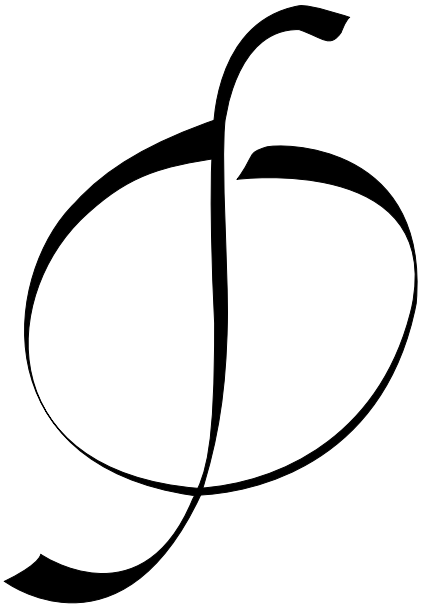

FINITE ELEMENT CENTER

PREPRINT 2007–04

Benchmarking domain-specific compiler optimizations for variational forms

Robert C. Kirby and Anders Logg

FINITE ELEMENT CENTER

PREPRINT 2007–04

Benchmarking domain-specific compiler optimizations for variational forms

Robert C. Kirby and Anders Logg

Finite Element Center
http://www.femcenter.org/

Benchmarking domain-specific compiler optimizations for variational forms

Robert C. Kirby and Anders Logg

Finite Element Center Preprint
NO 2007–04
ISSN 1653–574X

This preprint and other preprints can be found at
http://www.femcenter.org/preprints/

BENCHMARKING DOMAIN-SPECIFIC COMPILER OPTIMIZATIONS FOR VARIATIONAL FORMS

ROBERT C. KIRBY AND ANDERS LOGG

Abstract. We examine the effect of using complexity-reducing relations to generate optimized code for the evaluation of finite element variational forms. The optimizations are implemented in a prototype code named FErari. We demonstrate that by invoking FErari as an optimizing backend to the form compiler FFC, we obtain reduced local operation counts by as much as a factor 7.9 and speedups for the assembly of the global sparse matrix by as much as a factor 2.8.

1. Introduction

Projects such as the FEniCS Form Compiler (hence, FFC) [13, 14, 18], Sundance [19, 20, 21], and deal.II [4] aim to automate important aspects of finite element computation. In the case of FFC, low-level code is generated for the evaluation of element stiffness matrices or their actions, together with the local-to-global mapping. The existence of such a compiler for variational forms naturally leads one to consider an *optimizing* compiler for variational forms. What mathematical structure in the element-level computations is tedious for humans to exploit by hand, but possible for a computer to find? We have provided partial answers to this question in a series of papers [11, 15, 16]. These ideas have been implemented in a prototype code called FErari, and we provide an empirical study of the optimizations implemented by FErari in this paper.

FFC takes as input a multilinear variational form and generates code for evaluating that form over affine elements. The formation of the local stiffness matrix on a single element is expressed as a linear transformation (known at compile-time) applied to a vector representing the geometry and coefficient data (known only at run-time). The linear transformation depends on the variational form and finite element basis, but not on the mesh. The generated code is completely unrolled. This internal kernel is then called for each of the many elements of the mesh at run-time to compute the global sparse matrix.

To a user of FFC, the optimizations are invoked simply with a `-O` flag, which turns on a call to FErari and thence a modified code generator. In assessing the efficacy of these techniques at reducing run-time, we focus on the construction of the sparse matrix and

Key words and phrases. finite element method, variational form, complexity-reducing relations , compiler, optimization, FFC, FErari.

Robert C. Kirby, Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409-1042, *Email:* robert.c.kirby@ttu.edu.

Anders Logg, Simula Research Laboratory, Martin Linges v 17, Fornebu, PO Box 134, 1325 Lysaker, Norway. *Email:* logg@simula.no.

its matrix-free application for a variety of variational forms. In particular, we study the "pure" effect of the FErari optimizations as well as the optimizations relative to the cost of inserting into a sparse matrix data structure.

2. Finite element assembly and the element tensor

2.1. **Multilinear forms.** In finite elements, the nonlinear and linear algebraic problems come from evaluating the variational forms on the finite element basis functions. In our work on FFC and FErari, we have focused on evaluating multilinear forms over affine elements, and we continue to do so here. In particular, we are concerned with the discretization of general *multilinear* forms of arity $r > 0$,

$$a : V_h^1 \times V_h^2 \times \cdots \times V_h^r \to \mathbb{R}, \tag{2.1}$$

defined on the product space $V_h^1 \times V_h^2 \times \cdots \times V_h^r$ of a given set $\{V_h^j\}_{j=1}^r$ of discrete function spaces on a triangulation $\mathcal{T}$ of a domain $\Omega \subset \mathbb{R}^d$. 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. We shall restrict our attention to multilinear forms expressed as integrals over the domain Ω. The typical example is the bilinear form ($r = 2$) for Poisson's equation,

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x. \tag{2.2}$$

Let now $\{\phi_i^1\}_{i=1}^{N^1}, \{\phi_i^2\}_{i=1}^{N^2}, \ldots, \{\phi_i^r\}_{i=1}^{N^r}$ be bases of $V_h^1, V_h^2, \ldots, V_h^r$ respectively and let $i = (i_1, i_2, \ldots, i_r)$ be a multiindex of length $|i| = r$. The multilinear form a then defines a rank r tensor given by

$$A_i = a(\phi_{i_1}^1, \phi_{i_2}^2, \ldots, \phi_{i_r}^r) \quad \forall i \in \mathcal{I}, \tag{2.3}$$

where $\mathcal{I}$ is the index set

$$\mathcal{I} = \prod_{j=1}^{r} [1, |V_h^j|] = \{(1, 1, \ldots, 1), (1, 1, \ldots, 2), \ldots, (N^1, N^2, \ldots, N^r)\}. \tag{2.4}$$

For any given multilinear form of arity r, the tensor A is a (typically sparse) tensor of rank r and dimension $(|V_h^1|, |V_h^2|, \ldots, |V_h^r|) = (N^1, N^2, \ldots, N^r)$.

Typically, the arity of the multilinear form a is $r = 2$ or $r = 1$. When $r = 2$, a is a bilinear form and so the corresponding tensor A is a matrix. When $r = 1$, the corresponding tensor is a vector and when $r = 0$, the corresponding tensor is a scalar.

Forms of higher arity also appear, though they are rarely assembled as a higher-dimensional sparse tensor. As an example, consider the discrete trilinear form $a : V_h^1 \times V_h^2 \times V_h^3 \to \mathbb{R}$ associated with the weighted Poisson equation $-\nabla \cdot (w \nabla u) = f$. The trilinear form a is given by

$$a(v, u, w) = \int_\Omega w \nabla v \cdot \nabla u \, \mathrm{d}x, \tag{2.5}$$

and the corresponding rank three tensor is given by

$$A_i = \int_\Omega \phi^3_{i_3} \nabla\phi^1_{i_1} \cdot \nabla\phi^2_{i_2} \,\mathrm{d}x. \tag{2.6}$$

For any $w = \sum_{i=1}^{N^3} w_i \phi^3_i$, the tensor contraction $A : w = \left(\sum_{i_3=1}^{N^3} A_{i_1 i_2 i_3} w_{i_3}\right)_{i_1 i_2}$ is a matrix. We may thus obtain the solution u by solving the linear system

$$(A : w)u = b, \tag{2.7}$$

where $b_i = L(\phi^1_{i_1}) = \int_\Omega \phi^1_{i_1} f \,\mathrm{d}x$. We typically consider w as a fixed finite element function and directly compute the matrix A associated with the bilinear form $a(\cdot, \cdot, w)$. It may also be desirable, both in trilinear and bilinear forms, to consider the function u as fixed and directly compute a vector A (the *action*) associated with the linear form $a(\cdot, u, w)$. This corresponds to a "matrix-free" application of the matrix, which may be used when solving the linear system with a Krylov method.

2.2. **Assembling the discrete system.** The standard algorithm [23, 10, 17] for computing the matrix A is known as *assembly*; it is computed by iterating over the cells of the mesh $\mathcal{T}$ and adding from each cell the local contribution to the global sparse matrix A. A similar process can compute a global action.

The integral defining a multilinear form a may be written as a sum of integrals over the cells K of a triangulation $\mathcal{T}$ of the domain Ω:

$$a = \sum_{K\in\mathcal{T}} a_K, \tag{2.8}$$

and thus

$$A_i = \sum_{K\in\mathcal{T}} a_K(\phi^1_{i_1}, \phi^2_{i_2}, \ldots, \phi^r_{i_r}). \tag{2.9}$$

For Poisson's equation, the element bilinear form a_K is thus given by $a_K(v, u) = \int_K \nabla v \cdot \nabla u \,\mathrm{d}x$.

With $n^j_K = |\mathcal{P}_K|$, the dimension of the local finite element space on K, we now let $\iota^j_K : [1, n^j_K] \to [1, N^j]$ denote the standard local-to-global mapping for each discrete function space V^j_h, $j = 1, 2, \ldots, r$, and define for each $K \in \mathcal{T}$ the collective local-to-global mapping $\iota_K : \mathcal{I}_K \to \mathcal{I}$ by

$$\iota_K(i) = (\iota^1_K(i_1), \iota^2_K(i_2), \ldots, \iota^r_K(i_r)) \quad \forall i \in \mathcal{I}_K, \tag{2.10}$$

where $\mathcal{I}_K$ is the index set

$$\mathcal{I}_K = \prod_{j=1}^{r} [1, |\mathcal{P}^j_K|] = \{(1, 1, \ldots, 1), (1, 1, \ldots, 2), \ldots, (n^1_K, n^2_K, \ldots, n^r_K)\}. \tag{2.11}$$

Furthermore, for each V^j_h we let $\{\phi^{K,j}_i\}_{i=1}^{n^j_K}$ denote the restriction to an element K of the subset of the basis $\{\phi^j_i\}_{i=1}^{N^j}$ of V^j_h supported on K, and for each $i \in \mathcal{I}$ we let $\mathcal{T}_i \subset \mathcal{T}$ denote the subset of cells on which all of the basis functions $\{\phi^j_{i_j}\}_{j=1}^r$ are supported.

We may now compute A by summing the contributions from each local cell K,

$$\begin{aligned} A_{\iota_K(i)} &= \sum_{K \in \mathcal{T}} a_K(\phi^1_{\iota_K(i_1)}, \phi^2_{\iota_K(i_2)}, \ldots, \phi^r_{\iota_K(i_r)}) \\ &= \sum_{K \in \mathcal{T}_i} a_K(\phi^1_{\iota_K(i_1)}, \phi^2_{\iota_K(i_2)}, \ldots, \phi^r_{\iota_K(i_r)}) \\ &= \sum_{K \in \mathcal{T}_i} a_K(\phi^{K,1}_{i_1}, \phi^{K,2}_{i_2}, \ldots, \phi^{K,r}_{i_r}) = A^K_i, \end{aligned} \tag{2.12}$$

where A^K is the local *element tensor* on cell K (the "element stiffness matrix"). This computation may be carried out by iterating once over all cells $K \in \mathcal{T}$ and adding the contribution from each K to each entry A_i of A such that $K \in \mathcal{T}_i$.

Our work in [13, 14] has focused on a general paradigm for efficiently constructing the element tensor A^K. In [11, 15, 16], we have explored special mathematical structure that leads to reduced operation counts. However, it was studied only in a limited case what the net impact of FErari operations were on overall assembly time, especially when the cost of global assembly is counted as well. Additionally, to assess the overall efficacy of the optimizations we have proposed, the relative costs of computing local matrices or vectors and collecting them into the global data structure must be considered.

2.3. **Sparse matrix insertion.** In principle, summing the entries of A^K into the global tensor A may be accomplished by iterating over all $i \in I_K$ and adding the entry A^K_i at position $\iota_K(i)$ of A as illustrated in Figure 1. Libraries such as PETSc [2, 1, 3] and Trilinos [5, 6] provide high-level operations for doing just this. Use of such libraries is encouraged, for they handle most of the parallel issues, provide good performance, and give access to many advanced solver algorithms. Despite their success, however, many external factors affect the actual performance they can provide. First, data locality plays an important role in assembling either vectors or matrices. If the local degrees of freedom on an element are mapped to entries close together in the global ordering, then memory performance will be better. In addition, several additional factors affect the performance of assembling a sparse matrix. These include the storage format (compressed sparse row or column, coordinate indices, etc.), whether the items to be inserted are pre-sorted, how memory has been preallocated, and what internal data structures are used.

2.4. **A representation formula for the element tensor.** It has long been known that precomputing certain integrals on the reference element can speed up computation of the element tensor, especially for bilinear forms with straight-sided elements. A general approach to precomputing certain integrals was first introduced in [12, 11] and later formalized and automated in [13, 14]. A similar approach was implemented in early versions of DOLFIN [9, 7, 8], but only for piecewise linear elements.

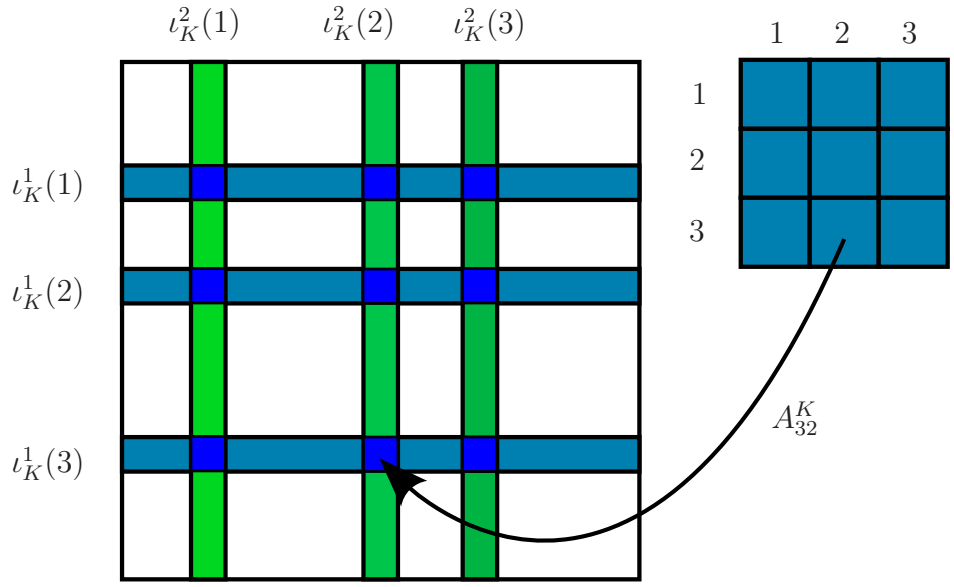

FIGURE 1. Adding the entries of the element tensor A^K to the global tensor A using the local-to-global mapping ι_K, illustrated here for a rank two tensor (a matrix).

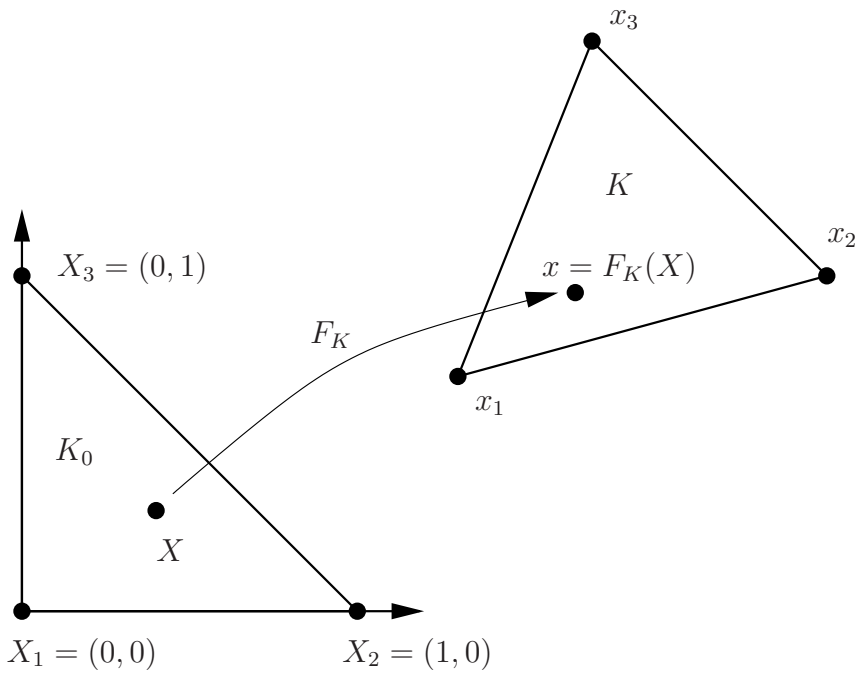

FIGURE 2. The (affine) mapping F_K from a reference cell K_0 to some cell $K \in \mathcal{T}$.

When the mapping F_K from the reference cell is affine (Figure 2), we have for the Laplacian

$$A_i^K = \int_K \nabla \phi_{i_1}^{K,1} \cdot \nabla \phi_{i_2}^{K,2} \, \mathrm{d}x = \int_K \sum_{\beta=1}^{d} \frac{\partial \phi_{i_1}^{K,1}}{\partial x_\beta} \frac{\partial \phi_{i_2}^{K,2}}{\partial x_\beta} \, \mathrm{d}x, \tag{2.13}$$

whence a change of variables yields

$$A_i^K = \sum_{\alpha \in \mathcal{A}} A_{i\alpha}^0 G_K^\alpha \quad \forall i \in \mathcal{I}_K, \tag{2.14}$$

or simply

$$A^K = A^0 : G_K, \tag{2.15}$$

where

$$\begin{aligned} A_{i\alpha}^0 &= \int_{K_0} \frac{\partial \Phi_{i_1}^1}{\partial X_{\alpha_1}} \frac{\partial \Phi_{i_2}^2}{\partial X_{\alpha_2}} \, \mathrm{d}X, \\ G_K^\alpha &= \det F_K' \sum_{\beta=1}^{d} \frac{\partial X_{\alpha_1}}{\partial x_\beta} \frac{\partial X_{\alpha_2}}{\partial x_\beta}. \end{aligned} \tag{2.16}$$

We refer to the tensor A^0 as the *reference tensor* and to the tensor G_K as the *geometry tensor*. Furthermore, we refer to $\mathcal{I}_K$ as the set of primary indices and to $\mathcal{A}$ as the set of secondary indices. The tensor representation (2.15) generalizes to other multilinear forms. This is made precise in [14]. The ranks of the tensors A^0 and G_K for a particular variational form a are determined mechanically, from properties such as the number of coefficients and derivatives.

We remark that in general, a multilinear form will correspond to a sum of tensor contractions, rather than a single tensor contraction,

$$A^K = \sum_k A^{0,k} : G_{K,k}. \tag{2.17}$$

One such example is the computation of the element tensor for the diffusion–reaction problem $-\Delta u + u = f$, which may be computed as the sum of a tensor contraction of a rank four reference tensor $A^{0,1}$ with a rank two geometry tensor $G_{K,1}$ and a rank two reference tensor $A^{0,2}$ with a rank zero geometry tensor $G_{K,2}$.

3. A framework for optimization

In this section, we present an overview of our framework for optimization of variational form evaluation. Two different approaches are presented. The first is a coarse-grained strategy based on phrasing the tensor contraction (2.15) as a matrix-vector or matrix-matrix multiplication that may be computed by an optimized library call. The second, which is what FErari implements, exploits the structure of the tensor contraction to find an optimized computation with a reduced operation count.

3.1. **Tensor contraction as a matrix-vector product.** To evaluate the element tensor A^K, one must evaluate the tensor contraction (2.15). A simple approach would be to iterate over the entries $\{A_i^K\}_{i\in\mathcal{I}_K}$ of A^K and for each entry A_i^K compute the value of the entry by summing over the set of secondary indices $\mathcal{A}$. However, by an appropriate reshaping of the tensors A^K, A^0 and G_K, one may phrase the tensor contraction as a matrix–vector product and call an optimized library routine for the computation of the matrix–vector product, such as the level 2 BLAS routine DGEMV.

To see this, let $\{i^j\}_{j=1}^{|\mathcal{I}_K|}$ be an enumeration of the set of primary multiindices $\mathcal{I}_K$ and let $\{\alpha^j\}_{j=1}^{|\mathcal{A}|}$ be an enumeration of the set of secondary multiindices $\mathcal{A}$. As an example, for the computation of the 6×6 element tensor for Poisson's equation with quadratic elements on triangles, we may enumerate the primary and secondary multiindices by

$$\begin{aligned}\{i^j\}_{j=1}^{|\mathcal{I}_K|} &= \{(1,1),(1,2),\ldots,(1,6),(2,1),\ldots,(6,6)\},\\ \{\alpha^j\}_{j=1}^{|\mathcal{A}|} &= \{(1,1),(1,2),(2,1),(2,2)\}.\end{aligned} \tag{3.1}$$

By similarly enumerating the 36 entries of the 6×6 element tensor A^K and the four entries of the 2×2 geometry tensor G_K, one may define two vectors $a^K\in\mathbb{R}^{36}$ and $g_K\in\mathbb{R}^4$ corresponding to the two tensors A^K and G_K respectively.

In general, the element tensor A^K and the geometry tensor G_K may be flattened to create the corresponding vectors $a^K \leftrightarrow A^K$ and $g_K \leftrightarrow G_K$, defined by

$$\begin{aligned}a^K &= (A_{i^1}^K, A_{i^2}^K, \ldots, A_{i^{|\mathcal{I}_K|}}^K)^\top,\\ g_K &= (G_K^{\alpha^1}, G_K^{\alpha^2}, \ldots, G_K^{\alpha^{|\mathcal{A}|}})^\top.\end{aligned} \tag{3.2}$$

Similarly, we define the $|\mathcal{I}_K|\times|\mathcal{A}|$ matrix $\bar{A}^0$ by

$$\bar{A}_{jk}^0 = A_{i^j\alpha^k}^0, \quad j=1,2,\ldots,|\mathcal{I}_K|, \quad k=1,2,\ldots,|\mathcal{A}|. \tag{3.3}$$

Since now

$$a_j^K = A_{i^j}^K = \sum_{\alpha\in\mathcal{A}} A_{i^j\alpha}^0 G_K^\alpha = \sum_{k=1}^{|\mathcal{A}|} A_{i^j\alpha^k}^0 G_K^{\alpha^k} = \sum_{k=1}^{|\mathcal{A}|} \bar{A}_{jk}^0 (g_K)_k, \tag{3.4}$$

it follows that the tensor contraction $A^K = A^0 : G_K$ corresponds to the matrix–vector product

$$a^K = \bar{A}^0 g_K. \tag{3.5}$$

Of course, once the computation of one a^K may be computed as a matrix-vector product, the computation of $\{a^{K_i}\}_{i=1}^M$ for some M elements of the mesh can naturally be encoded as a matrix-matrix multiplication. Using DGEMM in such a context is an example of coarse-grained optimization, making good use of cache in a large computation. Such an approach necessarily overlooks problem-specific optimizations such as we find in FErari, but may be very effective in many circumstances. It is to be expected that which approach is preferable will depend strongly on how much structure FErari finds and how well the resulting algorithms are mapped onto hardware, as well as whether $\bar{A}^0$ is large enough for DGEMM to be fast. We do not explore the coarse-grained strategy further in this paper.

3.2. **Complexity-reducing relations.** The matrix $\bar{A}^0$ is computed at compile-time by FFC, and it typically possesses significant structure that can be exploited to reduce the amount of arithmetic needed to multiply it by a vector g_K.

Letting $g_K \in \mathbb{R}^{|\mathcal{A}|}$ be the vector obtained by flattening the geometry tensor G_K as above, we note that each entry A_i^K of the element tensor A^K is given by the inner product

$$A_i^K = a_i^0 \cdot g_K, \tag{3.6}$$

where a_i^0 is the vector defined by

$$a_i^0 = (A_{i\alpha^1}^0, A_{i\alpha^2}^0, \ldots, A_{i\alpha^{|\mathcal{A}|}}^0)^\top, \tag{3.7}$$

corresponding to a row in the matrix $\bar{A}^0$.

To optimize the evaluation of the element tensor, we look for dependencies between the vectors $\{a_i^0\}_{i \in \mathcal{I}_K}$ that can be used to reduce the operation count. For example, if two vectors a_i^0 and $a_{i'}^0$ are collinear, then $a_i^0 \cdot g_K$ may be computed using $a_{i'}^0 \cdot g_K$ in only one multiply, and vice versa. If the Hamming distance (number of different entries between a_0^i and $a_0^{i'}$) is k, then the result $a_{i'}^0 \cdot g_K$ can be computed from $a_i^0 \cdot g_K$ in about k multiply-add pairs, and vice versa. These kinds of relations are called "complexity-reducing relations", and they are related to common subexpressions. In [15], we constructed a weighted, undirected graph, the vertices of which were the vectors a_i^0 and the weights of whose edges were the pairwise distances under a complexity-reducing relation. We proved that a minimum spanning tree of this graph encodes a minimal-arithmetic (in a specific sense) algorithm for evaluating the product of $\bar{A}^0$ with an arbitrary input vector.

In fact, other kinds of structure is to be found in $\bar{A}^0$, such as when three or more rows are linearly dependent. A first attempt at exploiting this structure is found in [16], but our present work is limited to complexity-reducing relations.

4. Benchmark results

For a range of forms and polynomial degrees, we report several quantities for forming the matrix and its action. First, we report the base operation count $|\mathcal{I}_K|\,|\mathcal{A}|$ for forming the element tensor A^K, as well as the operation counts generated by FFC[1] and the FErari optimizations. Having generated code for the local element computation from both FFC and FErari, we compare the run-time for these codes being executed several times. This measures the efficacy of FErari at exactly the point it seeks to optimize. Then, to provide a broader context, we present the speedup obtained in the global assembly process, when the overhead of sparse data structures is included.

In each case, we generated code for the local and global computation both with and without FErari optimizations. This code was compiled and run on an IBM Thinkpad T60p with 2GB of RAM and a dual core Intel T2600 chip running at 2.16 GHz. The operating system was Ubuntu Linux with kernel 2.6.17-10-386. The compiler was `g++` version 4.1.2 using optimization flag `-O2` on all variational forms except the weighted

[1]FFC reduces the base operation count by omitting computation of zeros when the element tensor is sparse.

Laplacian operator and action using quartics in 3D. The compiler and machine could only handle optimization mode `-O0` in these cases. For two-dimensional problems, we used a regular triangulation based on subdividing a 64×64 square mesh into right triangles, resulting in a total of 4,225 vertices and 8,192 triangles. For three dimensions, we used a $16 \times 16 \times 16$ partition of the unit cube into 4,913 vertices and 24,576 tetrahedra. The timing was performed adaptively to ensure that at least one second of CPU time elapsed for a set of at least ten repetitions for each test case. For the sparse matrix data structure, a simple `std::vector<std::map<unsigned int, double> >` was used, which was found competitive with insertion into a sparse PETSc matrix.

In most cases, we find decent speedup in the operation count, although it does not always translate into a speedup in the runtime for the local computation. FErari is currently architecture-unaware. Rearranging the matrix-vector computation in a way that makes poor use of registers, for example, can more than offset reductions in the actual amount of arithmetic. A better result would be obtained by somehow combining the graph-based optimizations with an architecture model, or using a special-purpose compiler such as Spiral [22].

Moreover, even a speedup in local computation does not always improve the global cost of assembling a matrix or vector. If a relatively small amount of work is required to compute A^K, then the cost of assembling it into the global matrix or vector may dominate; reductions in arithmetic are not significant. On the other hand, when the construction of A^K is relatively expensive, then speedup in the construction of the global matrix or vector can be realized by reduction of arithmetic in the local computation. In our empirical results, we observe a tendency of FErari to provide better global speedups for more complicated variational forms.

4.1. **Laplacian.** First, we consider the Laplacian, with the variational form

$$a(v, u) = \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x. \tag{4.1}$$

We use Lagrange polynomials P_k of degree $k = 1, 2, \ldots, 5$ on triangles and degree $k = 1, 2, \ldots, 4$ on tetrahedra.[2] When forming the element matrix for some K, the matrix $\bar{A}^0$ has $m = |P_k|^2$ rows and $n = d^2$ columns. For particular values of k and d, we tabulate the size of $\bar{A}^0$ and display the operation counts for the FFC- and FErari-generated algorithms in Table 1. We also give the base operation count mn which is the number of operations needed to compute the entries of the element tensor A^K with no optimization.

In each case, FErari provides up to about a factor of three improvement in operation count. The reduction in operation count, local computation time, and global computation time required is plotted in Figure 3. The reduction in arithmetic reduces the run-time to evaluate the local stiffness matrix (multiplying by $\bar{g}_K$) by a factor of 1.5 to 2 in both two and three dimensions. However, the reduction does not have a major impact on the global time to assemble the matrix. In this case, there are very few arithmetic operations needed

[2]The polynomial degree on tetrahedra was limited by available resources to compute the optimization.

Triangles

k	m	n	mn	FFC	FErari
1	9	4	36	16	11
2	36	4	144	64	21
3	100	4	400	252	71
4	225	4	900	712	236
5	441	4	1764	1480	523

Tetrahedra

k	m	n	mn	FFC	FErari
1	16	9	144	36	28
2	100	9	900	381	196
3	400	9	3600	1968	848
4	1225	9	11025	6924	2494

TABLE 1. Base operation count mn versus FFC- and FErari-generated operation counts for forming the element stiffness matrix for the Laplacian (4.1).

Triangles

k	m	n	mn	FFC	FErari
1	3	12	36	16	12
2	6	24	144	64	52
3	10	40	400	252	189
4	15	60	900	712	566
5	21	84	1764	1480	1300

Tetrahedra

k	m	n	mn	FFC	FErari
1	4	36	144	36	30
2	10	90	900	381	330
3	20	180	3600	1968	1540
4	35	315	11025	6924	5546

TABLE 2. Base operation count mn versus FFC- and FErari-generated operation counts for applying the element stiffness matrix for the Laplacian (4.1).

to construct the local matrix, and the cost of inserting into the global matrix overshadows the gains FErari provides.

We also consider the matrix action as needed in a Krylov solver. Assembling into a global vector is less expensive than into a global matrix, and we see better speedups in evaluating the action of the Laplacian operator. In this case, FFC and FErari generate code for evaluating (4.1) with u a member of the finite element space. Speedup of this operation is felt at each iteration of a Krylov method and so translates directly into decreased solve time. The matrix $\bar{A}^0$ has the same entries as for forming the stiffness matrix, but has a different shape. In this case, the shape is $|P_k| \times (d^2|P_k|)$. Table 2 shows the dimensions of $\bar{A}^0$ and the FFC and FErari operation counts. Note that FErari does not do as well for the action as for forming the matrix. Although the entries of $\bar{A}^0$ are the same as before, the difference in shapes complicates finding collinear relationships. When the rows have only d^2 (4 or 9) entries for the stiffness matrix, more collinearity is found than when there are $|P_k|$ times as many entries. However, finding Hamming distance relations is as effective as before. Despite the smaller reduction in operation count, the effect of the optimizations on run-time is much greater than in forming the matrix, as we can see by comparing Figure 4 to Figure 3. A global speedup of about 10% is observed for degrees three through five in two dimensions, and a speedup of 20%–40% for quadratics through quartics in three dimensions. Again, only a small improvement is observed for low order methods.

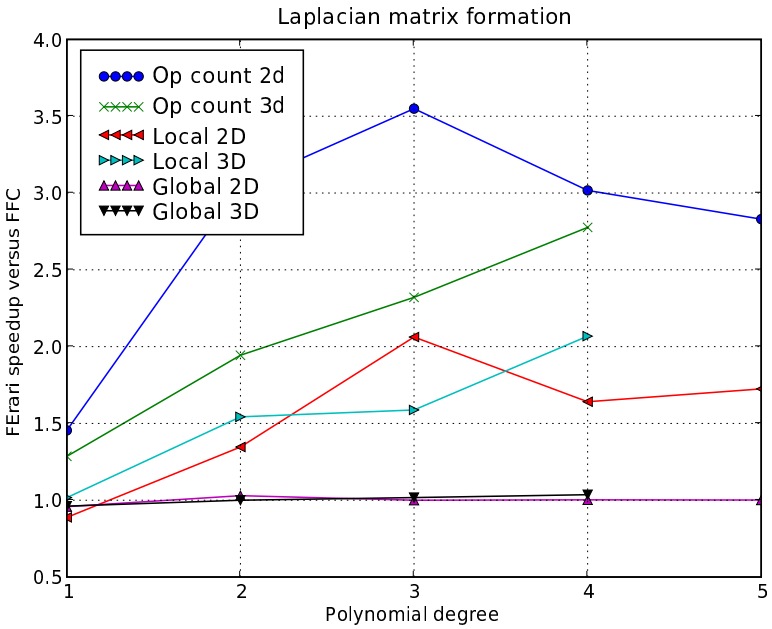

FIGURE 3. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the Laplacian (4.1).

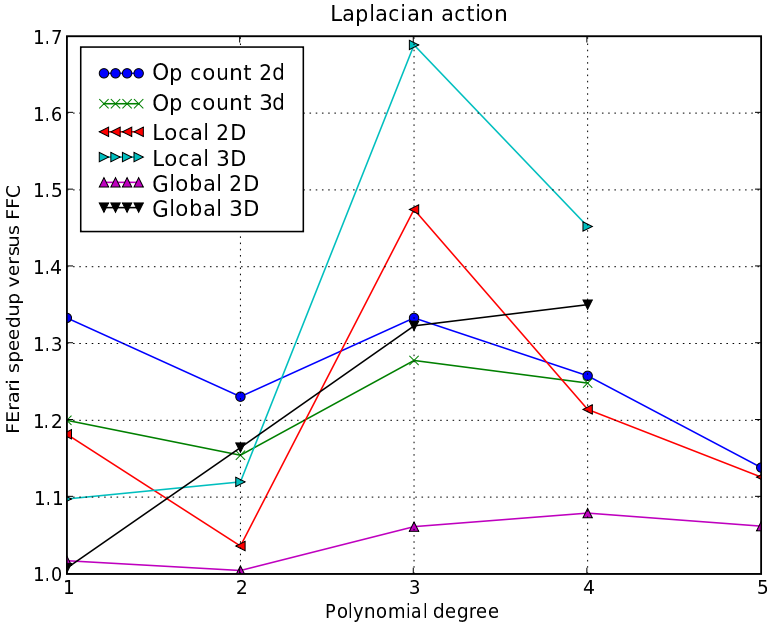

FIGURE 4. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the action of the Laplacian (4.1).

Triangles

k	m	n	mn	FFC	FErari
1	9	12	108	48	33
2	36	24	864	552	331
3	100	40	4000	2992	1976
4	225	60	13500	11512	7758
5	441	84	37044	33260	23120

Tetrahedra

k	m	n	mn	FFC	FErari
1	16	36	576	144	112
2	100	90	9000	4122	3101
3	400	180	72000	42846	30771
4	1225	315	385875	272832	196910

TABLE 3. Base operation count mn versus FFC- and FErari-generated operation counts for forming the element stiffness matrix for the weighted Laplacian (4.2).

Triangles

k	m	n	mn	FFC	FErari
1	3	36	108	48	36
2	6	144	864	552	443
3	10	400	4000	2992	2596
4	15	900	13500	11512	10625
5	21	1764	37044	33260	31410

Tetrahedra

k	m	n	mn	FFC	FErari
1	4	144	576	144	120
2	10	900	9000	4122	3561
3	20	3600	72000	42846	36394
4	35	11025	385875	272832	238641

TABLE 4. Base operation count mn versus FFC- and FErari-generated operation counts for applying the element stiffness matrix for the weighted Laplacian (4.2).

4.2. **Weighted Laplacian.** Now, we consider the form

$$a(v, u, w) = \int_\Omega w \nabla v \cdot \nabla u \, \mathrm{d}x, \tag{4.2}$$

for a fixed weight w where we assume that v, u, w all come from the same Lagrange finite element space. In this case, the presence of the coefficient w makes the local form more expensive to evaluate. The matrix $\bar{A}_0$ now has $|P_k|^2$ rows and $d^2|P_k|$ columns. However, the graph of the global matrix for this form is the same as for the constant coefficient case, assuming the same basis and mesh are used. Consequently, the cost of assembly is exactly the same once A^K is constructed.

Table 3 tabulates the shape of $\bar{A}^0$ for the same degrees as the constant coefficient case, as well as the operation count resulting from FFC and FErari. Again, FErari reduces the operation count and run-time for the local computation considerably. Given that the arithmetic cost is much larger than for the constant-coefficient case, it is not surprising that the global speedups are much better, as seen in Figure 5.

As before, $\bar{A}^0$ has the same entries but a different shape when the action of the form is considered. Now, the shape is $|P_k| \times (d^2|P_k|^2)$. The shape of $\bar{A}^0$ and the FFC- and FErari-generated operation counts are shown in Table 4. While FErari does not reduce the operation count for the matrix action as significantly as it does for the matrix itself, the global speedups are more significant (Figure 6).

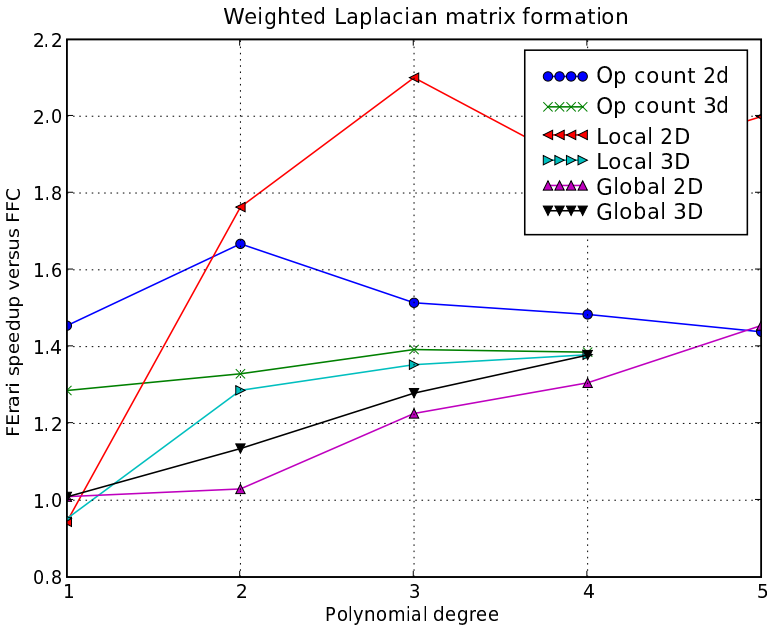

FIGURE 5. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the weighted Laplacian (4.2).

4.3. **Advection.** Next, we consider the advection operator

$$a(v,u) = \int_\Omega v(\beta \cdot \nabla u)\,\mathrm{d}x, \tag{4.3}$$

where β is some constant vector and consider forming the global stiffness matrix and its action. For the matrix, the dimension of $\bar{A}^0$ is $|P_k|^2 \times d^3$, and these values together with the operation counts obtained from FFC and FErari are presented in Table 5. The advection β is defined as a piecewise constant vector-valued Lagrange function which has d degrees of freedom on each element. As a result, the matrix $\bar{A}^0$ is physically of dimension $|P_k|^2 \times d^3$, but the number of nonzero elements scales like $|P_k|^2 \times d^2$. This is because the reference tensor A^0 generating the matrix $\bar{A}^0$ is formed as an outer product with $\Phi_{\alpha_1}[\alpha_2] = \delta_{\alpha_1\alpha_2}$, that is, component α_2 of the piecewise constant vector-valued basis function Φ_{α_1}. Precontracting the reference tensor along dimensions α_1, α_2 would thus reduce the size of the matrix $\bar{A}^0$ to $|P_k|^2 \times d^2$. Low-order elements like piecewise constants and linears often generate particular structures that can be used for further optimizations. Such optimizations are not handled by FErari and are an interesting venue for further research.

As with forming the Laplacian, the reduced operation counts (Table 5) do not significantly affect the global runtime (Figure 7). The operation counts and speedups for the matrix action are found in Table 6 and Figure 8. Global speedup is again most significant for higher order elements in three dimensions.

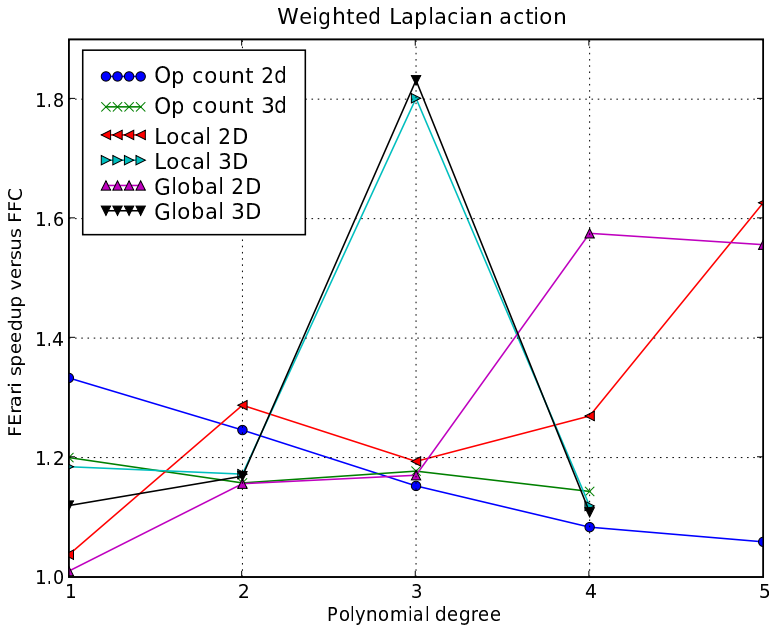

FIGURE 6. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the action of the weighted Laplacian (4.2).

Triangles

k	m	n	mn	FFC	FErari
1	9	8	72	24	6
2	36	8	288	112	35
3	100	8	800	336	95
4	225	8	1800	776	185
5	441	8	3528	1592	449

Tetrahedra

k	m	n	mn	FFC	FErari
1	16	27	432	72	15
2	100	27	2700	558	71
3	400	27	10800	2646	450
4	1225	27	33075	8730	1335

TABLE 5. Base operation count mn versus FFC- and FErari-generated operation counts for forming the element stiffness matrix for the advection operator (4.3).

Triangles

k	m	n	mn	FFC	FErari
1	3	24	72	24	8
2	6	48	288	112	92
3	10	80	800	336	306
4	15	120	1800	776	712
5	21	168	3528	1592	1508

Tetrahedra

k	m	n	mn	FFC	FErari
1	4	108	432	72	18
2	10	270	2700	558	387
3	20	540	10800	2646	2070
4	35	945	33075	8730	7428

TABLE 6. Base operation count mn versus FFC- and FErari-generated operation counts for applying the element stiffness matrix for the advection operator (4.3).

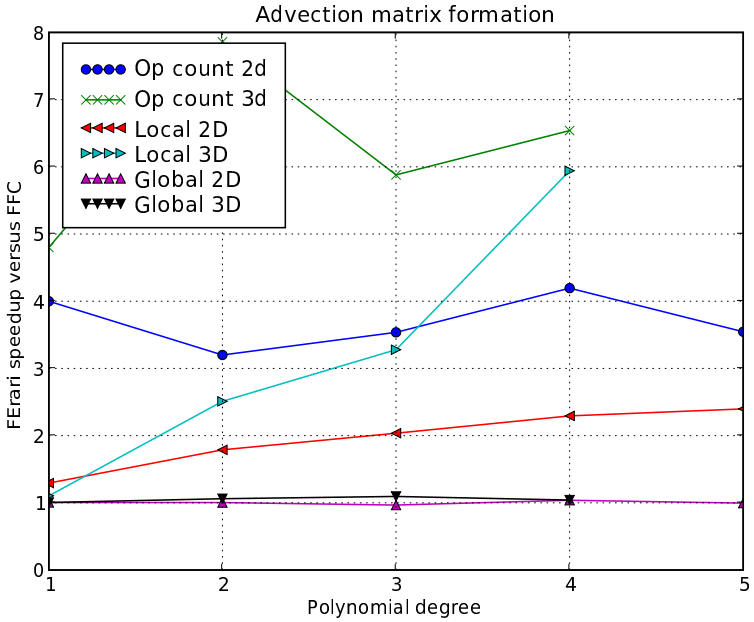

FIGURE 7. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the advection operator (4.3).

4.4. **Weighted advection in a coordinate direction.** Finally, we consider the advection operator oriented along a coordinate axis, but with the velocity field varying in space (projected into the finite element space):

$$a(v, u, w) = \int_\Omega vw \frac{\partial u}{\partial x_1} \, \mathrm{d}x, \tag{4.4}$$

We consider forming the matrix and its action for a fixed weight w. This operator is a portion of the trilinear momentum advection term in the Navier-Stokes equations. For constructing the matrix, we observe a nice speedup in local computation, although in two dimensions this has only a marginal effect on the global run-time for assembly. However, we gain significantly for higher-order elements in three dimensions, where we see a global speedup with 180% for quartics. The operation counts for the local matrix construction and action are shown in Tables 7 and 8, and the speedups are shown in Figures in 9 and 10.

4.5. **Speedup versus work.** As we noted before, reducing floating-point arithmetic is expected to be more significant to the global computation when the individual entries in the local matrix or vector are already expensive to compute. As a test of this, we plot the speedup of FErari over FFC against the number of columns in each reference operator $\bar{A}^0$ in Figure 11. We do this for all orders and forms, considering matrices and their actions

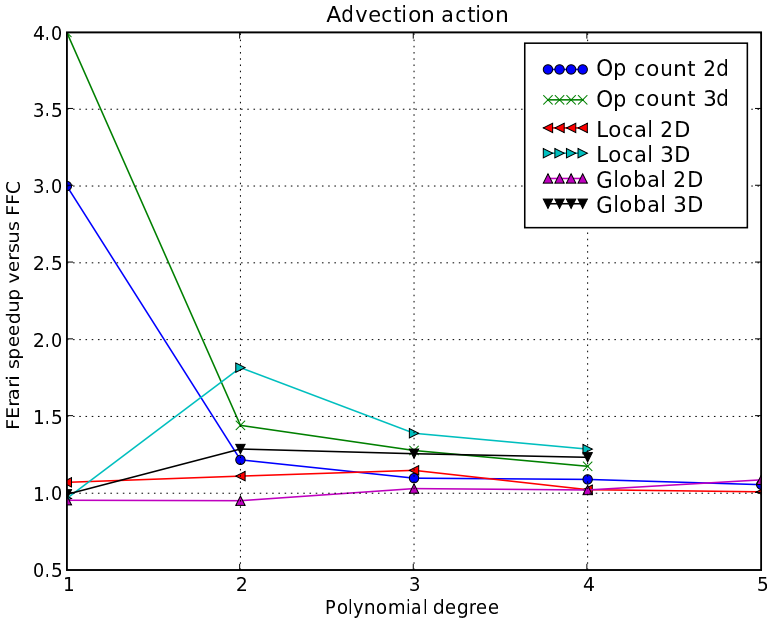

FIGURE 8. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the action of the advection operator (4.3).

Triangles

k	m	n	mn	FFC	FErari
1	9	6	54	36	23
2	36	12	432	344	247
3	100	20	2000	1660	1301
4	225	30	6750	6168	5209
5	441	42	18522	17436	15194

Tetrahedra

k	m	n	mn	FFC	FErari
1	16	12	192	96	56
2	100	30	3000	1686	984
3	400	60	24000	18582	11723
4	1225	105	128625	107178	56555

TABLE 7. Base operation count mn versus FFC- and FErari-generated operation counts for forming the element stiffness matrix for the advection operator(4.4).

Triangles

k	m	n	mn	FFC	FErari
1	3	18	54	36	28
2	6	72	432	344	320
3	10	200	2000	1660	1595
4	15	450	6750	6168	6082
5	21	882	18522	17436	17091

Tetrahedra

k	m	n	mn	FFC	FErari
1	4	48	192	96	60
2	10	300	3000	1686	1399
3	20	1200	24000	18582	15243
4	35	3675	128625	107178	78617

TABLE 8. Base operation count mn versus FFC- and FErari-generated operation counts for forming the element stiffness matrix for the advection operator(4.4).

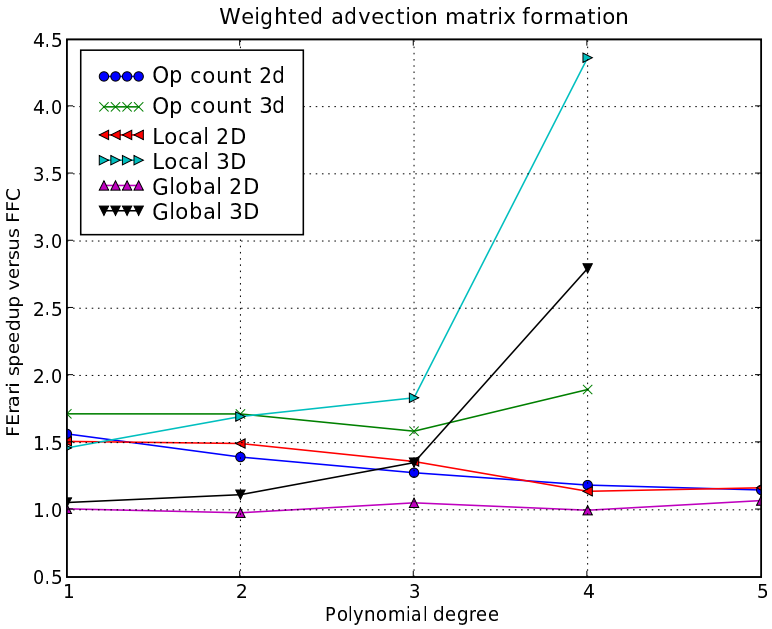

FIGURE 9. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the weighted advection operator (4.4).

separately. Although it is not an exact relation (as to be expected), Figure 11 does indicate a general trend of speedup increasing with the base cost of work per entry.

5. Conclusions

Several things emerge from our empirical study of optimizing FFC with FErari. In certain contexts, FErari can provide tens of percent speedup in runtime in forming or applying stiffness matrices. Moreover, these cases tend to be the computationally harder ones (three dimensions, higher order polynomials). However, FErari is not without its costs. It dramatically adds to the compile-time for FFC, and when used for simple forms can actually hinder runtime.

Besides improving the run-time performance of finite element codes generated by FFC and FErari, our results shed some light on where FErari could be improved and in how a fully functional optimizing compiler for finite elements might be developed. First, our calculations did little to optimally order the degrees of freedom; better ordering algorithms should decrease the cost of insertion. Second, algorithms trying to maximize performance must have some awareness of the underlying computer architecture. The success of Spiral in signal processing suggests this should be possible. Moreover, knowing when to do what kinds of optimization, such as FErari's fine-grained optimization versus a coarse-grained level 3 BLAS approach, must be determined. This must also be compared against when quadrature-based algorithms might be effective, as well as whether the stiffness matrix

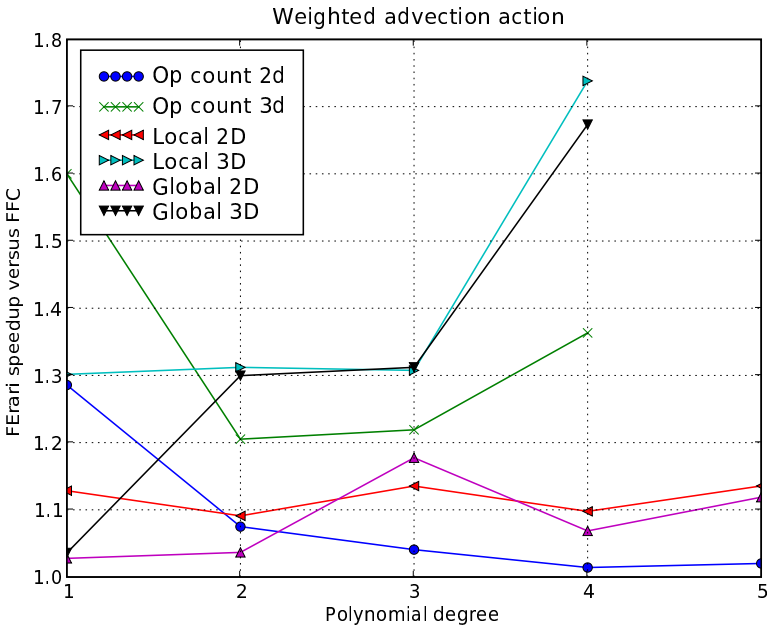

FIGURE 10. Speedup in operation count, local run-time and global run-time for using FErari versus FFC only for the action of the weighted advection operator (4.4).

should be explicitly constructed or if it is suitable to compute directly the action of the matrix.

REFERENCES

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

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

[3] S. BALAY, W. D. GROPP, L. C. MCINNES, AND B. F. SMITH, *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, 1997, pp. 163–202.

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

[5] M. HEROUX, R. BARTLETT, V. H. R. HOEKSTRA, J. HU, T. KOLDA, R. LEHOUCQ, K. LONG, R. PAWLOWSKI, E. PHIPPS, A. SALINGER, H. THORNQUIST, R. TUMINARO, J. WILLENBRING, AND A. WILLIAMS, *An Overview of Trilinos*, Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003.

[6] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS, A. G. SALINGER, H. K. THORNQUIST, R. S. TUMINARO, J. M. WILLENBRING, A. WILLIAMS, AND K. S. STANLEY, *An overview of the Trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

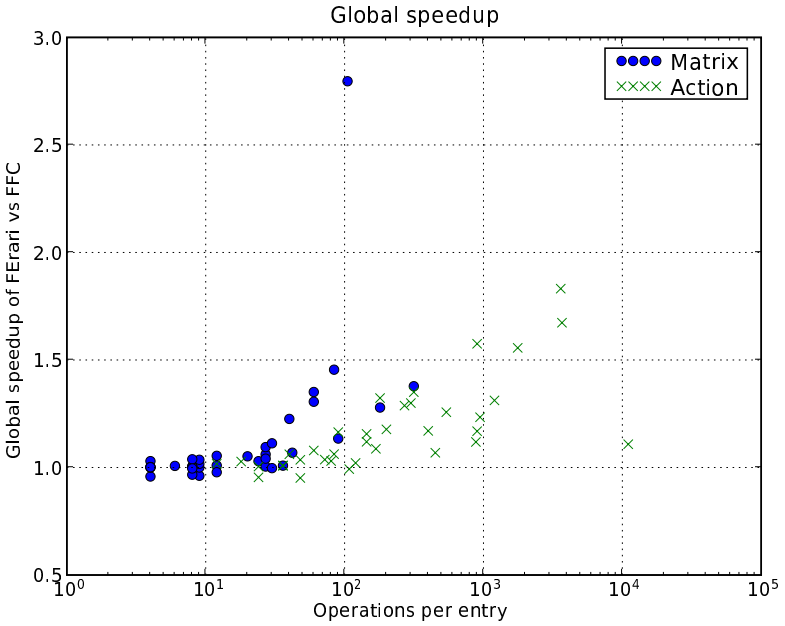

FIGURE 11. The global speedup that FErari produces over FFC is plotted against the number of columns in the associated reference matrix $\bar{A}^0$, which is a measure of the work required to compute each entry of A^K.

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

[8] ———, *DOLFIN User Manual*, 2006.

[9] J. HOFFMAN AND A. LOGG, *DOLFIN: Dynamic Object oriented Library for FINite element computation*, Tech. Rep. 2002–06, Chalmers Finite Element Center Preprint Series, 2002.

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

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

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

[13] R. C. KIRBY AND A. LOGG, *A compiler for variational forms*, ACM Transactions on Mathematical Software, 32 (2006), pp. 417–444.

[14] ———, *Efficient compilation of a class of variational forms*, ACM Transactions on Mathematical Software, 33 (2007).

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

[16] R. C. KIRBY AND L. R. SCOTT, *Geometric optimization of the evaluation of finite element matrices*, to appear in SIAM J. Sci. Comput., (2007).

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

[18] A. LOGG, *FFC*, 2007. `http://www.fenics.org/ffc/`.

[19] K. Long, *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, 2003.
[20] ——, *Sundance 2.0 tutorial*, Tech. Rep. TR–2004–09, Sandia National Laboratories, 2004.
[21] ——, *Sundance*, 2006. URL: `http://software.sandia.gov/sundance/`.
[22] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo, *SPIRAL: Code generation for DSP transforms*, Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation", 93 (2005), pp. 232–275.
[23] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The Finite Element Method — Its Basis and Fundamentals, 6th edition*, Elsevier, 2005, first published in 1967.