A

Optimizing Finite Element Integration through Automated Expression Re-writing and Code Specialization

Fabio Luporini, Imperial College London
David A. Ham, Imperial College London
Paul H.J. Kelly, Imperial College London

Abstract goes here

Categories and Subject Descriptors: G.1.8 [**Numerical Analysis**]: Partial Differential Equations - Finite element methods; G.4 [**Mathematical Software**]: Parallel and vector implementations

General Terms: Design, Performance

Additional Key Words and Phrases: Finite element integration, local assembly, compilers, optimizations, SIMD vectorization

ACM Reference Format:
Fabio Luporini, David A. Ham, and Paul H. J. Kelly, 2014. Optimizing Finite Element Integration through Automated Expression Re-writing and Code Specialization. *ACM Trans. Arch. & Code Opt.* V, N, Article A (January YYYY), 3 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

2. PRELIMINARIES

2.1. Quadrature for Finite Element Local Assembly

2.2. Code Generation for Quadrature Representation

Rapid implementation of high performance, robust, and portable code evaluating element matrices using quadrature can be achieved through automated code generation. This has been succesfully proved in the context of the popular FEniCS project [Logg et al. 2012]. The FEniCS Form Compiler (FFC) accepts as input the variational form of a partial differential equation and generates C++ code implementing local assembly routines. The variational form is expressed at high-level by means of the domain-specific Unified Form Language (UFL). Local assembly code must be high performance: as the complexity of a form increases, in terms of number of derivatives, pre-multiplying functions, and polynomial order of the chosen functions, the resulting ker-

---

This research is partly funded by the MAPDES project, by the Department of Computing at Imperial College London, by EPSRC through grants EP/I00677X/1, EP/I006761/1, and EP/L000407/1, by NERC grants NE/K008951/1 and NE/K006789/1, by the U.S. National Science Foundation through grants 0811457 and 0926687, by the U.S. Army through contract W911NF-10-1-000, and by a HiPEAC collaboration grant. The authors would like to thank Dr. Carlo Bertolli, Dr. Lawrence Mitchell, and Dr. Francis Russell for their invaluable suggestions and their contribution to the Firedrake project.
Author's addresses: Fabio Luporini & Paul H. J. Kelly, Department of Computing, Imperial College London; David A. Ham, Department of Computing and Department of Mathematics, Imperial College London;

DOI:http://dx.doi.org/10.1145/0000000.0000000

```
for (int i=0; i<IP; ++i)
{
  double F = 0.0;
  for (int r=0; r<T; ++r)
  {
    F += (w[r]*X[i][r]);
  }
  for (int j  = 0; j<T; ++j)
  {
    for (int k=0; k<T; ++k)
    {
      A[j][k] += B[i][j]*C[i][k] + (B[i][j]*D[i][k])*F;
    }
  }
}
```

Fig. 1: Original code

nels evaluating element matrices become more computationally expensive, which impacts significantly the run-time of the overall computation.

Achieving high performance is non-trivial due to the complexity of the mathematical expressions involved in the numerical integration and because of the small sizes of loops and accessed arrays. In [Olgaard and Wells 2010], [Kirby et al. 2005] and [Russell and Kelly 2013], it is shown how automated code generation allows the introduction of powerful optimizations, which a user cannot be expected to write "by hand", as well as the exploration of non-standard integration techniques, based, for instance, on symbolic manipulation. In [**?**] we made one step forward by showing that different problems require distinct set of transformations if close-to-peak performance needs to be obtained, and that low-level, domain-aware code transformations, which are not supported by available compilers, are essential to maximize instruction-level parallelism and register locality.

2.3. Summary of Low-level Optimizations and Results for Quadrature Representation

To neatly distinguish the contributions of this paper from those in [**?**], we summarize in this section the results of our previous work on automated code transformation for quadrature representation.

...TODO...

3. EXPRESSION RE-WRITING

3.1. Re-writing Rules

3.2. Avoiding Operations on Zeros

4. CODE SPECIALIZATION

4.1. Standard Compiler Transformations

4.2. Precomputation of Invariant Terms

4.3. Exposing Linear Alegbra Operations

4.4. Model-driven Autotuning

5. PERFORMANCE EVALUATION

5.1. Experimental Setup

5.2. Results for Forms of Increasing Complexity

6. CONCLUSIONS

REFERENCES

Robert C. Kirby, Matthew Knepley, Anders Logg, and L. Ridgway Scott. 2005. Optimizing the Evaluation of Finite Element Matrices. *SIAM J. Sci. Comput.* 27, 3 (Oct. 2005), 741–758. DOI:http://dx.doi.org/10.1137/040607824

Anders Logg, Kent-Andre Mardal, Garth N. Wells, and others. 2012. *Automated Solution of Differential Equations by the Finite Element Method*. Springer. DOI:http://dx.doi.org/10.1007/978-3-642-23099-8

Kristian B. Olgaard and Garth N. Wells. 2010. Optimizations for quadrature representations of finite element tensors through automated code generation. *ACM Trans. Math. Softw.* 37, 1, Article 8 (Jan. 2010), 23 pages. DOI:http://dx.doi.org/10.1145/1644001.1644009

Francis P. Russell and Paul H. J. Kelly. 2013. Optimized Code Generation for Finite Element Local Assembly Using Symbolic Manipulation. *ACM Trans. Math. Software* 39, 4 (2013).