# Towards Rapid Geoscience Model Development – The Snark Project

Bill Appelbe[(1)], David May[(1)], Steve Quenette[(1)],
Siew-Ching Tang[(1)], Feng Wang[(1)] and Louis Moresi[(2)]

(1) Victorian Partnership for Advanced Computing, 110 Victoria Street, PO Box 210, Carlton South, Victoria 3053, Australia (e-mail: bill@vpac.org; phone: +61 3-9925-4648). (2) School of Mathematical Sciences, Monash University, Clayton, Victoria, Australia (e-mail: louis.moresi@sci.monash.edu.au; phone: +61 3-9905-4468).

## Abstract

**The primary goal of the Snark project is developing geoscience modeling software that is scaleable and adaptable. The test of whether Snark software is meeting these goals is the time required by geoscientists to build, test, and evaluate new models. By new models, we do not mean just different input data sets, or slight changes of critical parameters. Rather, we mean major changes in the underlying constitutive laws, or the techniques used to solve the problem. At present, the time required to build new geoscience models is of the order of months, often stretching to years. Our goal is to reduce this to days. The technology we exploit to achieve such adaptability is component–based software design and scripting languages for model customization**

## Introduction

The underlying physics, or constitutive laws, that govern geodynamics are poorly understood and operate on dimensions ranging from millimeters to thousands of kilometers. A wide range of numerical techniques are being used investigate how to effectively model geological deformation. The time required to build new *quantitative geodynamic models* ("geomodels") is thus an obvious problem, with a big impact on scientific research productivity. Ideally, a geomodeller should be able to very rapidly try out new models, and generate those models from very simple inputs of the constitutive laws. The ideal geomodelling tool might be something like a cross between Mathematica and Fluent, where the input to the model would be equations, initial conditions, and grids. However, the catch-22 is that it is only feasible to build such sophisticated tools when the requirements of geoscience are well-understood and stable.

While it is not feasible to build a "point and click" general-purpose geomodelling tool, it is feasible to build software that is far more scaleable and adaptable than any geomodelling software in use at present. At present, almost all geomodelling software is only applicable to a narrow range of constitutive laws and input conditions, not written to

be either portable or adaptable. The key technologies needed to build scaleable and adaptable software are well known from other disciplines:

1. *Component-based* development – build software on top of existing modular libraries and tools. Maximize the reuse of existing software and minimize the new software that must be created to build new models

2. *Script based* customization – develop new models by writing small scripts, rather than by custom coding changes at random to the application

3. *User Documentation* – to minimize the time required by geomodellers to understand how to use the software

Component-based development is the key to reducing software development time and productivity. In industry, software development for everything from websites and databases to payroll systems, is all now based upon component-based development. Commercial software is no longer written from scratch in COBOL. By contrast, the overwhelming majority of geomodelling software is written from scratch in Fortran or C – languages that have changed little in decades. Part of the problem has been that until recently there has been very little advances in libraries or tools for generic scientific software, beyond, say the BLAS or NAG libraries which were designed to make vector hardware accessible to scientific programmers. Efficient parallel libraries with comparable ease-of-use to BLAS or NAG have not been available until very recently.

For example, PETSc now offers a well-supported and extensible library that can automate much of geoscience. PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. (http://www-fp.mcs.anl.gov/petsc/). PETSc connects to packaged components for ancillary tasks such as grid generation and decomposition, and visualization. The key downside to PETSc is the "learning curve". While PETSc is well documented, it takes months of use to get comfortable with writing applications from scratch in PETSc.

Script-based customization complements component-based development. Instead of having to understand existing source code; then edit, compile, and link code changes, scripting allows a programmer to customize programs without understanding the internals. Scripting is common in commercial software. The zenith of scripting is application generators and "point and click" tools but these require an order of magnitude more effort than scripting tools. Some geomodelling software, such as LSMearth, do have simple scripting capabilities built-in.

Documentation is a sore-point in geomodelling software. Lack of documentation makes software useable and useful only to experts. Another piece to the puzzle is what software development methodology to adopt to maximize outcomes and minimize risk. Traditionally, scientific software has been managed in a laissez-faire academic environment, where turnaround and delivery of new code in measured in months. Modern software development emphasizes incremental delivery, very rapid turnaround and ongoing testing of code, in times measured in days rather than months (see http://www.agilealliance.org).

## The Snark Project – Technical Design

The overall *software architecture* of most geoscience applications are striking similar. For example, there is a fundamental duality between particles and grids. Any particle can be viewed as a point in an irregular grid, and conversely. Software architecture means the major subsystems and how they interact. Major geoscience software subsystems typically include:

1. Generation and refinement of grids (or particle distributions)
2. Distribution of grid points and/or particles across processors
3. Linear (Ax = B) and non-linear solvers, including preconditioners
4. Local particle interaction and particle tracking
5. Visualization
6. Initialization of boundary conditions and parameters

PETSc provides libraries for the core linear solvers (#4) and a host of ancillary utilities. Tools are available for many other subsystems (such as #1, #2, #5, and #6), that interface well with PETSc (see http://www.vpac.org/snark/collab for a summary of these tools). Essentially the only subsystem that is not covered by PETSc is particle interaction and tracking, and initialization of boundary conditions. These components are typically less than 10% of the applications code, and the PETSc library is extensible to deal with them.

It is important to note that the source code of PETSc itself is not the "holy grail". The PETSc interface is more important. By using a "high-level" library such as PETSc to encapsulate functionality, enormous simplification in program design becomes feasible. The software architecture of Snark can be extended to cover new application areas. For example, the PETSc team is currently extending the library to include mesh and grid support.

The software architecture of Snark can be viewed in one of two ways. Firstly, as a flowchart of processing steps:

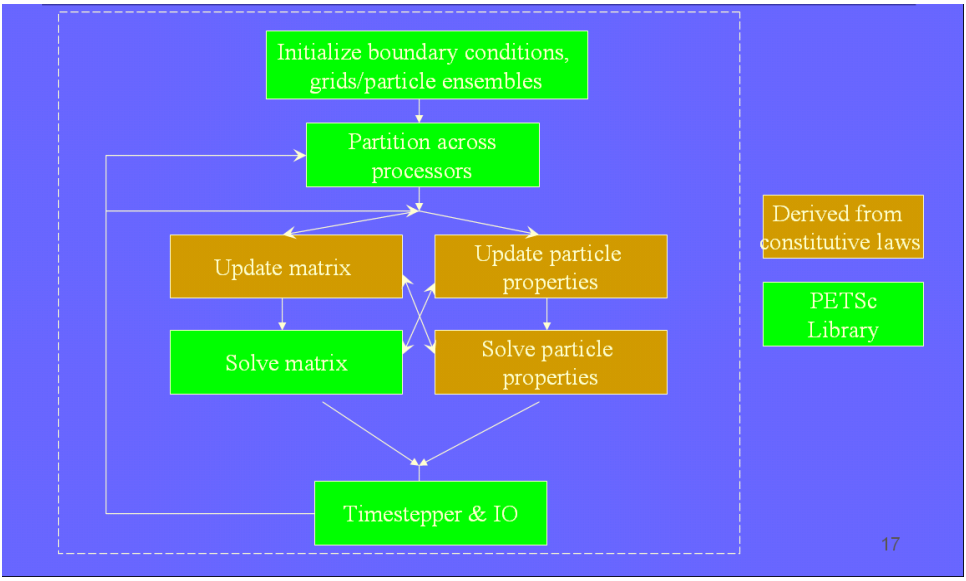


Figure 1 – The Snark Flowchart

The overall flowchart shows the fundamental flow of most numerical models for solving systems of partial differential equations via either matrix based methods (continuum approaches), particle based approaches, and combinations of these (such as Ellipsis). The flowchart ignores details such as pre and postprocessing (visualization). The boxes denote major subsystems, and the arrows denote flow of data and control. Our view of particles is that they are the same as irregular grid points, and the PETSc functionality that supports irregular grids can be extended to cover particles.

Another view of the software architecture is the package view. What major packages and tools are being used and how do they interact with the PETSc library?

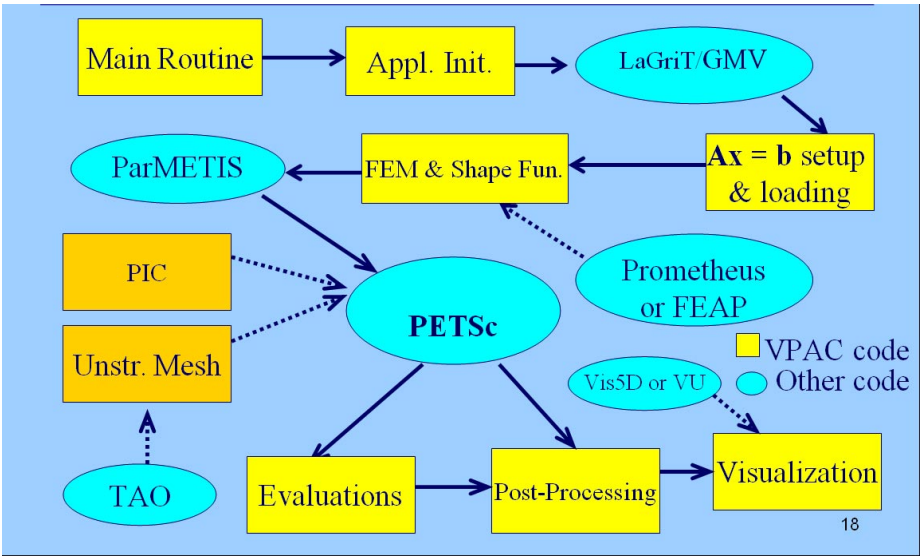


Figure 2 –Snark Tools

The tools above, and their use in Snark, are all discussed in detail on the Snark website (see http://vpac.org/snark/collab/, or access the PETSc website which links to many of these tools.

## The Snark Project – Implementation

The development of Snark is by a software development process called iterative development. The fundamental idea is design, development, testing, and delivery should be done on very short time scales. In essence, a new version of Snark is delivered each week, and there is a process of continuous code improvement. As code is developed, there is corresponding development of detailed documentation on the science (constitutive laws), numerical methods used to solve those laws, and the exact implementation in the program. Obviously, if code is in continuous development/improvement, there is a need to have fixed, stable versions of Snark available for others to download, test and use. We call such fixed versions "frozen codes". They are essentially a package of code, makefiles, test data, and documentation that is known to be tested and operational.

The first frozen code was delivered in early June. It is essentially a 2D cavity driven stokes flow, with particle tracking. This version is a useful validation of the whole code, as there are well published and known results for this problem, which we used to validate the correctness of the code. After that version, we have made major improvements in the memory allocation and organization of the code to make better use of the PETSc library, while simultaneously changing the code to use an alternative PETSc solver (Conjugate Gradient), and incorporate and XML scripting tool for specifying program parameters. By the end of July 2002 we project that a full 3D version of this solver will be frozen.

Planned development is that all of the functionality currently in Ellipsis will now be available in Snark by about the end of August 2002. The total size of the code is only

about 4,000 LOC (Lines of Code). This contrasts with a typical hand-written, from scratch solver for the same functionality of about 50,000 LOC.

## Acknowledgments

This research was funded by the Victorian Partnership for Advanced Computing, though grants from the state government and APAC, the Australian Partnership

## References

[1] L. Moresi, H.-B. Mühlhaus, and F. Dufour. *Particle-in-cell solutions for creeping viscous flows with internal interfaces*. In H.-B. M¨uhlhaus, A. Dyskin, and E. Pasternak, editors, In Bifurcation and Localization in Soils and Rocks, pages 345–354, Rotterdam, 2001a. Balkema

[2] Mora, P., Place, D., Abe, S. & Jaume, S. (2000) *Lattice solid simulation of the physics of earthquakes: the model, results and directions* , in: GeoComplexity and the Physics of Earthquakes (Geophysical Monograph series;