Domain specific languages and automated code generation: high expressiveness and high performance

P. E. Farrell[1,2]

[1] University of Oxford

[2] Simula Research Laboratory, Oslo

March 17, 2015

A quote

Mathematical languages and code generation

[A]n automatically coded problem, which has been concisely stated in a language which does not resemble a machine language, will be executed in about the same time that would be required had the problem been laboriously hand coded.

...

Such a system will make experimental investigation of various mathematical models and numerical methods more feasible and convenient both in human and economic terms.

— John Backus, Specifications for the IBM Mathematical Formula Translating System, 1954

Main idea of this talk

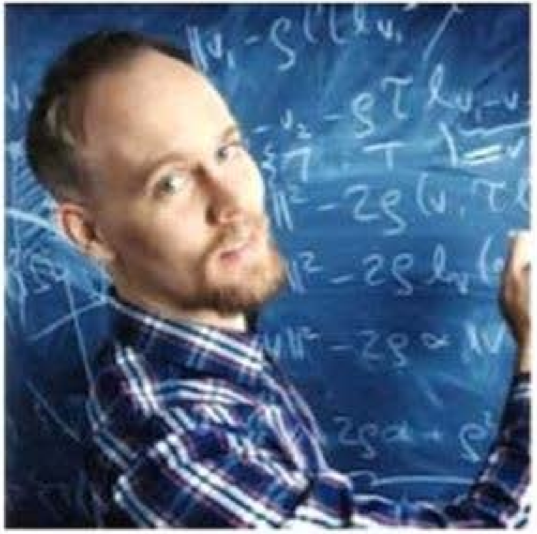
Main idea

Represent the variational problem to be solved as data.

- Generate C++ code with a special compiler.
- Developing finite element models becomes significantly faster.
- Generated code can run faster (than busy humans would bother).
- This enables lots of automatic **program transformations**!

The people responsible

(a) Martin Alnæs



(b) Anders Logg

(c) Garth Wells

Maths and code: I

Start with the strong equation:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

Multiply by a test function and integrate over the domain:

$$-\int_\Omega (\Delta u) v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x$$

Integrate by parts and set $v = 0$ on the Dirichlet boundary:

$$\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x = \int_\Omega f v \,\mathrm{d}x$$

In code:

```
F = inner(grad(u), grad(v))*dx - inner(f, v)*dx
```

Maths and code: II

Too simple? Let's make it harder:

$$\begin{aligned} -\nabla \cdot (\gamma(u) \nabla u) &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

where

$$\gamma(u) = (\epsilon^2 + \frac{1}{2} |\nabla u|^2)^{(p-2)/2}$$

Maths and code: III

Coupled problems? Let's try Navier–Stokes:

$$
\begin{aligned}
-\frac{1}{\mathrm{Re}}\nabla^2 u + u \cdot \nabla u + \nabla p &= 0, \\
\nabla \cdot u &= 0,
\end{aligned}
$$

Maths and code: IV

Optimisation constrained by an eigenvalue problem?

$$\begin{aligned}
\text{minimise} \quad & \int_\Omega \phi \\
\text{subject to} \quad -\nabla^2\phi &= \lambda\phi \qquad \text{in } \Omega \\
\phi &= 0 \qquad \text{on } \delta\Omega \\
\int_\Omega \phi^2 &= 1
\end{aligned}$$

I: Jacobian calculation

Mathematical idea

Given the problem residual F, calculate $J = F'$.

Jacobian calculation

Forming each element of J requires taking analytic or discrete derivatives of the system of equations with respect to u. This can be both error-prone and time consuming.

— Knoll and Keyes, Jacobian–free Newton–Krylov methods, 2004

Page 21 of 52

II: tangent predictors

Mathematical idea

Given a change to a parameter δm, what will be the linearised change in solution δu?

$$F(u, m) = 0$$

$$\implies \frac{\partial F}{\partial u}\delta u + \frac{\partial F}{\partial m}\delta m = 0$$

III: deflation

Mathematical idea

Given

- the problem residual $F : V \to W$
- a solution $r \in V$, $F(r) = 0$, $F'(r)$ nonsingular
- $\tilde{r} \in V$, $\tilde{r} \neq r$

construct a **new nonlinear problem** $G : V \to Z$ such that:

- (Preservation of solutions.) $F(\tilde{r}) = 0 \iff G(\tilde{r}) = 0.$
- (Deflation property.) Newton's method applied to G will never converge to r again, starting from any initial guess.

IV: Why are transient adjoints hard?

- Adjoints *reverse propagation of information*:
  - IVPs induce terminal-value problems
  - Parallel communication flows the other way
- Precise form depends sensitively on the problem
  - Must be modified whenever PDE, discretisation, parameter, prior, likelihood change
- Practical implementation requires checkpointing
  - Control flow must weave between forward and adjoint solution
  - Delicately balance memory and disk I/O
  - Expert knowledge required to make this work on HPC

Conclusion

Adjoint derivation **should be automated**.

IV: transient adjoint derivation

$$
\begin{tikzcd}
\boxed{\text{discrete forward equations}} \arrow[r, "\text{implement}"] & \boxed{\text{forward code}} \arrow[d, "\text{adjoin}"'] \\
& \boxed{\text{adjoint code}}
\end{tikzcd}
$$

Difficulties

- Loses mathematical structure of problem
- Usually very inefficient (Naumann (2011): 3–30$\times$ slower)
- Major intervention to work in parallel

IV: transient adjoint derivation

$$
\begin{tikzcd}
\boxed{\text{discrete forward equations}} \arrow[r, "\text{implement}"] & \boxed{\text{forward code}} \arrow[d, "\text{adjoin}"'] \\
& \boxed{\text{adjoint code}}
\end{tikzcd}
$$

A better idea

Adjoin the equations, not the code!

IV: transient adjoint derivation

$$
\begin{tikzcd}
\boxed{\text{discrete forward equations}} \arrow[r, "\text{code generation}"] \arrow[d, "\text{symbolic manipulation}"] & \boxed{\text{forward code}} \\
\boxed{\text{discrete adjoint equations}} \arrow[r, "\text{code generation}"] & \boxed{\text{adjoint code}}
\end{tikzcd}
$$

A better idea

Adjoin the equations, not the code!

IV: transient adjoint derivation

$$\begin{tikzcd}
\boxed{\text{discrete forward equations}} \arrow[r, "\text{code generation}"] \arrow[d, "\text{symbolic manipulation}"] & \boxed{\text{forward code}} \\
\boxed{\text{discrete adjoint equations}} \arrow[r, "\text{code generation}"] & \boxed{\text{adjoint code}}
\end{tikzcd}$$

Advantages

- Retains mathematical structure of problem
- Achieves optimal theoretical performance for adjoint
- Works naturally in parallel

IV: dolfin-adjoint

dolfin-adjoint takes the adjoint of FEniCS models.

```
from dolfin import *

F = ...

while t < T:
    solve(F == 0, u)
```

IV: optimisations

Optimisations

Having the high-level structure available allows for many optimisations that are very difficult to do in general.

IV: two-phase linearisation

Forward problem

Solve $F(u, m) = 0$ (taking N linear solves).

Piggyback linearisation

Differentiate through each of the N iterations.

Page 44 of 52

IV: two-phase linearisation

Forward problem

Solve $F(u, m) = 0$ (taking N linear solves).

Two-phase linearisation

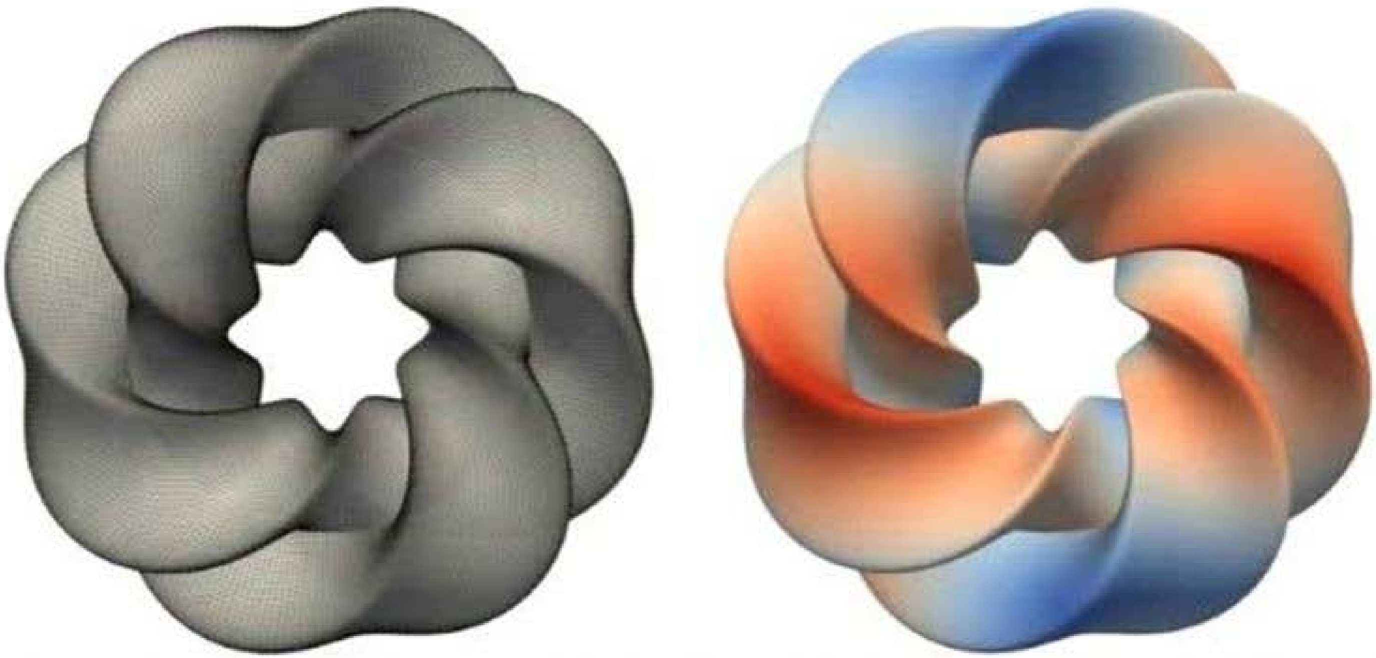
Solve in one iteration

$$\frac{\partial F}{\partial u}\dot{u} = -\frac{\partial F}{\partial m}\dot{m}.$$

Advantage

A huge gain in efficiency ($\propto$ number of nonlinear iterations)

IV: two-phase linearisation of the p-Laplace equation



p-Laplace equation

$$-\nabla \cdot \left(\underbrace{\left(\epsilon^2 + \frac{1}{2}\,|\nabla u|^2\right)^{p-2/2}}_{\gamma(u)}\ \nabla u\right) = f$$

IV: two-phase linearisation of the p-Laplace equation

Operation	Time (s)	R
forward model	2949.8	1
Piggyback	2890.7	0.9799
Two-phase	14.3	**0.0048**

Conclusions

- Domain specific languages allow for **huge productivity gains**.