Development of a linear algebra library to support loop and data transformations in an optimising compiler

Michael Koutroumpas

Master of Science
School of Informatics
University of Edinburgh
2005
Abstract

This paper is concerned with the implementation of a new program optimization technique. More specifically, it describes an implementation of non-unimodular transformations with the ultimate goal of exploiting parallelism in sequential code. This includes a description of the problems that are solved in contrast to previous approaches, the desired outcomes, as well as methods to be used for evaluating results. In the end, we propose a timeline for the actual realization of the proposal. Additional ideas and thoughts about possible applications are also included.
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Michael Koutroumpas)
# Table of Contents

1 Introduction 1

2 Proposal 2

2.1 Preamble 2

2.2 Research Proposal 2

2.3 Part I: The linear algebra library 3

2.3.1 Description 3

2.3.2 Testing 3

2.3.3 Benchmarking 4

2.4 Part II: Integration 4

2.4.1 Description 4

2.4.2 Testing 4

2.4.3 Benchmarking 5

2.5 Timetable 6

2.6 Importance 7

2.6.1 Possible shortcomings 7

2.6.2 Conclusion 7

Bibliography 8
Chapter 1

Introduction

High performance computing is becoming more and more of an issue. This is mostly due to a sudden rise in computational demand in both scientific, commercial and consumer oriented fields. The huge computational needs of bio-informatics, medical hardware, physics are only some of the examples in the academic sector. Furthermore modeling and simulation techniques for all kinds of applications are another example. In general the areas requiring more computing performance than already available are seemingly endless.

The most promising hardware setup for addressing the above computing requirements seems to be concurrent execution of software code. To that end huge amounts of legacy code, as well as new software have to be implemented in a way that takes advantage of concurrency. So far this has led to an anarchy of competing standards with no apparent winner. [Hill; McColl, 1996].

This paper proposes an automated method of approaching the problem, based on the latest research on source optimization. The idea is that instead of proposing some new way of implementing parallel execution, we should optimize existing sequential code in a way that executes concurrently. Substantial research has already been done in that area [Wolf and Lam, 1991]. Many of the latest theories, though, have yet to be implemented in an optimizing compiler. In the following section we propose an implementation that tries to address all the above issues.
Chapter 2

Proposal

2.1 Preamble

In an earlier paper we have reviewed latest research in the area of loop and data optimizations. There is a huge amount of publications regarding different approaches. The ones that we have already reviewed are source level transformations. The reason is that optimizations can be more sophisticated when applied earlier in the compilation phase. A rich amount of semantic information is lost during later stages of a compilation. Of the existing approaches the unimodular framework is the established one for unifying specific loop optimizations. There exists a free \(^1\) implementation.

The existing methods used in compilers for source code optimization have several shortcoming, though. More specifically they are limited to a reduced set of transformations, not including important ones like loop distribution and loop fusion. Furthermore it is difficult to be applied in non-perfectly nested loops. It therefore ignores a rich set of potential optimizations. Further analysis of the existing approaches is in the previous review paper as well as the Bacon D. et al. (1994).

2.2 Research Proposal

In an attempt to address the shortcomings of existing methods we are proposing an implementation based on a non-unimodular framework [O’Boyle, 2002]. This way we

\(^1\)the SUIF parallelizing compiler (http://suif.stanford.edu)
can take advantage of new optimization techniques that so far are ignored by compilers. The implementation comprises of two parts. The first is the implementation of a supporting linear algebra library and the second is the integration of that library to the SCALE research compiler.

The SCALE compiler was chosen because it is a research compiler written in Java. This means that there is a solid, highly documented API, specifically designed for new extensions. Therefore an extension to the compiler can be the non-unimodular loop and data transformation algorithm.

2.3 Part I: The linear algebra library

2.3.1 Description

The first phase of the project is the development of a linear algebra library. The linear algebra implemented by this library will include the extended matrix operations as described by O’Boyle, M. and Knijnenburg, P. (2002). In particular, it should support the pseudo-inverse matrices and the related operations. This will include a sane way of representing extended matrices with a well-defined syntax.

The programming language will be Java. The main reason is because the SCALE compiler is written in Java as well. Therefore the library will directly communicate with the SCALE API during the integration phase. Furthermore the object oriented design which is mandated by Java will help produce a well engineered version of the library. It will also be extensible to allow any future modifications. Last but not least, a properly defined class hierarchy will help debugging in addition to promoting reusability.

2.3.2 Testing

Testing is essential before proceeding to the integration phase. The functionality of the library has to be thoroughly verified in order to isolate all implementation specific problems (in contrast to integration specific ones).

The functionality that will be implemented at that point is the extended linear algebra

---

2the SCALE compiler (http://www-ali.cs.umass.edu/Scale)
3The Java programming language (java.sun.com)
transformations. Therefore a series of regression tests should be applied to ensure that we get proper results for predefined inputs.

2.3.3 Benchmarking

Performance is a vital part of compilers. Although Java is not the most CPU efficient language, every care will be taken in order to ensure efficiency of the algorithm per se. This includes proper profiling of the code in order to find bottlenecks. Moreover, the library will be stress-tested with an amount of input similar to what would be expected during a real life compilation.

2.4 Part II: Integration

2.4.1 Description

Scale is a Java based research compiler with front-ends for C, Fortran and Java, and a number of different back-ends for various architectures, as well as a C backend for debugging purposes. During its optimisation phase it operatates several passes on its intermediate representation. It lacks, however, a linear algebra framework to represent and transform data and loops.

The most challenging part of the project is the integration of the library to the SCALE compiler. It involves thoroughly studying the intrinsics of the compiler API and carefully crafting a non-intrusive integration plan. By non-intrusive we mean that it should not conflict with the rest of the functionality of the compiler. The resulting code should function in exactly the same way.

2.4.2 Testing

SCALE is able to produce a C output as a backend code. We can take this into our advantage: compiling unoptimized C code should clearly indicate from its output whether the transformation worked as intended or not. The testing therefore will include a series of C input files, which would go through some example transformation passes using the library.
2.4.3 Benchmarking

Using the C backend feature of scale we can conveniently compare the performance gain of the transformations. Compiling the two source files with the same compiler switches should enable proper comparison of execution times.
2.5 Timetable

An estimated timetable can be shown in the form of a Gantt chart below:

<table>
<thead>
<tr>
<th>Task</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear algebra library</td>
<td>May 25, 2005</td>
<td></td>
</tr>
<tr>
<td>Define specifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library released</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCALE integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interegrate library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Benchmarking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project completion</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It can be summarized as follows:

- Project start at May 25, 2005
- Linear algebra library (duration 23 days)
- Scale integration (duration 38 days)
- Report writing (duration 15 days)
- Estimated release at August 9, 2005

It should be noted that the above tasks are to be implemented sequentially. That is, each one assumes that all previous tasks are completed.
2.6 Importance

A successful implementation is very important. At first, it will help validate or improve the theoretical non-unimodular transformation model. A compiler with an integrated transformation framework will allow massive testing of the existing theory in contrast to the manual verifications that were used so far.

Furthermore, a successful compiler integration will provide a basis for iterative improvements that will give a new momentum to the existing optimization techniques. More specifically, it will allow for comparing relative performances and adjust different transformations in order to achieve maximum results.

As far as other applications are concerned, we can obviously include all sectors that benefit from high performance computing. In other words, all scientific and commercial sectors can benefit from this research, as they would by any other improvement in computational efficiency. The most relevant in the short term are demanding applications like digital signal processing, modelling fluid dynamics and bio-informatics research.

2.6.1 Possible shortcomings

Since this proposal is based on latest compiler research there is a certain risk attached to it. It should be noted, though, that this field has been very vastly documented. If the actual implementation reveals any deviation in practice from what was expected, this can always be a basis for further refinement and progress.

2.6.2 Conclusion

To sum up, this paper proposes an implementation of the non-unimodular framework that integrates loop and data transformations. The expected outcome is twofold: i) to validate the non-unimodular framework and prove its usefulness and ii) to improve existing compiler technology by allowing more aggressive source code optimizations for concurrent execution.
Bibliography


