A Form Compiler based on Symbolic Computations

Martin Sandve Alnæs and Kent-Andre Mardal, Simula Research Laboratory

February 26, 2008

1 Introduction

SyFi is a software package for symbolic finite element computations, which is now part of the FEniCS project. It consists of two main parts: a kernel and a form compiler.

The kernel consists of a collection of tools for symbolic computations on polynomial spaces and polygonal domains, and a collection of elements. The SyFi kernel, written in C++, is roughly comparable to FIAT. which tabulates finite elements using numerical techniques.

The SyFi Form Compiler (SFC) is a Python module comparable to FFC. It takes as input a symbolic description of a variational form and a set of finite elements and generates low level C++ code. The generated code complies with the UFC interface and can be used in (Py)DOLFIN to assemble matrices and vectors. SFC supports Just-In-Time compilation via the package Instant, such that we can define the variational form within Python, then automatically compile generated C++ code into an extension module which is dynamically loaded back into Python.

This combination of symbolic mathematics and code generation enables the specification of finite element discretizations in a user-friendly environment while maintaining efficiency. In addition, the symbolic framework allow us to do certain calculations automatically that we earlier typically did by hand, e.g. symbolic differentiation which simplifies e.g. the implementation of complex material laws for hyper-elastic materials, and the calculation of the Jacobian in the case of a nonlinear PDE.

We will present a short code example using SFC together with PyDOLFIN, before we present some benchmarks showing the efficiency of the generated code for computing the element tensor for a few forms. We will compare with quadrature based examples written in Diffpack and Deal.II, and code generated by FFC. The code generated by SFC is run with both analytic integration and quadrature, and with and without attempts at further optimization.

2 Conclusions

In cases where analytic integration is feasible, it can result in a major speedup in the computation of the element tensor. For elements defined on simplices, the performance of SFC is roughly the same as FFC (without FErari). One point to notice is that if coefficient functions are involved in nonlinear terms in the form, the expressions blow up as the order of their finite element spaces increase and the efficiency gain from analytic integration may be lost. A good factorization algorithm could maybe remedy this problem. Preliminary attempts to optimize the symbolic expressions prior to code generation have been only partially successfull.

Moreover, the code generation process can be quite intensive in terms of memory usage and computing time. The code generation based on quadrature is less demanding, and the resulting code is (as will be demonstrated) much more efficient than the implementations in Diffpack and Deal.II, the reason being that Diffpack and Deal.II provide their abstractions at the C++ level. Thus, both the efficiency gain caused by generating low level code and the advantages of working with a more abstract input format based on symbolic tools are retained independent of integration method chosen.