Available online at www.sciencedirect.com

ScienceDirect

Comput. Methods Appl. Mech. Engrg. 276 (2014) 95–121

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Rapid development and adjoining of transient finite element models

J.R. Maddison[a,b,*], P.E. Farrell[c,d]

[a] School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom
[b] Atmospheric, Oceanic and Planetary Physics, Department of Physics, University of Oxford, Oxford OX1 3PU, United Kingdom
[c] Mathematical Institute, University of Oxford, Oxford OX2 6GG, United Kingdom
[d] Center for Biomedical Computing, Simula Research Laboratory, 1325 Lysaker, Norway

Received 20 June 2013; received in revised form 2 January 2014; accepted 24 March 2014
Available online 1 April 2014

Abstract

Recent advances in high level finite element systems have allowed for the symbolic representation of discretisations and their efficient automated implementation as model source code. This allows for the extremely compact implementation of complex non-linear models in a handful of lines of high level code. In this work we extend the high level finite element FEniCS system to introduce an abstract representation of the temporal discretisation: this enables the similarly rapid development of transient finite element models. Efficiency is achieved via aggressive optimisations that exploit the temporal structure, such as automated pre-assembly and caching of forms, and the robust re-use of matrix factorisations and preconditioner data. The resulting models are as fast or faster than hand-optimised finite element codes. The high level representation of the system remains extremely compact and easily manipulated. This structure is exploited to derive the associated discrete adjoint model automatically, with the adjoint model inheriting the performance advantages of the forward model. Combined, this provides a system for the rapid development of efficient transient models, together with their discrete adjoints.

Keywords: Automated code generation; Finite element method; Discrete adjoint; Navier–Stokes; Barotropic vorticity; FEniCS

1. Introduction

1.1. Automated code generation in computational science

Automated code generation is a crucial tool in computational science, as it allows scientists and engineers to express the structure of an algorithm in notation close to its mathematical formulation. It raises the level of

* Corresponding author at: School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom. Tel.: +44 131 6505036.
E-mail addresses: j.r.maddison@ed.ac.uk (J.R. Maddison), patrick.farrell@maths.ox.ac.uk (P.E. Farrell).

http://dx.doi.org/10.1016/j.cma.2014.03.010
0045-7825/© 2014 Elsevier B.V. All rights reserved.

abstraction at which computers may be used, shields programmers from low level details of how a solution is to be obtained on a particular machine, and allows for the solution of problems which would otherwise be too costly or too complex to program. For example, in the preliminary design document for the FORTRAN programming language, Backus [1] writes:

> FORTRAN will comprise a large set of programs to enable the IBM 704 to accept a concise formulation of a problem in terms of a mathematical notation and to produce automatically a high speed 704 program for the solution of the problem … [S]uch a system will make experimental investigation of various mathematical models and numerical methods more feasible and convenient both in human and economic terms.

In the decades since this same motivation of reflecting mathematical structure in code led to the development of other environments in which higher level problems may be expressed, such as the wildly successful MATLAB environment for numerical linear algebra [2].

The FEniCS project [3] aims to develop a software environment for the automated solution of partial differential equations (PDEs) via the finite element method. In particular, it allows the user to specify a variational form representation of the model equations in the Unified Form Language (UFL), [4,5], which closely mimics the mathematical notation in which finite element discretisations may be written. The UFL representation of a problem is automatically compiled by a dedicated form compiler [6] into efficient C++ code, much as FORTRAN code is compiled by a dedicated compiler into efficient machine code.

The UFL abstraction for the spatial discretisation of PDEs has a number of important advantages. As the mathematical structure of the PDE is available for analysis, important equation-specific optimisations may be performed that low level compilers cannot automate, or that would be too laborious to implement by hand [7,8]. UFL is remarkably compact: a finite element discretisation that might take thousands or tens of thousands of lines of FORTRAN or C++ code to implement can be cleanly expressed in just a handful of lines of UFL. For example, even the complicated elliptic relaxation turbulence model [9] can be represented in eleven lines of UFL code (ignoring boundary conditions) [10]. As UFL expresses *what* problem is to be solved, without specifying *how* it is to be solved, the system is free to adapt the implementation to the hardware, including GPUs [11–13]. Lastly, the ability to analyse the mathematical structure of the equations vastly simplifies the task of algorithmic differentiation, and allows for the fully automated derivation of the adjoint model associated with a given forward model [14]; these adjoint models can in turn be used to automatically solve PDE-constrained optimisation problems [15] and conduct generalised stability analyses [16]. Adjoint models will be discussed further in Section 1.2.

However, UFL lacks a native representation for specifying time-dependent problems: in general the user must perform the temporal discretisation by hand and implement it as a sequence of spatial problems. With this manual approach the FEniCS system is unable to automatically perform optimisations that exploit the temporal structure of the discretisation, such as the pre-assembly and caching of terms that occur repeatedly, and the reuse of preconditioners or factorisations in the linear solvers. These optimisations are crucial for efficient implementations of timestepping models, but the user must add them by hand. Some limited support for special types of time dependent problems within the FEniCS system is in development – here we address the general case.

Aside from the unnecessary labour, the absence of temporal abstraction has a major disadvantage: *the implementation of the temporal optimisations breaks the spatial abstraction*. The model can no longer be cleanly expressed as a list of variational problems to be solved; the user must manually pre-assemble certain terms outside of the time loop, assemble other terms inside the time loop, and express the problem at the level of matrices and vectors. Firstly, this loss of structure makes the code significantly harder to read, understand, debug and modify. The expression of the problem and guidance for how it should be solved have been irretrievably interwoven. Secondly, it damages performance portability: for example, on a GPU it may be preferable to recompute terms, rather than cache them, but the user has irrevocably committed to one strategy or the other. Thirdly, it significantly hampers the automated derivation of adjoint models, as the variational structure of the time-dependent problem must be pieced together from the lower-level implementation.

In this paper we introduce an abstraction for expressing time-dependent models, and apply this methodology in combination with the FEniCS system. This temporal abstraction allows for the high level expression of both the spatial and temporal structure of the problem, and resolves all the aforementioned disadvantages.