Coordinate-free numerics: all your variation points for free?*

Magne Haveraaen
Department of Informatics
University of Bergen
Norway

Helmer André Friis
IRIS research
Stavanger
Norway

Hans Munthe-Kaas
Department of Mathematics
University of Bergen
Norway

5th March 2007

1 Introduction

In the object oriented literature the notion of a variation point is important. A variation point is where we expect to, or, in retrospect, are forced to, accommodate alternative solutions. Numerical software, such as solvers for partial differential equations (PDEs), are full of variation points. We observe this from how we speak about PDEs.

Firstly, we speak about a multitude of PDEs to describe the various physical applications, e.g., the Navier-Stokes equations for fluid flow problems, the elastic wave equation for seismic simulation etc. Moreover, it is quite typical that mathematical modelling connected to a physical application may require various extensions of the "basic" PDE used to describe the problem, in order to include more physically realistic effects. Examples are the introduction of physical diffusion in inviscid compressible fluid flow, the introduction of temperature effects in isentropic fluid flow or the consideration of porous media effects starting from the usual elastic wave motion. So the first variation point is the choice of PDE.

The numerical discretisation process also obviously constitutes a rich source of variation points. As an example, the common numerical methods such as the finite element, finite volume and finite difference methods exist in a vast number of different variants, which also may be more or less associated with the PDE we want to solve, the type of numerical time-stepping technique, the underlying (structured or unstructured) numerical grid and so on.

Other variation points deal with computational simplifications of the problem: number of dimensions, Cartesian versus cylindrical versus spherical versus curvilinear coordinate system, symmetries (e.g., axio-symmetric), the actual size of the problem (i.e., the number of data points), and sequential versus parallel.

In the traditional development of numerical software we make our pick of each of these alternatives. Of course the choices are well motivated by the problem we want to solve. To some extent, we may also decide for certain variations. Normally we will fully parameterise the data set size. We may choose to cater for 2D and 3D. If the circumstances are special we

*Supported in part by The Research Council of Norway through the SAGA-GEO research grant and a donation of CPU resources through the NOTUR programme.

`Mesh`	D 2000				
	S 2000	P 2700			
`Tn`	D 1700				
	SFD 1600				
`Bn`	D 1500				
	S 1900	U 2100			
`Tensor`	SI 2200	STI 2200	STA 3000	CI 2200	CTI 2300
`Seismod`	SE 600	PE 700			

Figure 1: Indicative sizes of the modules in lines of code based on [HFJ99].

may also build other variation points into the software, but this used to be rare and require special motivation.

If, at a later stage, demands going beyond the requirements originally taken into account arose, one would normally redo the complete software development. The structural relationship between the software and the original choices would be too close to merit any reengineering of the original code in face of the new requirements. This shows up with the difficulty of, e.g., parallelising existing codes, or going from one PDE to another.

Modern software technologies, such as object orientation and feature modelling [CE00], focuses on building variation points into software. Object oriented numerical software, such as DiffPack, [BL97] and see `http://www.diffpack.com/`, shows the benefits of this technology. DiffPack treats the choice of discretisation method as a feature, such that choosing a numerical method does not imply reimplementing it, but only providing the problem specific details. The discretisation method is reused across many problems. Also the ease of parallelising DiffPack code shows how many important variation points are supported.

2 Coordinate-free numerics

The topic coordinate-free numerics [MKH96, Bjø00] was developed from the motivation of simplifying tensor calculus notation. This initiated a broader investigation of the PDE domain concepts using a methodology from algebraic specifications [Hav00a]. The result was a radical redesign of the PDE domain structure.

At the core of this redesign is the notion of a scalar-field which bridges the gap between the discrete data and the continuous computation [HFMK05], and the tensor field which bridges the gap between the coordinate representation of the computation fields and their coordinate free presentation. On top of the tensors the coordinate-free PDE is formulated and the timestepping solvers are written. In the implementation of the scalar-field the numerical methods excel, e.g., finite difference, volume and element methods. All these are supported by meshes efficiently handling vast amounts of discrete data points.

This redesign naturally gives software with a large amount of variation points, and a significant flexibility. Figure 1 shows this for a seismic simulation code. The modules presented in the figure are:

- `Mesh` deals with storing (discrete) data in arrays.
  D: domain information, S: sequential processing, P: parallel processing.

- `Tn` provides the bridge between the discrete and the continuous (the scalarfield) domains.
  D: domain information, SFD: staggered finite difference method.
- `Bn` defines boundary handling
  D: domain information, S: bounded box including sea surface, U: bounded box completely under the surface.
- `Tensor` provides the bridge between the coordinate representation and the coordinate-free abstraction.
  SI: standard Cartesian isotropic, STI: standard Cartesian transverse isotropic, STA: standard Cartesian transverse isotropic with arbitrary symmetry axis, CI: cylindrical symmetry isotropic, CTI: cylindrical symmetry transverse isotropic
- `Seismod` is the timestepping seismic equation solver
  SE: standard elastic, PE: poro-elastic

The figure shows interchangeable modules at every row, s.t. a complete program is created by choosing a module from each row. This gives us 40 variants of the program, but `Bn` S was never combined with `Tensor` CI or CTI, yielding 32 combinations which were built. Note how small the code variant extension are, typically 2000-3000 lines of code, compared to the size of a single program, around 13500 lines of code. This illustrates the incredible power of developing mix-and-match modules. And for all these cases we can easily add more alternatives, e.g., a standard finite difference implementation `Tn` FD as an alternative to the staggered finite difference implementation `Tn` SFD.

Most noticeable is the ease we had in moving between elastic and poro-elastic equations. The standard mathematical formulation of the elastic wave equation is

$$\begin{aligned} \rho \frac{\partial^2 \vec{u}}{\partial t^2} &= \nabla \cdot \sigma + \vec{f}(t), \\ \sigma &= \Lambda(e), \\ e &= \mathcal{L}_{\vec{u}}(g). \end{aligned} \tag{1}$$

The scalar field density ρ and the stiffness tensor field Λ are given data that vary within the physical domain, in accordance with the varying geophysical properties of the rocks. The particle displacement vector field $\vec{u}$ represents the propagation of the seismic wave and will be recomputed at every iteration of the solver algorithm. The tensor field g defines the coordinate system used, the tensor fields σ and e are computed intermediate values, and $\vec{f}(t)$ is a time-varying vector field representing the forces from the seismic explosion that sets off the seismic wave. The $\nabla\cdot$ and $\mathcal{L}_{\vec{u}}$ are derivation operators, the latter dependent on the displacement $\vec{u}$.

The coupled set of equations to be considered for the poro-elastic wave equations is

$$\begin{aligned} \rho_{11} \frac{\partial^2 \vec{u}}{\partial t^2} + \rho_{12} \frac{\partial^2 \vec{U}}{\partial t^2} &= \nabla \cdot \sigma + \vec{f}(t) + b(\frac{\partial \vec{U}}{\partial t} - \frac{\partial \vec{u}}{\partial t}), \\ \rho_{12} \frac{\partial^2 \vec{u}}{\partial t^2} + \rho_{22} \frac{\partial^2 \vec{U}}{\partial t^2} &= \nabla s + \vec{f}(t) - b(\frac{\partial \vec{U}}{\partial t} - \frac{\partial \vec{u}}{\partial t}), \\ \sigma &= \Lambda(e) + Q(\epsilon), \\ s &= M(e) + R(\epsilon), \\ e &= \mathcal{L}_{\vec{u}}(g), \\ \epsilon &= \nabla \cdot \vec{U}. \end{aligned} \tag{2}$$

Here the displacement vector fields are $\vec{u}$ for the solid and $\vec{U}$ for the fluid. Likewise the tensor fields σ and e of Equation 1 split into σ and s and e and ϵ. Further the stiffness tensor field

Λ is split into four tensor fields Λ, Q, M, R representing the stiffness of the two phases and their interaction. We also get three density coefficients ρ_{ij} and a friction tensor field b. The operators remain basically the same, but we get an additional derivation operation ∇.

As can be seen the two problems have rather different equations, and our claim is that, while in the coordinate-free case we can reprogram from one to the other by changing 600-700 lines of code (`Seismod` SE vs. PE), standard numerical software development practices probably would have required a full redevelopment of the software when moving from elastic to poro-elastic. We would expect object-oriented numerical software, such as DiffPack, to do much better than latter, although we still think the coordinate-free approach would be significantly better.

Since the development of these codes we have seen that there are large portions within the code base which either are verbatim copies of each other, or are very close to being identical. For instance, much of the domain description code is the same between the discrete, the continuous and the boundary sensitive levels. Likewise, much of the code in the variants of the tensor module is the same. Using generative techniques we hope to eliminate much of this duplication, thus significantly reducing the size of the code base, and perhaps even increasing our flexibility in building programs in the process.

One draw-back of a heavy focus on the use of abstraction in numerical software is the resulting run-time overhead. We are currently addressing this issue. First we will do a side-by-side comparison with traditionally developed codes, then we will investigate techniques for improving the efficiency of the coordinate-free codes without compromising the effectiveness in building software.

3 Future challenges

Currently high performance computing is facing severe challenges from the hardware industry. For decades we have relied on increase in processor speed on top of parallelism by MPI or OpenMP for our computational needs. Now it seems that processor speed is levelling out.

Instead, increased computational power comes from multi-core processors (100s of cores in a processor may appear within a few years), and specialised, programmable co-processors (confer SGI's use of FPGA coprocessors in some of their recent computer models). On par with this, game and graphics processors (GPU) are appearing with unprecedented floating point processing characteristics.

The problem is: how can we utilise this new source of processing power? We are confident that the focus coordinate-free numerics has given on collective operations on meshes [Hav00b] level will allow us to utilise multi-core quite easily.

4 Conclusion

We have presented the notion of variation points, and showed how coordinate-free numerics deals with most of the known PDE variation points in a natural way. Emerging hardware paradigms such as multicore processors and GPUs also seem to fit naturally with these variation points.

Much more challenging is the use of programmable hardware such as FPGAs.

References

[Bjø00] Petter E. Bjørstad. Special issue on coordinate-free numerics. *Scientific Programming*, 8(4), 2000.

[BL97] Are Magnus Bruaset and Hans Petter Langtangen. A comprehensive set of tools for solving partial differential equations; Diffpack. In Morten Dæhlen and Aslak Tveito, editors, *Numerical Methods and Software Tools in Industrial Mathematics*, pages 61–90. Birkhäuser, Boston, 1997.

[CE00] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative programming: methods, tools, and applications*. Addison-Wesley, 2000.

[Hav00a] Magne Haveraaen. Case study on algebraic software methodologies for scientific computing. *Scientific Programming*, 8(4):261–273, 2000.

[Hav00b] Magne Haveraaen. Machine and collection abstractions for user-implemented data-parallel programming. *Scientific Programming*, 8(4):231–246, 2000.

[HFJ99] Magne Haveraaen, Helmer André Friis, and Tor Arne Johansen. Formal software engineering for computational modelling. *Nordic Journal of Computing*, 6(3):241–270, 1999.

[HFMK05] Magne Haveraaen, Helmer André Friis, and Hans Munthe-Kaas. Computable scalar fields: a basis for PDE software. *Journal of Logic and Algebraic Programming*, 65(1):36–49, September-October 2005.

[MKH96] Hans Munthe-Kaas and Magne Haveraaen. Coordinate free numerics — closing the gap between 'pure' and 'applied' mathematics? *Zeitschrift für Angewandte Mathematik und Mechanik*, 76, supplement 1:487–488, 1996.