Efficient Compilation of a Class of Variational Forms

ROBERT C. KIRBY
The University of Chicago
and
ANDERS LOGG
Simula Research Laboratory

We investigate the compilation of general multilinear variational forms over affines simplices and prove a representation theorem for the representation of the element tensor (element stiffness matrix) as the contraction of a constant reference tensor and a geometry tensor that accounts for geometry and variable coefficients. Based on this representation theorem, we design an algorithm for efficient pretabulation of the reference tensor. The new algorithm has been implemented in the FEniCS Form Compiler (FFC) and improves on a previous loop-based implementation by several orders of magnitude, thus shortening compile-times and development cycles for users of FFC.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]—*Algorithm design and analysis, efficiency*; G.1.8 [**Numerical Analysis**]: Partial Differential Equations—*Finite Element Methods*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: granularity, loop hoisting, BLAS, monomials, compiler, variational form, finite element, automation

ACM Reference Format:

Kirby, R. C. and Logg, A. 2007. Efficient compilation of a class of variational forms. ACM Trans. Math. Softw. 33, 3, Article 17 (August 2007), 20 pages. DOI = 10.1145/1268769.1268771 http://doi.acm.org/10.1145/1268769.1268771

1. INTRODUCTION

It is our goal to improve the efficiency of compiling variational forms with FFC, the FEniCS Form Compiler, previously presented in Kirby and Logg [2006]. FFC

This work was supported by the United States Department of Energy under grant DE-FG02-04ER25650.
Authors' addresses: R. C. Kirby, Department of Computer Science, University of Chicago, 1100 East 58th Street, Chicago, IL 60637; email: Kirby@cs.uchicago.edu; A. Logg, Simula Research Laboratory, Martin Linges v 17, Fornebu, PO Box 134, 1325 Lysaker, Norway; email: logg@simula.no.

automatically generates efficient low-level code for evaluating a wide class of multilinear variational forms associated with finite element methods [Ciarlet 1976; Hughes 1987; Brenner and Scott 1994; Eriksson et al. 1996] for partial differential equations. However, the efficiency of FFC decreases rapidly with the complexity of the variational form and polynomial degree. In this paper, we investigate the core algorithms of FFC and rephrase them so as to diminish interpretive overhead and make better use of optimized numerical libraries. We thus wish to decrease the runtime for the form compiler, corresponding to a reduced compiletime for a finite element code. This becomes particularly important when FFC is used as a just-in-time compiler to generate and compile code on the fly at runtime.

1.1 FFC, the FEniCS Form Compiler

FFC [Logg 2006] was first released in 2004 as a prototype compiler for variational forms, automating a key step in the implementation of finite element methods. [Logg 2004]. Given a multilinear variational form and an affinely mapped simplex, FFC automatically generates low-level code for evaluation of the variational form (assembly of the associated linear system). More precisely, FFC generates efficient low-level code for computation of the element tensor (element stiffness matrix), based on the novel approach of representing the element tensor as a special tensor contraction presented earlier in Kirby et al. [2005, SISC] and Kirby et al. [2004]. FFC is implemented in Python and provides both a command-line and, Python interface for the specification of variational forms in a syntax very close to the mathematical notation. Together with other components of the FEniCS project [Hoffman et al. 2006, FEniCS], such as FIAT [Kirby 2004, 2006a, 2006b] and DOLFIN [Hoffman et al. 2006, DOLFIN], FFC automates some central aspects of the finite element method.

There exist today a number of competing efforts that strive to automate the finite element method. One such example is Sundance [Long 2003], which similarly to FFC provides a system for automated assembly/evaluation of variational forms given in mathematical notation. A main difference between Sundance and FFC is that Sundance provides a runtime system for parsing and evaluation of variational forms, whereas FFC precomputes important quantities at compiletime, which in many cases allows for generation of more efficient code for the runtime assembly of linear systems. Automated assembly from a high-level specification of a variational form is also supported by FreeFEM [Pironneau et al. 2006], GetDP [Dular and Geuzaine 2006], and Analysa [Bagheri and Scott 2003], which all implement domain-specific languages for specification and implementation of finite element methods for partial differential equations. Other projects such as Diffpack [Langtangen 1999] and deal.II [Bangerth et al. 2006] provide sophisticated libraries aiding implementation of finite element methods. These libraries provide tools such as meshes, ordering of degrees of freedom, and interfaces to solvers, but do not provide automated evaluation of variational forms.

Since the first release of FFC, a number of improvements have been made, mostly improving on the functionality of the compiler. New features that have been added since Kirby and Logg [2006] include support for mixed finite element formulations, an extension of the form language to include linear algebra operations such as inner products and matrix-vector products, differential operators such as the gradient, divergence and rotation, local projections between finite element spaces and an option to generate code in terms of level 2 BLAS operations. FFC also functions as a just-in-time compiler for PyDOLFIN, the Python interface of DOLFIN [Hoffman et al. 2006, DOLFIN].

However, the performance of FFC has been suboptimal, potentially lengthening development cycles for high-order simulations in three dimensions. Additionally, this inefficiency would inhibit the embedding of FFC in a run-time system, such as Sundance, as a just-in-time compiler.

1.2 Main Results

The main purpose of this paper is twofold. First, we extend and formalize our particular representation of multilinear variational forms. This involves writing each form as a sum of *monomials*, which are integrals of products of (derivatives of) the basis functions. Hence, we prove a representation theorem showing that the evaluation of any monomial form is equivalent to a contraction of a *reference tensor* with a *geometry tensor*. This makes precise what we have discussed in previous work [Kirby et al. 2005; Kirby and Logg 2006]. Second, as evaluating the reference tensor is a dominant cost of FFC, we discuss how to improve the efficiency of building the reference tensor for each monomial by rewriting the computation in terms of operations that may be performed by optimized libraries and by hoisting loop invariants. The loop hoisting is nontrivial since the depth of the loop nesting is not known until the code is executed. The speedups gained with the new algorithm for a series of test cases range between one and three orders of magnitude. Furthermore, we introduce the concept of *signatures* for monomial forms to allow factorization of common monomial terms when evaluating a given multilinear form.

1.3 Outline of This Article

In Section 2, we first derive a representation for the element tensor as a contraction of two tensors, which is at the core of the implementation of FFC. We then, in Section 3, discuss different approaches to precomputation of the monomial integrals that appear in this tensor representation, concluding that a suitable rearrangement of the computation can lead to significant improvement in performance.

To test the new algorithm, we present in Section 4 benchmark results for a series of test cases, comparing the latest version of FFC with a previous version. Finally, we summarize our findings in Section 5.

2. EVALUATION OF MULTILINEAR FORMS

We review here the basic idea of tensor representation of multilinear variational forms, as first presented in Kirby et al. [2005] and Kirby and Logg

[2006], and derive a representation theorem for a general class of multilinear forms.

At the core of finite element methods for partial differential equations is the assembly of a linear system from a given bilinear form. In general, we consider a multilinear form

$$a : V_1 \times V_2 \times \cdots \times V_r \to \mathbb{R}, \tag{1}$$

defined on the product space $V_1 \times V_2 \times \cdots \times V_r$ of a given set $\{V_i\}_{i=1}^r$ of discrete function spaces on a triangulation $\mathcal{T}$ of a domain $\Omega \subset \mathbb{R}^d$.

Typically, $r = 1$ (linear form) or $r = 2$ (bilinear form), but the form compiler FFC can handle multilinear forms of arbitrary *arity* r. In the simplest case, all function spaces are equal but there are many important examples, such as mixed methods, where it is important to consider arguments coming from different function spaces.

2.1 The Element Tensor

Let $\{\phi_i^1\}_{i=1}^{N_1}, \{\phi_i^2\}_{i=1}^{N_2}, \ldots, \{\phi_i^r\}_{i=1}^{N_r}$ be bases of $V_1, V_2, \ldots, V_r$ and let $i = (i_1, i_2, \ldots, i_r)$ be a multiindex. The multilinear form a then defines a rank r tensor, given by

$$A_i = a\big(\phi_{i_1}^1, \phi_{i_2}^2, \ldots, \phi_{i_r}^r\big). \tag{2}$$

In the case of a bilinear form, the tensor A is a matrix (the stiffness matrix), and in the case of a linear form, the tensor A is a vector (the load vector).

As discussed in Kirby and Logg [2006], to compute the tensor A by assembly, we need to compute the *element tensor* A^K on each element K of the triangulation $\mathcal{T}$ of Ω. With $\{\phi_i^{K,1}\}_{i=1}^{n_1}$ the restriction to K of the subset of $\{\phi_i^1\}_{i=1}^{N_1}$ supported on K, $V_1^K = \mathrm{span}\{\phi_i^{K,1}\}_{i=1}^{n_1}$ and the local spaces $V_2^K, \ldots, V_r^K$ defined similarly, we need to evaluate the rank r *element tensor* A^K, given by

$$A_i^K = a_K\big(\phi_{i_1}^{K,1}, \phi_{i_2}^{K,2}, \ldots, \phi_{i_r}^{K,r}\big) \quad \forall i \in \mathcal{I}, \tag{3}$$

where a_K is the local contribution to the given multilinear form a on the element K and where $\mathcal{I}$ is the index set

$$\mathcal{I} = \prod_{j=1}^{r} \big[1, \big|V_j^K\big|\big] = \{(1, 1, \ldots, 1), (1, 1, \ldots, 2), \ldots, (n_1, n_2, \ldots, n_r)\}. \tag{4}$$

We restrict our discussion to multilinear forms that may be written as a sum over terms consisting of integrals over Ω of products of derivatives of functions from sets of discrete spaces $\{V_i\}_{i=1}^r$. We call such terms *monomials*. For one such term, the element tensor takes the following (preliminary) form:

$$A_i^K = \int_K \prod_{j=1}^{r} D_x^{\delta_j} \phi_{\iota_j(i)}^{K,j} \, \mathrm{d}x, \tag{5}$$

where the subscript $\iota_j(i)$ picks out a basis function from the restriction of V_j to K for the current multiindex i and where δ_j is the multiindex for the corresponding derivative. For sequences of multiindices such as $\{\delta_j\}_{j=1}^r$, we use the convention that δ_{jk} denotes the kth element of the jth multiindex for $k = 1, 2, \ldots, |\delta_j|$.

To explain the notation, we consider a couple of illustrative examples. First, consider the bilinear form $a(v, u) = \int_\Omega vu \, \mathrm{d}x$ corresponding to a mass matrix. The element tensor (matrix) is then given by

$$A_i^K = \int_K \phi_{i_1}^{K,1} \phi_{i_2}^{K,2} \, \mathrm{d}x, \tag{6}$$

and so, in the notation of (5), we have $r = 2$, $\iota_j(i) = i_j$ and $\delta_j = \emptyset$ for $j = 1, 2$, where $\emptyset$ denotes an empty multiindex (basis function is not differentiated).

Next, we consider the bilinear form $a(v, u) = \int_\Omega v \frac{\partial^2 u}{\partial x_1 \partial x_2} \, \mathrm{d}x$ with corresponding element tensor given by

$$A_i^K = \int_K \phi_{i_1}^{K,1} \frac{\partial^2 \phi_{i_2}^{K,2}}{\partial x_1 \partial x_2} \, \mathrm{d}x, \tag{7}$$

which we can phrase in the notation of (5) with $r = 2$, $\iota_j(i) = i_j$, $\delta_1 = \emptyset$ and $\delta_2 = (1, 1)$.

More generally, variational problems involve sums over monomial terms, each of which may include a spatially varying coefficient. We express such a coefficient in a finite element basis as $\sum_{\gamma=1}^n c_\gamma \phi_\gamma$. Also, the function spaces involved may each be vector-valued. While we might reduce this situation to a collection of cases of the form (5), we instead extend our canonical form. This allows us to write the element tensor as

$$A_i^K = \sum_{\gamma \in \mathcal{C}} \int_K \prod_{j=1}^m c_j(\gamma) D_x^{\delta_j(\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j} [\kappa_j(\gamma)] \, \mathrm{d}x, \tag{8}$$

with summation over some appropriate index set $\mathcal{C}$, where $\kappa_j(\gamma)$ denotes a component index for factor j depending on γ. To distinguish a component index from a basis function index, we here use $[\cdot]$ to denote a component index. Note that the number of factors m may be different from the rank r of the element tensor (arity of the form).

To illustrate the notation, we consider the bilinear[1] form on $V_1 \times V_2$ for the weighted vector-valued Poisson's equation with given variable coefficient function $w \in V_3$:

$$a(v, u) = \int_\Omega w \nabla v : \nabla u \, \mathrm{d}x = \sum_{\gamma_1=1}^d \sum_{\gamma_2=1}^d \int_\Omega w \frac{\partial v[\gamma_1]}{\partial x_{\gamma_2}} \frac{\partial u[\gamma_1]}{\partial x_{\gamma_2}} \, \mathrm{d}x. \tag{9}$$

The corresponding element tensor is given by

$$A_i^K = \sum_{\gamma_1=1}^d \sum_{\gamma_2=1}^d \sum_{\gamma_3=1}^{|V_3^K|} \int_\Omega \frac{\partial \phi_{i_1}^{K,1}[\gamma_1]}{\partial x_{\gamma_2}} \frac{\partial \phi_{i_2}^{K,2}[\gamma_1]}{\partial x_{\gamma_2}} w_{\gamma_3}^K \phi_{\gamma_3}^{K,3} \, \mathrm{d}x, \tag{10}$$

with $\{w_{\gamma_3}^K\}_{\gamma_3=1}^{|V_3^K|}$ the expansion coefficients for w in the local basis on K for V_3. In the notation of (8), we thus have $r = 2$, $m = 3$, $\iota(i, \gamma) = (i_1, i_2, \gamma_3)$, $\delta(\gamma) = (\gamma_2, \gamma_2, \emptyset)$, $\kappa(\gamma) = (\gamma_1, \gamma_1, \emptyset)$ and $c_j(\gamma) = (1, 1, w_{\gamma_3}^K)$. The index set $\mathcal{C}$ is given by $\mathcal{C} = \{(\gamma_1, \gamma_2, \gamma_3)\} = [1, d]^2 \times [1, |V_3^K|]$.

[1]Note that we may alternatively consider this to be a *trilinear form*, if we think of the coefficient w as a free argument to the form and not as a fixed given function.

One may argue that the canonical form (8) may always be reduced to the simpler form (5) by considering any given element tensor as a suitable transformation (summation and multiplication with coefficients) of basic element tensors of the simple form (5). However, we shall consider the more general form (8) since it increases the granularity of the operation of computing the reference element tensor, which is the operation we set out to optimize. It is also a more flexible representation in that it allows us to directly express the element tensor for a wider range of multilinear forms such as that for the vector-valued Poisson's equation.

2.2 Representing the Element Tensor as a Tensor Contraction

We may write the element tensor for any (affine) element K as a contraction of one tensor depending only on the form and function spaces with another depending only on the geometry and coefficients in the problem. This is accomplished by affinely transforming from K to a reference element K_0. In order to show this, we first state some basic results providing a notational framework for our representation theorem.

We first make the following observation about interchanging product and summation.

LEMMA 1 (INTERCHANGING PRODUCT AND SUMMATION). *With $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_m$ a given sequence of index sets, product and summation may be interchanged as follows:*

$$\prod_{i=1}^{m} \sum_{j \in \mathcal{J}_i} a_{ij} = \sum_{j \in \mathcal{J}_1 \times \cdots \times \mathcal{J}_m} \prod_{i=1}^{m} a_{ij_i} = \sum_{j \in \prod_{k=1}^{m} \mathcal{J}_k} \prod_{i=1}^{m} a_{ij_i}, \tag{11}$$

where each $j \in \prod_{k=1}^{m} \mathcal{J}_k$ is a multiindex of length $|j| = m$.

Using the notation of Lemma 1, we may also prove the following chain rule for higher order partial derivatives.

LEMMA 2 (CHAIN RULE). *If $F : \mathbb{R}^d \to \mathbb{R}^d$ is a bijective and differentiable mapping (a diffeomorphism) between two coordinate systems, $x = F(X)$, then*

$$D_x^{\delta} = \sum_{\delta' \in [1,d]^{|\delta|}} \left(\prod_{k=1}^{|\delta|} \frac{\partial X_{\delta'_k}}{\partial x_{\delta_k}} \right) D_X^{\delta'}, \tag{12}$$

for each multiindex δ, where $D_x^{\delta} = \prod_{i=1}^{|\delta|} \frac{\partial}{\partial x_{\delta_i}}$ and $D_X^{\delta'} = \prod_{i=1}^{|\delta'|} \frac{\partial}{\partial X_{\delta'_i}}$.

PROOF. By the standard chain rule and Lemma 1, we have

$$\begin{aligned} D_x^{\delta} &= \prod_{i=1}^{|\delta|} \frac{\partial}{\partial x_{\delta_i}} = \prod_{i=1}^{|\delta|} \sum_{\delta'=1}^{d} \frac{\partial X_{\delta'}}{\partial x_{\delta_i}} \frac{\partial}{\partial X_{\delta'}} = \sum_{\delta' \in [1,d]^{|\delta|}} \prod_{i=1}^{|\delta|} \frac{\partial X_{\delta'_i}}{\partial x_{\delta_i}} \frac{\partial}{\partial X_{\delta'_i}} \\ &= \sum_{\delta' \in [1,d]^{|\delta|}} \prod_{k=1}^{|\delta|} \frac{\partial X_{\delta'_k}}{\partial x_{\delta_k}} \prod_{i=1}^{|\delta|} \frac{\partial}{\partial X_{\delta'_i}} = \sum_{\delta' \in [1,d]^{|\delta|}} \left(\prod_{k=1}^{|\delta|} \frac{\partial X_{\delta'_k}}{\partial x_{\delta_k}} \right) D_X^{\delta'}. \end{aligned} \tag{13}$$

□

We may now prove the following representation theorem[2] for the element tensor.

THEOREM 1 (REPRESENTATION THEOREM). *If F_K is a given affine mapping from a reference element K_0 to an element K and $\{V_j^K\}_{j=1}^m$ is a given set of discrete function spaces on K, each generated by a discrete function space on the reference element through the affine mapping, that is, for each $\phi \in V_j^K$ there is some $\Phi \in V_j^0$ such that $\Phi = \phi \circ F_K$, then the element tensor* (8) *may be represented as the tensor contraction of a* reference tensor A^0 *and a* geometry tensor G_K,

$$A^K = A^0 : G_K, \tag{14}$$

that is,

$$A_i^K = \sum_{\alpha \in \mathcal{A}} A_{i\alpha}^0 G_K^\alpha \quad \forall i \in \mathcal{I}, \tag{15}$$

where the reference tensor A^0 is independent of K. In particular, the reference tensor A^0 is given by

$$A_{i\alpha}^0 = \sum_{\beta \in \mathcal{B}} \int_{K_0} \prod_{j=1}^m D_X^{\delta_j'(\alpha,\beta)} \Phi_{\iota_j(i,\alpha,\beta)}^j [\kappa_j(\alpha, \beta)] \, \mathrm{d}X, \tag{16}$$

and the geometry tensor G_K is the outer product of the coefficients of any weight functions with a tensor that depends only on the Jacobian F_K',

$$G_K^\alpha = \prod_{j=1}^m c_j(\alpha) \det F_K' \sum_{\beta \in \mathcal{B}'} \prod_{j'=1}^m \prod_{k=1}^{|\delta_{j'}(\alpha,\beta)|} \frac{\partial X_{\delta_{j'k}'(\alpha,\beta)}}{\partial x_{\delta_{j'k}(\alpha,\beta)}}, \tag{17}$$

for some appropriate index sets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{B}'$. We refer the the index set $\mathcal{I}$ as the set of primary indices, *the index set $\mathcal{A}$ as the set of* secondary indices, *and to the index sets $\mathcal{B}$ and $\mathcal{B}'$ as* auxiliary indices.

PROOF. Starting from (8), we may move the product of constant expansion coefficients outside of the integral to obtain

$$\begin{aligned} A_i^K &= \sum_{\gamma \in \mathcal{C}} \int_K \prod_{j=1}^m c_j(\gamma) D_x^{\delta_j(\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j} [\kappa_j(\gamma)] \, \mathrm{d}x \\ &= \sum_{\gamma \in \mathcal{C}} \prod_{j'=1}^m c_{j'}(\gamma) \int_K \prod_{j=1}^m D_x^{\delta_j(\gamma)} \phi_{\iota_j(i,\gamma)}^{K,j} [\kappa_j(\gamma)] \, \mathrm{d}x. \end{aligned} \tag{18}$$

We now make a change of variables through F_K, mapping coordinates $X \in K_0$ to coordinates $x = F_K(X) \in K$, to carry out the integration on the reference element K_0. By Lemma 2, we thus obtain

[2] A similar representation was derived and presented in Kirby and Logg [2006] but in less formal notation.

$$\begin{aligned} A_i^K &= \sum_{\gamma\in\mathcal{C}} \prod_{j'=1}^m c_{j'}(\gamma) \int_{K_0} \prod_{j=1}^m \sum_{\delta'\in[1,d]^{|\delta_j|}} \prod_{k=1}^{|\delta_j|} \frac{\partial X_{\delta'_k}}{\partial x_{\delta_{jk}}} \\ &\quad \times D_X^{\delta'} \Phi^j_{\iota_j(i,\gamma)}[\kappa_j(\gamma)] \ \det F'_K \,\mathrm{d}X \\ &= \sum_{\gamma\in\mathcal{C}} \sum_{\delta'\in\prod_{l=1}^m[1,d]^{|\delta_l|}} \prod_{j'=1}^m c_{j'}(\gamma) \int_{K_0} \prod_{j=1}^m \prod_{k=1}^{|\delta_j|} \frac{\partial X_{\delta'_{jk}}}{\partial x_{\delta_{jk}}} \\ &\quad \times D_X^{\delta'_j} \Phi^j_{\iota_j(i,\gamma)}[\kappa_j(\gamma)] \ \det F'_K \,\mathrm{d}X, \end{aligned} \tag{19}$$

where we have also used Lemma 1 to change the order of multiplication and summation. Now, since the mapping F_K is affine, the transforms $\frac{\partial X}{\partial x}$ and the determinant are constant and may thus be pulled out of the integral. As a consequence, we obtain

$$\begin{aligned} A_i^K &= \sum_{\gamma\in\mathcal{C}} \sum_{\delta'\in\prod_{l=1}^m[1,d]^{|\delta_l|}} \prod_{j'=1}^m c_{j'}(\gamma) \det F'_K \prod_{j''=1}^m \prod_{k=1}^{|\delta_{j''}|} \frac{\partial X_{\delta'_{j''k}}}{\partial x_{\delta_{j''k}}} \\ &\quad \times \int_{K_0} \prod_{j=1}^m D_X^{\delta'_j} \Phi^j_{\iota_j(i,\gamma)}[\kappa_j(\gamma)] \,\mathrm{d}X. \end{aligned} \tag{20}$$

The summation over $\mathcal{C}$ and $\prod_{l=1}^m[1,d]^{|\delta_l|}$ may now be rearranged as a summation over an index set $\mathcal{B}$ local to the terms of the integrand, a summation over an index set $\mathcal{B}'$ local to the terms outside of the integral, and a summation over an index set $\mathcal{A}$ common to all terms. We may thus express the element tensor A^K as the tensor contraction

$$A_i^K = \sum_{\alpha\in\mathcal{A}} A^0_{i\alpha} G_K^\alpha, \tag{21}$$

where

$$\begin{aligned} A^0_{i\alpha} &= \sum_{\beta\in\mathcal{B}} \int_{K_0} \prod_{j=1}^m D_X^{\delta'_j(\alpha,\beta)} \Phi^j_{\iota_j(i,\alpha,\beta)}[\kappa_j(\alpha,\beta)] \,\mathrm{d}X, \\ G_K^\alpha &= \prod_{j=1}^m c_j(\alpha) \ \det F'_K \sum_{\beta\in\mathcal{B}'} \prod_{j'=1}^m \prod_{k=1}^{|\delta_{j'}(\alpha,\beta)|} \frac{\partial X_{\delta'_{j'k}(\alpha,\beta)}}{\partial x_{\delta_{j'k}(\alpha,\beta)}}. \end{aligned} \tag{22}$$

Note that since each coefficient $c_j(\alpha)$ in the geometry tensor G_K is always paired with a corresponding basis function in the reference tensor A^0, we were able to reorder the summation to move the coefficients outside of the summation over $\mathcal{B}'$. □

As demonstrated earlier in Kirby and Logg [2006], the representation (14) in combination with precomputation of the reference tensor A^0 may lead to very efficient computation of the element tensor A^K, with typical runtime speedups ranging between a factor 10 and a factor 1000 compared to standard runtime evaluation of the element tensor by numerical quadrature. The speedup is a direct result of the reduced operation count for the computation of the element tensor based on the tensor representation. In addition, one may omit multiplication with zeros and detect symmetries or other dependencies between the entries of the element tensor to further reduce the operation count as discussed in Kirby et al. [2005] and Kirby et al. [2006].

The rank of the reference tensor is determined both by the arity of the multilinear form and how the form is expressed as a product of coefficients and derivatives of basis functions. In general, the rank of the reference tensor is $|i| + |\alpha|$, where $|i| = r$ is the arity of the form and $|\alpha|$ is the rank of the geometry tensor. As a rule of thumb, the rank of the geometry tensor is the sum of the number of coefficients n_C and the number of derivatives n_D appearing in the definition of the form, and thus the rank of the reference tensor for a bilinear form is $2 + n_C + n_D$. Examples are given below in Section 4 for a set of test cases.

2.3 Runtime Evaluation of the Tensor Contraction

This framework also maps onto using matrix-vector or matrix-matrix products at run-time. We may recast the tensor contraction (14) as a matrix-vector product for each K in the mesh. This involves first casting A^0 as a matrix by labeling all the items of the index set $\mathcal{I}$ with integers in $[1, |\mathcal{I}|]$ and the index set $\mathcal{A}$ with integers in $[1, |\mathcal{A}|]$. This ordering of the multiindices $i \in \mathcal{I}$ corresponds to the rows of the matrix and the ordering of the multiindices $\alpha \in \mathcal{A}$ corresponds to the columns. The same ordering is imposed on G_K to make it a vector. Furthermore, we may take a batch of elements $\mathcal{T}' \subset \mathcal{T}$ and compute $\{A^K\}_{K \in \mathcal{T}'}$ with a matrix-matrix product. Currently, FFC supports code generation that sets up the matrix-vector products via level 2 BLAS, and computing batches of elements with matrix-matrix products via level 3 BLAS will be supported in a future version.

2.4 Equivalence of Reference Tensors

As noted earlier, forming the reference tensor is a dominant part of the cost for FFC when compiling code for the evaluation of multilinear forms. Before we proceed to discuss algorithms for efficiently evaluating the tensor for a monomial term in the next section, we conclude the discussion of form representation by noting that particular monomial terms have the same reference tensor but different geometry tensors. In such cases, the total cost may thus be reduced by recognizing the common reference tensor and only computing it once.

As an example, consider each term of the two-dimensional Laplacian,

$$A_i^{K,j} = \int_K \frac{\partial \phi_{i_1}^{K,1}}{\partial x_j} \frac{\partial \phi_{i_2}^{K,2}}{\partial x_j} \, \mathrm{d}x, \tag{23}$$

where $j = 1, 2$ is the coordinate direction. By a suitable definition of the index set $\mathcal{C}$, the sum of both terms may be phrased as a single canonical form (8). We may also consider the two terms separately and write each term in the canonical form (8). After changing coordinates to the reference domain, we obtain the reference tensors

$$A_{i\alpha}^{0,j} = \int_{K_0} \frac{\partial \Phi_{i_1}^1}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^2}{\partial X_{\alpha_2}} \, \mathrm{d}X, \quad j = 1, 2, \tag{24}$$

and geometry tensors

$$G_{K,j}^{\alpha} = \det F_K' \frac{\partial X_{\alpha_1}}{\partial x_j} \frac{\partial X_{\alpha_2}}{\partial x_j}, \quad j = 1, 2. \tag{25}$$

Note that the two terms of the form indeed have the same reference tensor but different geometry tensors. This has both compile-time and run-time implications. At compile-time, FFC should recognize this structure, hence building the reference tensor only once and generating code for a single geometry tensor that sums the basic parts. When the generated code is executed at run-time, this corresponds to fewer instructions and hence better performance. We may formalize this as follows.

THEOREM 1. *Consider two multilinear variational forms with corresponding element tensors A_i^K and B_i^K of the form* (5) *defined over spaces $\{V_j^{A,K}\}_{j=1}^{r_A}$ and $\{V_j^{B,K}\}_{j=1}^{r_B}$, respectively, that is,*

$$A_i^K = \int_K \prod_{j=1}^{r_A} D_x^{\delta_j^A} \phi_{\iota_j^A(i)}^{A,K,j} \, \mathrm{d}x, \tag{26}$$

$$B_i^K = \int_K \prod_{j=1}^{r_B} D_x^{\delta_j^B} \phi_{\iota_j^B(i)}^{B,K,j} \, \mathrm{d}x. \tag{27}$$

Suppose that $r_A = r_B \equiv r$, $V_j^A = V_j^B$, $\iota_j^A = \iota_j^B$ and $|\delta_j^A| = |\delta_j^B|$ for $j = 1, 2, \ldots, r$. Then, the corresponding reference tensors are equal, that is, $A^0 = B^0$. Moreover, this relationship is an equivalence relation on the set of variational forms.

We remark that this result can be generalized slightly. If permuting the integrands of one form, say B, would lead to the hypotheses of this theorem being satisfied, then A^0 and B^0 are the same after a similar permutation of the axes of B^0.

FFC recognizes and factors out common reference tensors by computing for each term of a given multilinear form a string that uniquely identifies the term. This may be accomplished by simply concatenating the names of the finite elements that generate the function spaces for the basis functions in the term, together with derivatives and component indices. We refer to such a string as a (hard) *signature* and note that the signature may be computed cheaply for each term by just looking at its canonical representation. We may then factor out common reference tensors by checking for equality of signatures. If two terms have the same signature, they also have a common reference tensor that may be factored out.

FFC also computes a *soft signature* for each term, which is similar to a *hard signature* but disregarding ordering of multiindices. By checking for equality

Table I. Hard Signature (top) and Soft Signature (bottom) for the Reference Tensor of the Bilinear Form $a(v, u) = \int_\Omega \nabla v \cdot \nabla\, dx$ with Piecewise Linear Elements on Triangles (Note that there are no line breaks in the signatures.)

```
1.000000000000000e+00*
{Lagrange finite element of degree 1 on a triangle;i0;[];[(d/dXa0)]}*
{Lagrange finite element of degree 1 on a triangle;i1;[];[(d/dXa1)]}*dX

1.000000000000000e+00*
{Lagrange finite element of degree 1 on a triangle;i0;[];[(d/dXa)]}*
{Lagrange finite element of degree 1 on a triangle;i1;[];[(d/dXa)]}*dX
```

of soft signatures, it is possible to find terms which have reference tensors that only differ by the ordering of their axes. If two soft signatures match but the corresponding hard signatures differ, it is possible to find a reordering that results in equal hard signatures. FFC thus computes for each term a soft signature and if the soft signatures match for two terms, a suitable reordering is found, and the reference tensor may be factored out as before. In Table I, we include the hard and soft signatures for the bilinear form for Poisson's equation, $a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, dx$.

3. COMPUTING THE REFERENCE TENSOR

Given a multilinear variational form, the form compiler FFC automatically generates the canonical form (8) and the representation (14). The computationally most expensive part of this process is the computation of the reference tensor A^0, that is, the tabulation of each integral

$$A^0_{i\alpha} = \sum_{\beta \in \mathcal{B}} \int_{K_0} \prod_{j=1}^{m} D_X^{\delta'_j(\alpha,\beta)} \Phi^j_{\iota_j(i,\alpha,\beta)}[\kappa_j(\alpha, \beta)] \, dX, \tag{28}$$

for $i \in \mathcal{I}$ and $\alpha \in \mathcal{A}$.

As an example, consider the bilinear form

$$a(v, u) = \int_\Omega v \cdot (w \cdot \nabla) u \, dx, \tag{29}$$

appearing in a linearization of the incompressible Navier–Stokes equations (see Section 4 below). Computing the $12 \times 12 \times 12 \times 3 \times 3 = 15,552$ entries of the rank five reference tensor A^0 for piecewise linear elements on tetrahedra takes about 9.5 seconds on a 3.0GHz Pentium 4 with FFC version 0.2.2. Since this computation only needs to be done once at compiletime, one may argue that this is no big issue. However, limitations on computer resources can be a limit to the complexity of the forms we can compile and the degree of polynomials we can use. Furthermore, long turn-around times to compile new, complex models diminish the usefulness of FFC as a tool for truly rapid development.

We present below two very different ways to compute the reference tensor, first the obvious naive approach used in FFC version 0.2.2 and earlier versions, and then a more efficient algorithm used in FFC version 0.2.5 and beyond, which cuts the cost of computing the reference tensor by several orders of magnitude. In the case of the form (29) for piecewise linear elements on tetrahedra, the cost of computing the reference tensor is reduced from 9.5 seconds to around 0.02 seconds.

3.1 Iterating over the Entries of the Reference Tensor

The obvious way to compute the reference tensor is to iterate over all indices of the reference tensor and compute each entry by quadrature over a suitable set of quadrature points $\{X_k\}_{k=1}^{N_q}$ and a corresponding set of quadrature weights $\{w_k\}_{k=1}^{N_q}$ on the reference element K_0, as outlined in Algorithm 1. Note that the iteration over multiindices α and β are themselves multiply nested loops, however the length of α, β and hence the depth of the loop nest depends on the form being compiled. FFC uses the "collapsed-coordinate" Gauss-Jacobi rules described in Karniadakis and Sherwin [1999] based on taking tensor products of Gaussian integration rules over the square and cube and mapping them to the reference simplex. These are the arbitrary-order rules provided by FIAT. Since we are integrating polynomials, we may pick a quadrature rule which is exact for the total polynomial degree of the integrand. Alternatively, we can pick an approximate rule that is sufficiently accurate as per the theory of variational crimes [Ciarlet 1976; Brenner and Scott 1994].

Algoritm 1 is expressed at a very low granularity—a loop over quadrature points for each entry of the reference tensor. The interpretive overhead associated with this algorithm explains the poor performance of earlier versions of FFC in a language such as Python. However, we may express the computation at a much higher level of granularity and leverage optimized libraries written in C, such as Python `Numeric` [Oliphant et al. 2006]. In fact, we wind up with a loop over auxiliary indices and quadrature points, inside which the entire reference tensor is updated by an extended outer product. This higher abstraction dramatically improves performance while allowing us to remain in a high-level language.

Algorithm 1. $A^0 = \text{ComputeReferenceTensor}()$

for $i \in \mathcal{I}$
    for $\alpha \in \mathcal{A}$
        $I = 0$
        for $\beta \in \mathcal{B}$
            for $k = 1, 2, \ldots, N_q$
                $I = I + w_k \prod_{j=1}^m D_X^{\delta'_j(\alpha,\beta)} \Phi^j_{\iota_j(i,\alpha,\beta)}[\kappa_j(\alpha, \beta)](X_k)$
            end for
        end for
        $A^0_{i\alpha} = I$
    end for
end for

Algorithm 2. A^0 = AssembleReferenceTensor()

for $j = 1, 2, \ldots, m$
  $\Psi^j = \text{Tabulate}(V_j^0, \{X_k\}_{k=1}^{N_q}, \mathcal{I}, \mathcal{A}, \mathcal{B}, \iota_j, \delta'_j, \kappa_j)$
end for
$A^0 = 0$
for $k = 1, 2, \ldots, N_q$
  for $\beta \in \mathcal{B}$
    $A^0 = A^0 + \text{ComputeProduct}(\{\Psi^j\}_{j=1}^m, k, \beta)$
  end for
end for

Algorithm 3. $B = \text{ComputeProduct}(\{\Psi^j\}_{j=1}^m, k, \beta)$

$B = w_k$
for $j = 1, 2, \ldots, m$
  $B = B \otimes \Psi_{k\beta}^j$ (*outer product*)
end for

3.2 Assembling the Reference Tensor

Algorithm 1 may be reorganized to significantly improve the performance. By first tabulating the basis functions at all quadrature points according to

$$\Psi_{k\beta,i\alpha}^j = D_X^{\delta'_j(\alpha,\beta)} \Phi_{\iota_j(i,\alpha,\beta)}^j [\kappa_j(\alpha,\beta)](X_k), \tag{30}$$

which may be done efficiently using FIAT, we may improve the granularity of the computation by iterating over quadrature points $\{X_k\}_{k=1}^{N_q}$ and auxiliary indices $\mathcal{B}$, assembling the contributions to the reference tensor from each pair (x_k, β), as outlined in Algorithm 2 and Algorithm 3.

The higher level of abstraction of Algorithm 2 allows us to simultaneously reduce the interpretive overhead and make use of optimized libraries, such as the Python `Numeric` extension module. The accumulation of the outer products may be accomplished with the `Numeric.add` function, which is implemented in terms of efficient C loops over the low-level arrays. Moreover, the sequence of outer products is accumulated through calls to the function `Numeric.multiply.outer`. A sketch of the Python code corresponding to Algorithm 2 and Algorithm 3 is included in Table II. The full code can be downloaded from the FFC Web page [Logg 2006].

The situation is similar to that of the assembly of a global sparse matrix for a variational form; by separating the concerns of computing the local contribution (the element tensor) from the insertion of the local contribution into the global sparse matrix, we may optimize the two steps independently. In the former case, we call the optimized code generated by FFC to compute the element tensor and in the latter case, we may use an optimized library call such as the PETSc [Balay et al. 2006; Balay et al. 2004; Balay et al. 1997] call `MatSetValues()`.

Table II. A Sketch of the Python Implementation of Algorithm 2 and Algorithm 3 in FFC

```
# Iterate over quadrature points
for k in range(num_points):
   # Iterate over secondary indices
   for beta in B:
      # Compute cumulative outer product
      P = w[k]
      for j in range(m):
            P = Numeric.multiply.outer(P, Psi[...])
      # Add to reference tensor
      Numeric.add(A0, P, A0)
```

Moreover, a closer investigation of Algorithm 1 also reveals a source of redundant computation. As one entry in the multiindex α changes, most of the factors of the product in the innermost loop remain the same. This problem grows worse as the arity of the form and the number of derivatives increase. Although this is logically equivalent to a multiply nested loop, the structure of looping over an enumeration of multiindices makes it highly unlikely that an optimizing compiler would hoist invariants. However, Algorithm 3 includes the hoisting out of the arbitrary-depth loop nest.

4. BENCHMARK RESULTS

To measure the efficiency of the proposed Algorithm 2, we compute the reference tensor for a series of test cases, comparing FFC version 0.2.2, which is based on Algorithm 1, with FFC version 0.2.5, which is based on Algorithm 2.

The benchmarks were obtained on an Intel Pentium 4 (3.0 GHz CPU, 2GB RAM) running Debian GNU/Linux with Python 2.4, Python Numeric 24.2-1 and FIAT 0.2.3. The numbers reported are the CPU times/second for the precomputation of the reference tensor, which was previously the main bottleneck in the compilation of a form. With the new and more efficient precomputation of the reference tensor in FFC, the precomputation is no longer a bottleneck and is in some cases dominated by the cost of code generation.

4.1 Test Cases

We take as test cases the computation of the reference tensor for the set of bilinear forms used to benchmark the runtime performance of the code generated by FFC in Kirby and Logg [2006]. For convenience, we choose a common discrete function space $V_1 = V_2 = \cdots = V_n = V$ for all basis functions, but there is no such limitation in FFC; function spaces can be mixed freely.

We also add a fifth test case which is a more demanding problem posing real difficulties for earlier versions of FFC based on Algorithm 1.

Test Case 1: *The Mass Matrix.* As a first test case, we consider the computation of the mass matrix M with $M_{i_1 i_2} = a(\phi^1_{i_1}, \phi^2_{i_2})$ and the bilinear form a given by

$$a(v, u) = \int_\Omega vu \, \mathrm{d}x. \tag{31}$$

The corresponding rank two reference tensor takes the form

$$A_i^0 = \int_{K_0} \Phi_{i_1} \Phi_{i_2} \, \mathrm{d}X, \tag{32}$$

with the rank zero geometry tensor given by $G_K = \det F_K'$.

Test Case 2: *Poisson's Equation.* As a second example, consider the bilinear form for Poisson's equation,

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x. \tag{33}$$

The corresponding rank four reference tensor takes the form

$$A_{i\alpha}^0 = \int_{K_0} \frac{\partial \Phi_{i_1}}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}}{\partial X_{\alpha_2}} \, \mathrm{d}X, \tag{34}$$

with the rank two geometry tensor given by $G_K^\alpha = \det F_K' \sum_{\beta=1}^d \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}$.

Test Case 3 : *Navier-Stokes.* We consider next the nonlinear term $u \cdot \nabla u$ of the incompressible Navier-Stokes equations,

$$\begin{aligned} \dot{u} + u \cdot \nabla u - \nu \Delta u + \nabla p &= f, \\ \nabla \cdot u &= 0. \end{aligned} \tag{35}$$

Linearizing this term as part of either a Newton or fixed-point based solution method (see for example [Eriksson et al. 2003; Hoffman and Johnson 2004]),we need to evaluate the bilinear form

$$a(v, u) = a_w(v, u) = \int_\Omega v \cdot (w \cdot \nabla) u \, \mathrm{d}x. \tag{36}$$

The corresponding rank five reference tensor takes the form

$$A_{i\alpha}^0 = \sum_{\beta=1}^d \int_{K_0} \Phi_{i_1}[\beta] \Phi_{\alpha_1}[\alpha_2] \frac{\partial \Phi_{i_2}[\beta]}{\partial X_{\alpha_3}} \, \mathrm{d}X, \tag{37}$$

with the rank three geometry tensor given by $G_K^\alpha = \det F_K' w_{\alpha_1}^K \frac{\partial X_{\alpha_3}}{\partial x_{\alpha_2}}$.

Test Case 4: *Linear Elasticity.* As our next test case, we consider the strain-strain term of linear elasticity [Brenner and Scott 1994],

$$\begin{aligned} a(v, u) &= \int_\Omega \frac{1}{4} (\nabla v + (\nabla v)^\top) : (\nabla u + (\nabla u)^\top) \, \mathrm{d}x \\ &= \int_\Omega \sum_{i,j=1}^d \frac{1}{4} \frac{\partial v_i}{\partial x_j} \frac{\partial u_i}{\partial x_j} + \frac{1}{4} \frac{\partial v_i}{\partial x_j} \frac{\partial u_j}{\partial x_i} + \frac{1}{4} \frac{\partial v_j}{\partial x_i} \frac{\partial u_i}{\partial x_j} + \frac{1}{4} \frac{\partial v_j}{\partial x_i} \frac{\partial u_j}{\partial x_i} \, \mathrm{d}x \qquad (38) \\ &= \int_\Omega \sum_{i,j=1}^d \frac{1}{2} \frac{\partial v_i}{\partial x_j} \frac{\partial u_i}{\partial x_j} + \frac{1}{2} \frac{\partial v_i}{\partial x_j} \frac{\partial u_j}{\partial x_i} \, \mathrm{d}x. \end{aligned}$$

Table III. Speedups for Test Cases 1–5 in 2D and 3D for Different Polynomial Degrees q of Lagrange Basis Functions

Test Case	$q=1$	$q=2$	$q=3$	$q=4$	$q=5$	$q=6$	$q=7$	$q=8$
1. Mass matrix 2D	1.4	2.6	4.0	5.6	7.6	9.9	12.5	15.2
1. Mass matrix 3D	1.6	3.5	6.4	10.8	16.9	23.1	28.3	20.9
2. Poisson 2D	2.5	7.0	11.4	16.4	21.9	27.5	33.5	39.4
2. Poisson 3D	7.4	19.3	33.8	47.8	43.8	38.8	28.1	23.1
3. Navier–Stokes 2D	67.2	264.3	239.0	—	—	—	—	—
3. Navier–Stokes 3D	461.3	291.7	82.3	—	—	—	—	—
4. Elasticity 2D	20.2	44.3	68.9	—	—	—	—	—
4. Elasticity 3D	142.5	230.7	138.0	—	—	—	—	—
5. Stabilization 2D	1114.7	—	—	—	—	—	—	—
5. Stabilization 3D	1101.4	—	—	—	—	—	—	—

Considering here for simplicity only the first of the two terms[3], the rank four reference tensor takes the form

$$A^0_{i\alpha} = \sum_{\beta=1}^{d} \int_{K_0} \frac{\partial \Phi_{i_1}[\beta]}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}[\beta]}{\partial X_{\alpha_2}} \, \mathrm{d}X, \tag{39}$$

with the rank two geometry tensor given by $G_K^\alpha = \frac{1}{2} \det F_K' \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}$.

Test Case 5 : *Stabilization.* As a final test case, we consider the bilinear form for a stabilization term appearing in a least-squares stabilized cG(1)cG(1) method for the incompressible Navier-Stokes equations [Eriksson et al. 2003; Hoffman and Johnson 2004]:

$$a(v,u) = \int_\Omega (w \cdot \nabla v) \cdot (w \cdot \nabla u) \, \mathrm{d}x = \sum_{i,j,k=1}^{d} w[j] \frac{\partial v[i]}{\partial x_j} w[k] \frac{\partial u[i]}{\partial x_k} \, \mathrm{d}x. \tag{40}$$

The corresponding rank eight reference tensor takes the form

$$A^0_{i\alpha} = \sum_{\beta=1}^{d} \int_{K_0} \Phi_{\alpha_1}[\alpha_3] \frac{\partial \Phi_{i_1}[\beta]}{\partial X_{\alpha_5}} \Phi_{\alpha_2}[\alpha_4] \frac{\partial \Phi_{i_2}[\beta]}{\partial X_{\alpha_6}} \, \mathrm{d}X, \tag{41}$$

with the rank six geometry tensor given by $G_K^\alpha = w^K_{\alpha_1} w^K_{\alpha_2} \det F_K' \frac{\partial X_{\alpha_5}}{\partial x_{\alpha_3}} \frac{\partial X_{\alpha_6}}{\partial x_{\alpha_4}}$. As a consequence of the high rank of the reference tensor, the computation of the reference tensor is very costly. For piecewise linear basis functions on tetrahedra with $4 \times 3 = 12$ basis functions on the reference element, the number of entries in the reference tensor is $12 \times 12 \times 12 \times 12 \times 3 \times 3 \times 3 \times 3 = 1,679,616$.

4.2 Results

In Table III, we present a summary of the speedups obtained with Algorithm 2 (FFC version 0.2.5) compared to Algorithm 1 (FFC version 0.2.2). Detailed results are given in Figures 1–4 for test cases 1–4. Because of limitations in the earlier version of FFC, that is, the poor performance of Algorithm 1, the comparison is made for polynomial degree $q \le 8$ in test cases 1–2, $q \le 3$ in test

[3]The benchmark measures the time to compute the reference tensors for both terms.

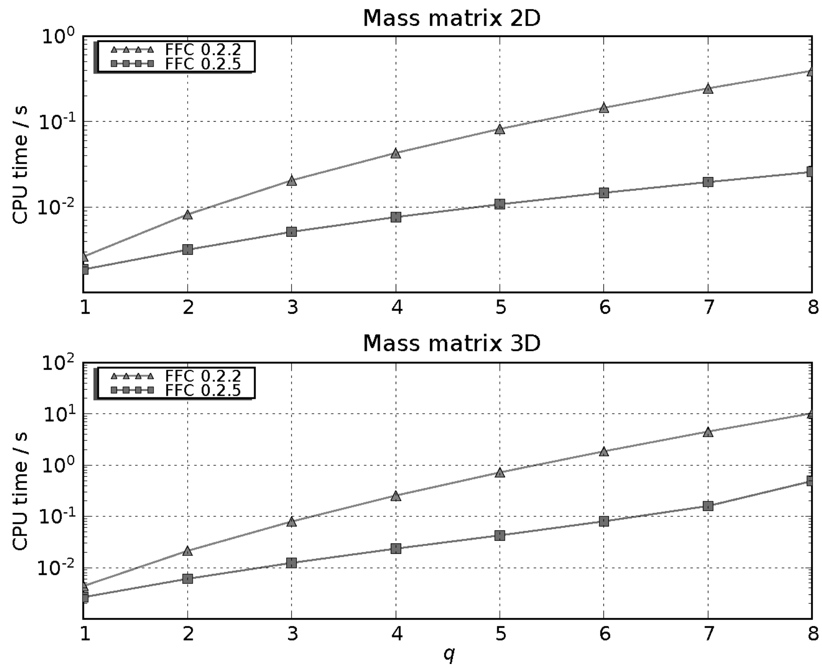

Fig. 1. Compilation time as function of polynomial degree q for test case 1, the mass matrix, specified in FFC by `a = v*u*dx`.

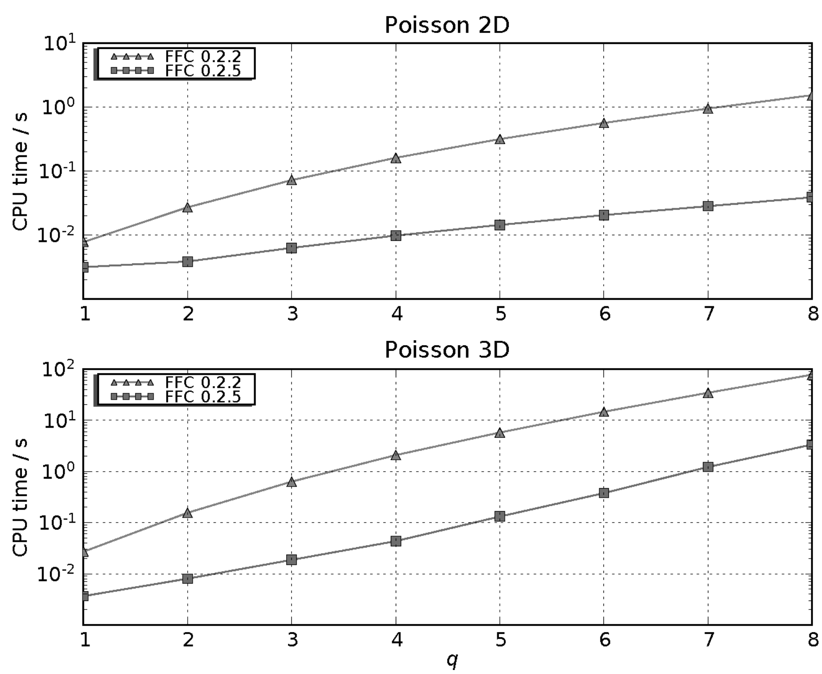

Fig. 2. Compilation time as function of polynomial degree q for test case 2, Poisson's equation, specified in FFC by `a = v.dx(i)*u.dx(i)*dx`.

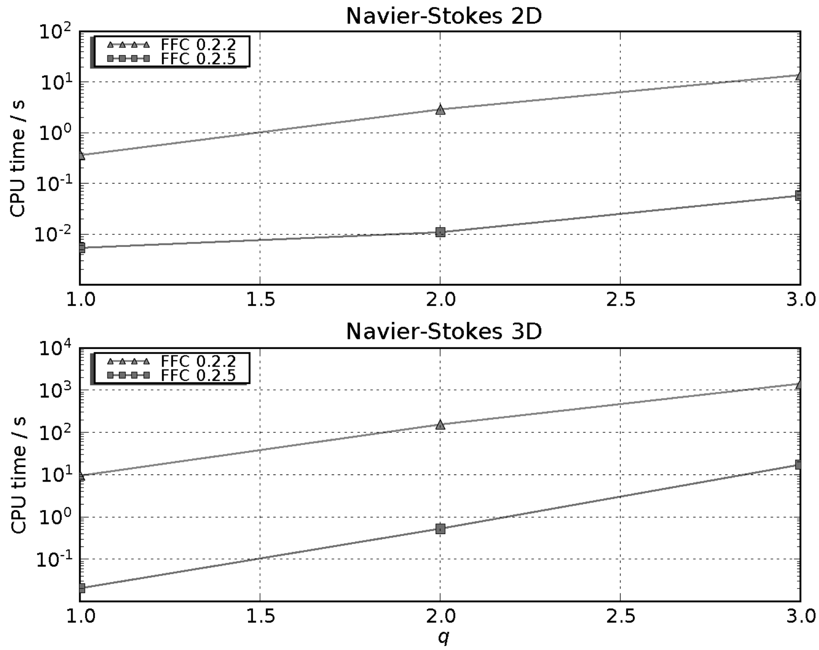

Fig. 3. Compilation time as function of polynomial degree q for test case 3, the nonlinear term of the incompressible Navier-Stokes equations, specified in FFC by `a = v[i]*w[j]*u[i].dx(j)*dx`.

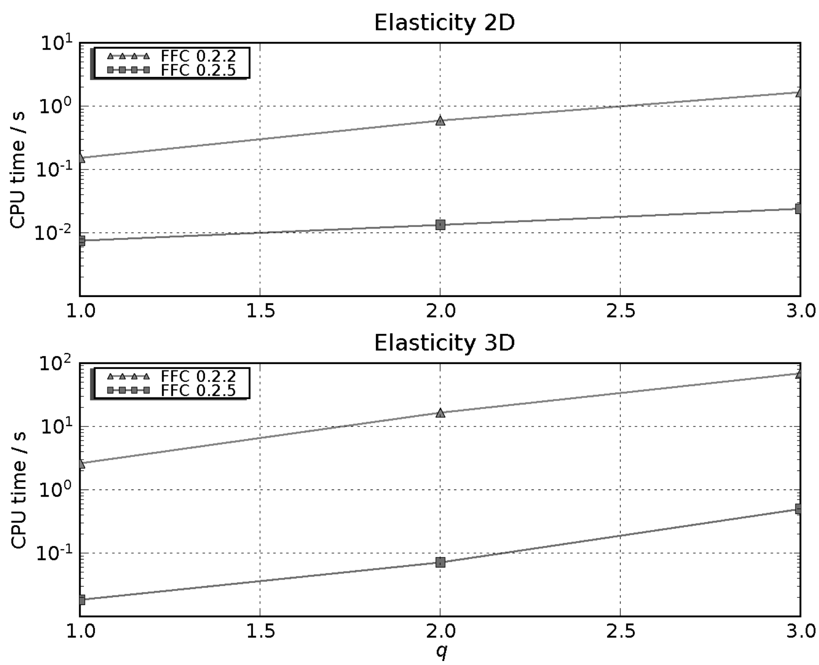

Fig. 4. Compilation time as function of polynomial degree q for test case 4, the strain-strain term of linear elasticity, specified in FFC by `a = 0.25*(v[i].dx(j) + v[j].dx(i)) * (u[i].dx(j) + u[j].dx(i)) * dx`.

cases 3–4 and $q = 1$ in test case 5. Higher degree forms may be compiled with FFC version 0.2.5, but even then the memory requirements for storing the reference tensor may in some cases exceed the available 2GB on the test system.

As evident from Table III, the speedup is significant in most test cases, typically one or two orders of magnitude. In test case 5, the stabilization term in Navier-Stokes, the speedup is as large as three orders of magnitude.

5. CONCLUSION

The new improved precomputation of the reference tensor removes an outstanding bottleneck in the compilation of variational forms. This improves the possibilities of using FFC as a tool for rapid prototyping and development. The feature set for FFC is also quickly expanding, with an expanded form language, recently added support for arbitrary mixed formulations, and with built-in support for functionals, nonlinear formulations and error estimates on the horizon. At the same time, FFC is still very much a test-bed for basic research in efficient evaluation of general variational forms.

ACKNOWLEDGMENTS

We wish to thank Johan Hoffman, Johan Jansson, Matthew Knepley, Ola Skavhaug, Andy Terrel, and Garth N. Wells for testing early versions of the compiler and providing constructive feedback and real-world problems that motivated the development of the new improved tabulation of integrals.

REFERENCES

BAGHERI, B. AND SCOTT, L. R. 2003. Analysa. `http://people.cs.uchicago.edu/~ridg/al/aa.html`.

BALAY, S., BUSCHELMAN, K., EIJKHOUT, V., GROPP, W. D., KAUSHIK, D., KNEPLEY, M. G., MCINNES, L. C., SMITH, B. F., AND ZHANG, H. 2004. PETSc Users Manual. Tech. Rep. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory.

BALAY, S., BUSCHELMAN, K., GROPP, W. D., KAUSHIK, D., KNEPLEY, M. G., MCINNES, L. C., SMITH, B. F., AND ZHANG, H. 2006. PETSc. `http://www.mcs.anl.gov/petsc/`.

BALAY, S., GROPP, W. D., MCINNES, L. C., AND SMITH, B. F. 1997. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 163–202.

BANGERTH, W., HARTMANN, R., AND KANSCHAT, G. 2006. `deal.II` Differential Equations Analysis Library. `http://www.dealii.org/`.

BRENNER, S. C. AND SCOTT, L. R. 1994. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag.

CIARLET, P. G. 1976. *Numerical Analysis of the Finite Element Method*. Les Presses de l'Universite de Montreal.

DULAR, P. AND GEUZAINE, C. 2006. GetDP: a General environment for the treatment of Discrete Problems. `http://www.geuz.org/getdp/`.

ERIKSSON, K., ESTEP, D., HANSBO, P., AND JOHNSON, C. 1996. *Computational Differential Equations*. Cambridge University Press.

ERIKSSON, K., ESTEP, D., AND JOHNSON, C. 2003. *Applied Mathematics: Body and Soul*. Vol. III. Springer-Verlag.

HOFFMAN, J., JANSSON, J., JOHNSON, C., KNEPLEY, M. G., KIRBY, R. C., LOGG, A., SCOTT, L. R., AND WELLS, G. N. 2006. FEniCS. `http://www.fenics.org/`.

HOFFMAN, J., JANSSON, J., LOGG, A., AND WELLS, G. N. 2006. DOLFIN. `http://www.fenics.org/dolfin/`.

HOFFMAN, J. AND JOHNSON, C. 2004. *Encyclopedia of Computational Mechanics, Volume 3 (Chapter 7, Computability and Adaptivity in CFD)*. John Wiley.

HUGHES, T. J. R. 1987. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall.

KARNIADAKIS, G. E. AND SHERWIN, S. J. 1999. *Spectral/Hp Element Methods for CFD*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York.

KIRBY, R. C. 2004. FIAT: A new paradigm for computing finite element basis functions. *ACM Trans. Math. Softw. 30*, 502–516.

KIRBY, R. C. 2006a. FIAT. `http://www.fenics.org/fiat/`.

KIRBY, R. C. 2006b. Optimizing FIAT with Level 3 BLAS. *ACM Trans. Math. Softw.*. To appear.

KIRBY, R. C., KNEPLEY, M. G., LOGG, A., AND SCOTT, L. R. 2005. Optimizing the evaluation of finite element matrices. *SIAM J. Sci. Comput. 27,* 3, 741–758.

KIRBY, R. C., KNEPLEY, M. G., AND SCOTT, L. R. 2004. Evaluation of the action of finite element operators. Tech. rep. TR–2004–07, University of Chicago, Department of Computer Science.

KIRBY, R. C. AND LOGG, A. 2006. A compiler for variational forms. *ACM Trans. Math. Softw.*. To appear.

KIRBY, R. C., LOGG, A., SCOTT, L. R., AND TERREL, A. R. 2006. Topological optimization of the evaluation of finite element matrices. *SIAM J. Sci. Comput. 28,* 1, 224–240.

LANGTANGEN, H. P. 1999. *Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*. Lecture Notes in Computational Science and Engineering. Springer-Verlag.

LOGG, A. 2004. Automation of computational mathematical modeling. Ph.D. thesis, Chalmers University of Technology, Sweden.

LOGG, A. 2006. FFC. `http://www.fenics.org/ffc/`.

LONG, K. 2003. Sundance, a rapid prototyping tool for parallel PDE-constrained optimization. In *Large-Scale PDE-Constrained Optimization*. Lecture notes in Computational Science and Engineering. Springer-Verlag.

OLIPHANT, T. ET AL. 2006. *Python Numeric*. URL: `http://numeric.scipy.org/`.

PIRONNEAU, O., HECHT, F., HYARIC, A. L., AND OHTSUKA, K. 2006. FreeFEM. `http://www.freefem.org/`.

Received February 2006; revised June 2006; accepted August 2006