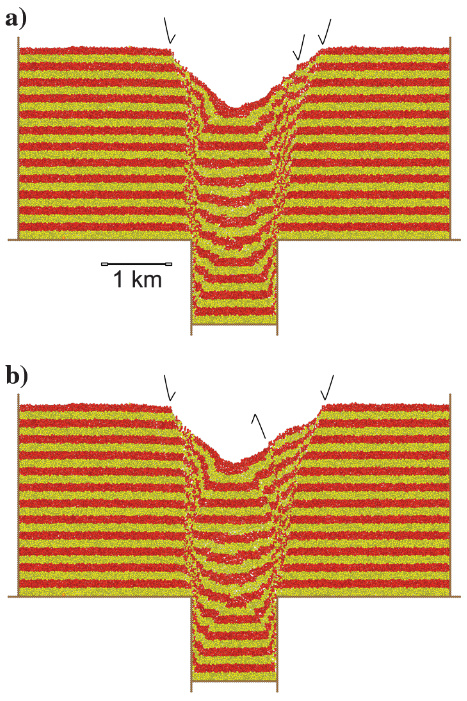

Figure 2. Results of a discrete element model of caldera formation run with identical initial and boundary conditions but with different levels of optimization and compiler flags. (a) Level 2 optimization (O2) and (b) level 2 optimization, only value-safe optimizations, source code precision, and architecture consistency.

A strategy for determinism and reproducibility

A basic strategy for identifying nondeterminism and, if possible, regaining reproducibility in codes such as the example given above will be proposed here. Several key steps, which are simple but effective, can lead to determinism and reproducibility. These steps variably apply to commercial, open source, and research codes. What is ultimately desired is *byte-for-byte compatibility* that can be achieved with care and attention to detail in the coding and compiler options (see section "Illustration of nondeterminism").

An initial series of steps that can identify possible nondeterminism in an optimized and/or parallelized program, and its origin, are as follows:

- Run an identical parallelized model repeatedly on the same machine.
- Run an identical parallelized model with a variable number of threads repeatedly on the same machine.
- Run an identical model with and without parallelization activated.
- Run an identical optimized model on multiple machines with different processors and/or operating systems.
- Compile the same model code on the same machine, incrementally using all levels of optimization. Compare identical sets of model runs under these different levels of optimization.

The results of such tests need to be analyzed in some tedious detail, but the "Devil really is in the detail." What needs to be done is not just a visual inspection or comparison of results. Results that are formally different can look extremely similar. Model results need to be compared numerically to pick up subtle, small differences that cannot be explained via rounding arguments. However, such steps will quickly identify if the program/code in use is subject to nondeterminism from whatever source.

Once any nondeterminism is identified, several steps can be taken to ensure that the level of reproducibility desired is regained. Much of this will depend on the user's ability to compile the code:

- First, restrict any optimizations used to those that are value safe and specify the precision of all intermediate results; with the Intel compiler, this is achieved via the fp-model switch — Insisting on strict and source, respectively, should go a long way toward ensuring the reproducibility of results.
- Second, the architecture independence flag (or its equivalent) appears to be crucial to ensure that a consistent math library is used by the program. With the Intel compiler, the switch -fimf-arch-consistency=true ensures bitwise consistency between results returned by math library functions on different processor types of the same architecture, including between results on Intel processors and on compatible, non-Intel processors. This switch may result in somewhat reduced performance because it results in calls to less optimized functions that can execute on a wide range of processors; but consistency is ensured.
- Third, make sure that all OpenMP reductions (or their MPI equivalents) are either value safe or do not directly influence model results.

Implications for the interpretation and use of model results in structural geology

The main implication for the use of such numerical modeling results in structural geology is the need for a full, deep understanding of the subtleties of compilation, optimization, and parallelization of any numerical code. Without this understanding, one loses control over the cause, and importance, of particular structural features produced when running a numerical model. If (for whatever reason) subtle nondeterminism is inevitable, a decision must be made about which result will be used, e.g., to condition a reservoir simulation model or act as input to a seismic simulation (see Figure 1). Similarly, with distributed teamwork, on different machines and systems, the issue is more about which result is the definitive result. On the upside, if such nondeterminism is inevitable, one could view such results as perhaps just reflecting natural variation and regard the results as a set of possible outcomes — Much the same way that repeated sandbox experiments produce subtly different results (Santimano et al., 2012). The objective then becomes to extract the key features or geometries produced in a set of models rather than focusing on an individual result.

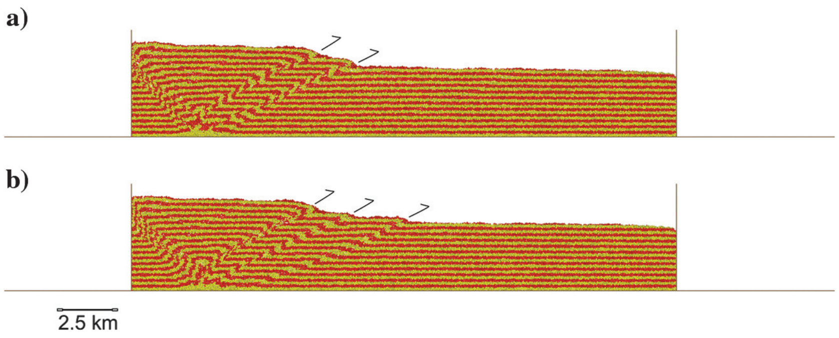

Figure 3. Results of a discrete element model of simple contraction above a horizontal decollement run with identical initial and boundary conditions but with different levels of optimization and compiler flags. (a) Level 2 optimization (O2) and (b) level 2 optimization, only value-safe optimizations, source code precision, and architecture consistency.

Why are such differences in structural geology numerical codes so marked? This is partly a result of the types of conceptual models used in mechanical models of deformation. These often include concepts such as threshold-limited frictional slip (Mohr-Coulomb) and elastic bonds between elements that have a finite breaking strain (see Hardy, 2011, 2013). Many of these conceptual models involve binary states, such that, once a threshold is passed, a change in the local force/stress balance arises and then conditions the subsequent deformation path. With subtly different numerical calculations, the precise path of, e.g., a propagating fracture may thus change and results can diverge markedly. These factors all combine to produce models results that are very sensitive to subtle differences during the calculation path and which ultimately influence the deformation path. Such issues have much less importance in kinematic models in which the deformation or restoration path is defined at the start of the model run, thus allowing no mechanical freedom for new faults or deformation styles to arise.

Conclusions

Undoubtedly, today's HPC techniques have allowed the production of numerical modeling results in timescales previously unthinkable and the ability to tackle problems that were not just not tractable. However, compilers and optimization/parallelization technologies are often complex, confusing, and opaque to the average user. As scientists, what we strive for is reproducibility, in order that imposed changes can be isolated and their results understood. Unfortunately, without proper control, optimization, and parallelization of codes can result in nondeterminism, and thus raise doubts as to the veracity of model results. Many of these issues are difficult to track down, but it is possible, with appropriate coding and compilation, to isolate and correct them. From my own experience, nothing short of exhaustive testing of research, open source, or commercial codes is sufficient. Ironically, attempts to isolate and control nondeterminism in a given code (by changing numbers of threads, checking precision, etc.) can often prove to be a useful diagnostic tool that reveals genuine, subtle programming errors.

Acknowledgments

This short paper is the result of much head scratching but ultimately comprehension. Two people have guided me on this path regarding attention to detail: D. Waltham and J. Suppe. N. Cardozo is thanked for comments on an earlier version of this manuscript. Comments by reviewers H. Broichhausen, S. Abe, E. Finch, and the associate editor helped to improve the manuscript.

References

Balaji, P., and D. Kimpe, 2013, On the reproducibility of MPI reduction operations: High performance computing and communications (HPCC/EUCC 2013), 407–414, http://www.mcs.anl.gov/papers/P4093-0713_1.pdf, accessed 14 January 2015.

Botter, C., N. Cardozo, S. Hardy, I. Lecomte, and A. Escalona, 2014, From mechanical modeling to seismic imaging of faults: A synthetic workflow to study the impact of faults on seismic: Marine and Petroleum Geology, **57**, 187–207, doi: 10.1016/j.marpetgeo.2014.05.013.

Butt, R., 2009, Introduction to numerical analysis using MATLAB: Jones & Bartlett Learning.

Hardy, S., 2011, Cover deformation above steep, basement normal faults: Insights from 2D discrete element modeling: Marine and Petroleum Geology, **28**, 966–972, doi: 10.1016/j.marpetgeo.2010.11.005.

Hardy, S., 2013, Propagation of blind normal faults to the surface in basaltic sequences: Insights from 2D discrete element modelling: Marine and Petroleum Geology, **48**, 149–159, doi: 10.1016/j.marpetgeo.2013.08.012.

Kellert, S. H., 1993, In the wake of chaos: Unpredictable order in dynamical systems: University of Chicago Press.

Li, Q., M. Liu, and H. Zhang, 2009, A 3-D viscoelastoplastic model for simulating long-term slip on non-planar faults: Geophysical Journal International, **176**, 293–306, doi: 10.1111/j.1365-246X.2008.03962.x.

McMillan, R., 2011, Prof promises supercomputer on every desktop: Wired, http://www.wired.com/2011/12/vt-supercomputer/, accessed 14 January 2015.

Raymond, E. S., 1996, The new hacker's dictionary (3rd ed.): MIT Press.

Santimano, T., M. Rosenau, M. Morlock, and O. Oncken, 2012, On the precision of sandbox experiments — Insight from test-retest variability: General Assembly European Geosciences Union, Geophysical Research Abstracts, EGU2012-7667-2.

Stuart Hardy received B.S., M.S., and Ph.D. degrees. He is an ICREA research professor at the Faultat de Geologia, Universitat de Barcelona. With a background in both geology and computational science, his research focused on numerical modeling of geological processes. His research interests include high-performance computing, salt tectonics, deltaic sedimentation, seismic simulation, fault propagation, and fault-related folding.