Imperial College London
Department of Computing

Bho

Fabio Luporini

September 2014

Supervised by: Dr. David A. Ham, Prof. Paul H. J. Kelly

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing
of Imperial College London
and the Diploma of Imperial College London

Declaration

I herewith certify that all material in this dissertation which is not my own work has been properly acknowledged.

Fabio Luporini

Abstract

TODO

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

In many fields, such as computational fluid dynamics, computational electromagnetics and structural mechanics, phenomena are modelled by partial differential equations (PDEs). Unstructured meshes, which allow an accurate representation of complex geometries, are often used to discretize their computational domain. Numerical techniques, like the finite volume method and the finite element method, approximate the solution of a PDE by applying suitable numerical operations, or kernels, to the various entities of the unstructured mesh, such as edges, vertices, or cells. On standard clusters of multicores, typically, a kernel is executed sequentially by a thread, while parallelism is achieved by partitioning the mesh and assigning each partition to a different node or thread. Such an execution model, with minor variations, is adopted, for instance, in **?**, **?**, **?**, **?**, which are examples of frameworks specifically thought for writing numerical methods for PDEs.

The time required to execute these unstructured-mesh-based applications is a fundamental issue. An equation domain needs to be discretized into an extremely large number of cells to obtain a satisfactory approximation of the solution, possibly of the order of trillions (e.g. **?**), so applying numerical kernels all over the mesh is expensive. For example, it is well-established that mesh resolution is crucial in the accuracy of numerical weather forecasts; however, operational centers have a strict time limit in which to produce a forecast - 60 minutes in the case of the UK Met Office - so, executing computation- and memory-efficient kernels has a direct scientific payoff in higher resolution, and therefore more accurate predictions. Motivated by this and analogous scenarios, this thesis studies, formalizes, and implements a number of code transformations to improve the performance of real-world scientific applications using numerical methods over unstructured meshes.

1.2 Contributions

Besides developing novel compilers theory, contributions of this thesis come in the form of tools that have been, and currently are, used to accelerate scientific computations, namely:

- **A run-time library, which can be regarded as a prototype compiler, capable of optimizing sequences of non-affine loops by fusion and automatic parallelization. The library has been used to speed up a real application for tsunami simulations as well as other representative benchmarks.**

  One limiting factor to the performance of unstructured mesh applications is imposed by the need for indirect memory accesses (e.g. `A[B[i]]`) to read and write data associated with the various entities of the discretized domain. For example, when executing a kernel that numerically computes an integral over a cell, which is common in a finite element method, it may be necessary to read the coordinates of the adjacent vertices; this is achieved by using a suitable indirection array, often referred to as "map", that connects cells to vertices. The problem with indirect memory accesses is that they break several hardware and compiler optimizations, including prefetching and standard loop blocking (or tiling). A first contribution of this thesis is the formalization and development of a technique, called "generalized sparse tiling", that aims at increasing data locality by fusing loops in which indirections are used. The challenges are 1) to determine if and how a sequence of loops can be fused, and 2) doing it efficiently, since the fusability analysis must be performed at run-time, once the maps are available (i.e. once the mesh topology has been read from disk).

- **A fully-operating compiler, named COFFEE[1], which optimizes numerical kernels solving partial differential equations**

[1]COFFEE stands for COmpiler For FinitE Element local assembly.

using the finite element method. The compiler is integrated with the Firedrake system (?).

The second contribution, in the context of the finite element method, is about optimizing the computation of so called local element matrices, which can be responsible for a significant fraction of the overall computation run-time. This is a well-known problem, and several studies can be found in the literature, among which **?**, **?**, **?**, **?**. With respect to these studies, we propose a novel set of composable, model-aware code transformations explicitly targeting, for the first time, instruction-level parallelism - with emphasis on SIMD vectorization - and register locality. These transformations are automated in COFFEE.

1.3 Thesis Statement

1.4 Motivation

1.5 Contributions

1.6 Statement of Originality

1.7 Dissemination

1.8 Thesis Outline

Chapter 2

Background

2.1 bho

Chapter 3

Sparse Tiling

3.1 bho

Chapter 4

COFFEE: an Optimizing Compiler for Finite Element Local Assembly

4.1 Motivations

4.2 Preliminaries

4.2.1 Overview of the Finite Element Method

4.2.2 From Math to Actual Code

4.3 On the Space of Possible Code Optimizations

4.3.1 Outline of COFFEE

4.4 Expression Rewriting

4.5 Code Specialization

4.6 Composing Optimizations

4.7 Design and Implementation of COFFEE

4.7.1 Input ad Output: the Integration with Firedrake

4.7.2 Structure

4.7.3 Conveying Domain-Specific Knowledge

4.8 Performance Analysis

4.8.1 Contribution of Basic Optimizations

4.8.2 Evaluation in Forms of Increasing Complexity

4.9 Generalizing to Local Vector Assembly

4.10 Conclusions

Chapter 5

Conclusion

5.1 bho