PhD Progression Review: Autotuning and Skeleton-aware Compilation

Chris Cummins
September 2015

Abstract

I present a research proposal to develop an optimising compiler specifically tailored for algorithmic skeletons. Such a compiler would be capable of performing optimisations which are not possible by library-level skeleton frameworks: optimisations across skeleton calls (e.g. to re-order the execution of nested skeletons), and across muscle functions (e.g. to perform load balancing of pipeline stages). This document briefly outlines the proposed project, and summarises the findings from the MSc phase of my CDT.

1 Introduction

In my initial CDT application, I postulated that in order to address the under-utilisation of parallel resources that is a symptom of the modern hardware/software ecosystem, the goal should be to move towards *automated* parallelism. My MSc thesis worked towards this goal by developing a system for adaptive tuning of patterns (algorithmic skeletons) for high level parallel programming. Skeletons provide abstractions that free developers from the concerns of managing parallel resources. The need for adaptive tuning of skeletons was demonstrated by the significant performance improvements which were achieved through autotuning of only a single two-dimensional parameter space for stencil skeletons. I demonstrated a median geometric speedup of 3.79× across benchmarks from both real world and synthetically generated sources. My intention over the coming months is to broaden the scope of this work to consider the interaction between algorithmic skeletons and optimising compilers.

2 MSc Phase Review

My MSc thesis introduced *OmniTune*, a framework for runtime autotuning of parameters. The effectiveness of this approach was demonstrated using the workgroup size of stencil skeletons for CPUs and GPUs. Machine learning models trained using data collected from synthetically generated benchmarks were evaluated for prediction quality in this two-dimensional parameter space.

The plan for the project consisted of three sequential phases: identifying a profitable optimisation space, probing the space to gather empirical performance data, and developing a system to exploit this optimisation space. In reality, there was a great overlap between the progress of each phase: identifying the optimisation space was achieved using empirical performance data, and in turn gathering performance data required the partial development of the autotuning system. Broadly, the stencil optimisations space had been identified by early March, the collection of empirical performance data began in mid April, and the development of the final OmniTune autotuning system began in mid May. For future work, a methodology based upon shorter, iterative cycles of testing, implementing, and evaluating should be used. This should also be coupled with regular write-ups of current findings and results, in addition to the logbook I kept this year. Another insight which can be transferred to future work is the importance of clear, testable hypotheses, which should be identified during early planning stages.

3 PhD Phase

While the tuning of parameter values demonstrably has a significant impact on the performance of skeleton programs, there is great scope for optimisation at the compilation stage. The complexity of parallel programs restricts the number of optimisations which compilers may apply, and furthermore there are higher-level restructurings and implementation choices which could be made by compilers equipped with an understanding of the semantics of skeleton operations.

3.1 Objectives

The objective of compiler research for algorithmic skeletons would be to enable "skeleton-aware compilation". This can be considered both as the enabling of existing compiler optimisations to algorithmic skeletons which cannot currently be applied due to limitations in the static analysis of parallel programs, and the development of novel optimisations which are specific to algorithmic skeletons. In both cases, skeleton-aware compilation will be able to provide measurable performance improvements of programs compiled using the existing state of the art in optimising compilers.

3.2 Methodology

This research project will begin with an exposition of the relevant literature to help enumerate the range of possible optimisations which could be applied, and to identify what has already been achieved. When identifying a range of optimisations which could be developed, low cost proof-of-concept tests can be performed for each by manually performing an optimisation by hand for a specific benchmark and gathering empirical performance data. If the optimisation leads to measurable performance improvements, a transformation pass in a relevant compiler (e.g. LLVM) can be implemented to automatically apply this optimisation. Test programs may be selected from existing benchmark suites (e.g. the Intel Thread Building Blocks port of PARSEC), or by using synthetically generated programs, perhaps by extending the benchmark generator developed for my MSc project. In case of optimisations for which the potential benefit cannot statically be determined (e.g. load balancing between stages of a pipeline), the OmniTune framework may be extended to provide autotuning capabilities for the space of possible optimisations.

4 Summary

In my first year I have demonstrated the importance of adaptive tuning for maximising the performance of high level parallel patterns. I intend to continue improving the performance of high level parallel patterns, but taking an approach from the level of the compiler, with the aim of developing skeleton-aware compilation. The skills I acquired this year in skeletal programming, heterogeneous parallelism, machine learning, and autotuning will all be required for achieving this aim. The success of the project can be quantifiably measured in terms of performance improvements to real world skeleton programs.