SPECULATIVE LOOP PARALLELIZATION

Johannes Doerfert

Bachelor Thesis

Compiler Design Lab
Department of Computer Science and Mathematics
Saarland University

Supervisor: Prof. Dr. Sebastian Hack
Second reader: ... TODO ...
Advisors: Clemens Hammacher and Kevin Streit

April 2012

UNIVERSITÄT
DES
SAARLANDES

Declaration of Authorship

I, Johannes Doerfert, declare that this thesis titled, 'SPECULATIVE LOOP PARALLELIZATION' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Write a funny quote here."

If the quote is taken from someone, their name goes here

SAARLAND UNIVERSITY

Abstract

Compiler Design Lab
Department of Computer Science and Mathematics

Bachelor of Science

by Johannes Doerfert

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Content

List of Figures

List of Tables

Abbreviations

AA	**A**lias **A**nalysis
LLVM	**L**ow **L**evel **V**irtual **M**achine
SCoP	**S**tatic **Co**ntrol **P**art
SPolly	**S**peculative **P**olly

For/Dedicated to/To my...

Chapter 1

Introduction

Chapter 2

Background Theory

SPolly is based on several frameworks and tools which will be now introduced. As this is not part of the actual work, this section will just provide overviews for each one. To get a deeper understanding it is necesary to consult the further readings.

TODO this section was just copied TODO
TODO it is also incomplete TODO

2.1 LLVM - The Low Level Virtual Machine

The Low Level Virtual Machine is a compiler infrastructure designed to optimize during compiletime, linktime and runtime. Originally designed for C/C++, many other frontends for a variety of languages exist by now. The source is translated into an intermediate representation (LLVM-IR). The LLVM-IR is a type-safe, static single assignment based language designed for low-level operations. It is capable of representing high-level structures in a flexible way. Due to the fact that LLVM is built in a modular way and can be extended easily, most of the state of the art analysis and optimization techniques are implemented and shipped with LLVM. Plenty of other extensions, e.g., Polly, can be added by hand. Another point for LLVM is the included just-in-time compiler for runtime analysis and optimization mentioned in the context of Sambamba in Section **??** in more detail.

Further Reading

- A Compilation Framework for Lifelong Program Analysis & Transformation [1]
- `http://www.llvm.org`

2.2 The Polyhedral Model

The polyhedral model is a mathematical description of a set of (integer) points defined by a finite system of affine functions. If this set is bounded we will talk about polytopes whereas polyhedra are built the same way but consist of an unbounded point set. Polytopes can be used to describe the index space of a loop nest with affine loop bounds. The dimension of this space is determined by the depth of the loop nest. Each occurring iteration vector, which is a vector containing all surrounding induction variables, corresponds to a point in the index space. In addition to the index space the polytope model consists of a set of vectors within this index space. Each vector corresponds to a data dependency between two iterations.

Figure **??** shows such a loop represantation in the polytope model. The iterations are shown as points and the dependencies between two iterations are represented as arrows. For simplicity the initialization process of the array is skipped.

Further Reading

- Loop Parallelization in the Polytope Model [3]
- PoCC - The Polyhedral Compiler Collection [4]

2.3 Polly - A Polyhedral Optimizer For LLVM

As parallelism and data locality becomes more and more important, there is heavy research and development in this area. Based on the polyhedral model, Polly aims for an automatic optimization of LLVM bitcode with no need for annotations or other user interactions. Besides the support of external optimizers, there is a state of the art polyhedral library included, as well as support for SIMD and OpenMP code generation[7].

To gain a deeper understanding, the three main parts of Polly, represented as (solid) translations in Figure **??**, are explained in particular now.

Further Reading

- Polly - Polyhedral optimization in LLVM [5]
- Enabling Polyhedral Optimizations in LLVM [6]

- A Framework for Automatic OpenMP Code Generation [7]
- `http://polly.llvm.org`

2.4 Sambamba - A Framework For Adaptive Program Optimization

As an extension for LLVM the *Sambamba* compiler framework is designed to allow runtime analyses and (speculative) optimization. Furthermore these optimization can create and refine runtime profiles which are used to recalibrate and specialize the (speculative) execution. Method versioning allows conservative and speculative versions of a method to be stored and switched during runtime. Written in a completely modular way, Sambamba extensions consist of a static part (compiletime) and a dynamic one (runtime). Both extension parts can use Sambamba to store information, collected at the corresponding time, accessible for the dynamic part at runtime. Profiling combined with the method versioning system allows runtime interactions to explore more parallelism or minimize the overhead in case of misspeculation.

Further Reading

- Sambamba: A Runtime System for Online Adaptive Parallelization [9]
- `http://www.sambamba.org`

Chapter 3

Implementation

TODO polly – regionspeculation diagram

3.1 SPolly

3.1.1 Speculative Extension For Polly

3.1.2 Sambama Compile Time Module

3.1.3 Sambama Runtime Module

3.2 Profiling For Sambamba

Chapter 4

Evaluation

To evaluate the implementation, the SPEC2000 benchmark suite —— TODO cite and the Polybench/C 3.2 —— TODO cite
suite were used to meassure correctness, applicability and speedup.

4.1 The Environment

Two computers were involved in the evaluation. The first one was a general purpose machine running Arch linux with an *Intel(R) Core(TM) i5 CPU M 560 @ 2.67GHz* and 6GB RAM. Parallel versions could use up to four simultaneous running threads. The second one ... TODO ... As the compile time evaluation is machine independent, it was performed on the general purpose machine only. Contrary, the runtime evaluation has been performed twice, once on each machine.

The work and thus the evaluation is based on an LLVM 3.0 build with enabled assertions and disabled optimization. All source files have been converted by clang to LLVM-IR files, optimized by opt and ... TODO linked TODO

TODO picture of the chain

4.2 Compile Time Evaluation

The main part of the compile time evaluation aims to get quantitative results on the transformable code regions. These results correspond with the applicability of this work, as they both outline how many regions may be transformed to increase the performance and which work is needed to be done to raise this number. As mentioned earlier this part is machine independent if only these quantitative results are regarded, but there is one case where compile time transformation can improve the program with no need of speculation at all. These cases are explained and evaluated separately (see section 4.2.3).

4.2.1 Preperation

TODO -basicaa -indvars -mem2reg -polly-independent -polly-region-simplify -polly-prepare

4.2.2 Quantitative Results

4.2.2.1 SPEC2000

TODO why not all spec2000 benchmarks ?
TODO

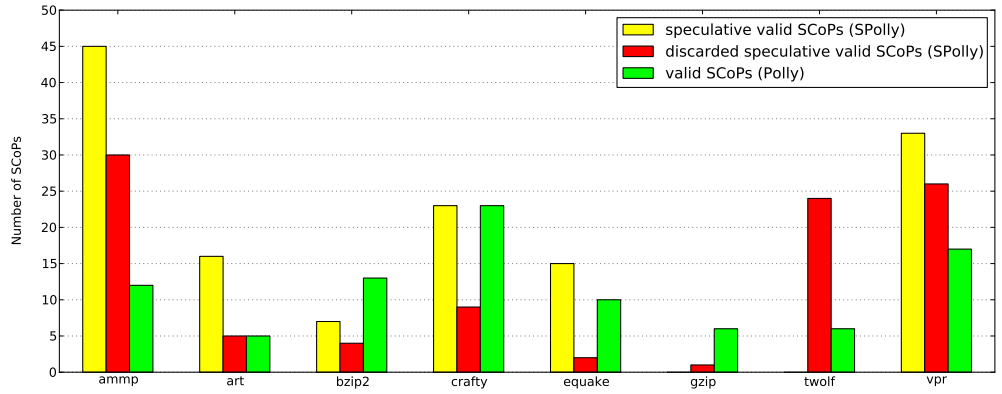

FIGURE 4.1: Numbers of valid and speculative valid SCoPs

4.2.2.2 Polybench 3.2

4.2.2.3 Available tests

Alias tests

Invariant tests

4.2.3 Sound Transformations

As described earlier, region speculation collects violations within a SCoP and can introduce tests for some of them. There are cases when these tests will suffice to get a sound result, thus there is no need for a runtime system at all. Although this hold in respect to the soundness of a program, this does not mean performance will rise when

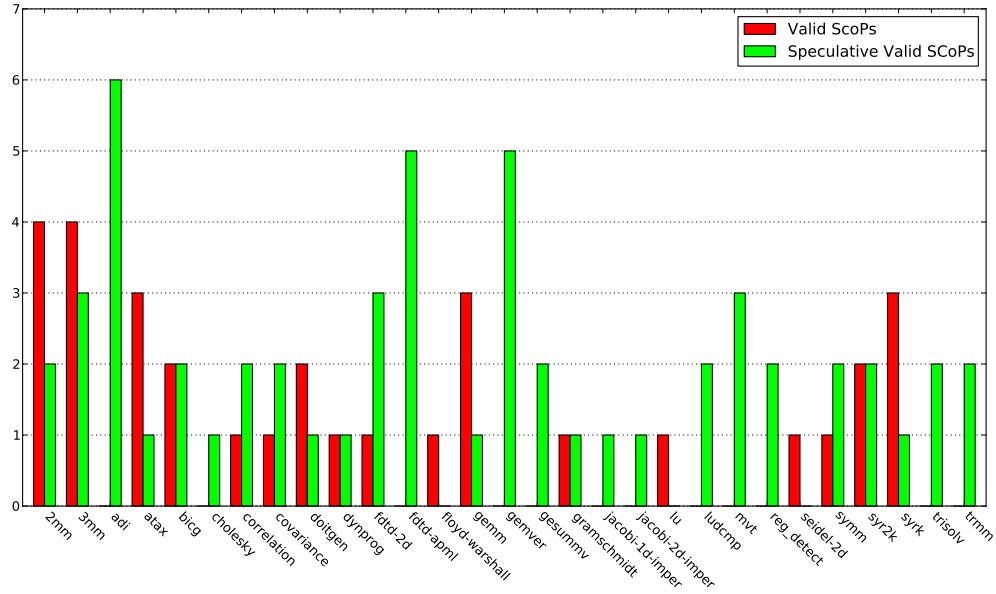

FIGURE 4.2: Numbers of valid and speculative valid SCoPs

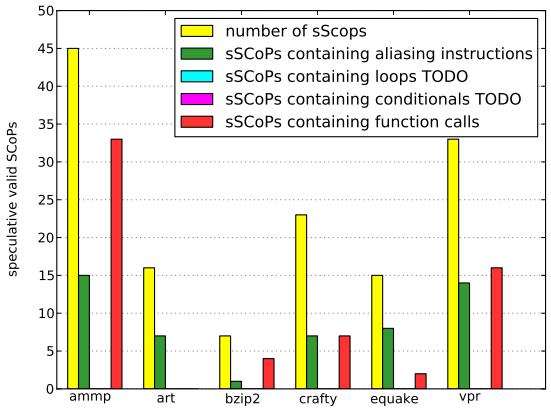

FIGURE 4.3: Details about speculative valid SCoPs

these transformations are used.
TODO scores – heuristic / statistics

4.3 Runtime Evaluation

4.4 Problems

During the work with LLVM 3.0 and a corresponding version of Polly a few problems occurred. Some of them could not be reproduced in newer versions they were just be tackled with tentative fixes, as they will be resolved as soon as Sambamba and SPolly

will be ported to a newer version. Others, which could be reproduced in the current trunk versions, have been reported and listed in figure 4.1. All bugs were reported with a minimal test case and a detailed description why they occur.

Table 4.1: Reported bugs

ID	Description	Status	Patch provided	Component
12426	Wrong argument mapping in OpenMP subfunctions	RESOLVED FIXED	yes	Polly
12427	Invariant instruction use in OpenMP subfunctions	NEW	yes	Polly
12428	PHInode use in OpenMP subfunctions	NEW	no	Polly
TODO	add the others to bugzilla			
TODO	PollybenchC2 gemver , 2mm			

Chapter 5

Discussion

5.1 Quantitative Results

5.2 Qualitative Results

5.3 Interpretation

Chapter 6

Conclusion

Appendix A

Matrix Multiplication Example

Modified version of the matrix multiplication example presented on the Polly website. Changes include:

- arrays are given as parameters
- sum up the result instead of printing it

```
#define N 1536

float A[N][N], B[N][N], C[N][N];

void init_arrays(float A[N][N], float B[N][N]) {
    int i, j;

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            A[i][j] = (1+(i*j)%1024)/2.0;
            B[i][j] = (1+(i*j)%1024)/2.0;
        }
    }
}

void multiply_arrays(float A[N][N], float B[N][N], float C[N][N]) {
    int i, j, k;

    for(i=0; i<N; i++) {
        for(j=0; j<N; j++) {
            C[i][j] = 0;
            for(k=0; k<N; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
    }
}

double sum_array(float C[N][N]) {
    double sum = 0.0;
```

```
    int i, j;

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
          sum += C[i][j];
        }
    }

    return sum;
}

int main() {
    double sum_no_alias, sum_alias;

    init_arrays(A, B);
    multiply_arrays(A, B, C);
    sum_no_alias = sum_array(C);

    init_arrays(A, A);
    multiply_arrays(A, A, C);
    sum_alias = sum_array(C);

    return sum_no_alias != sum_alias;
}
```

Bibliography

[1] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[2] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation, 2004. URL http://www.llvm.org.

[3] Christian Lengauer. Loop parallelization in the polytope model. In *CONCUR '93, Lecture Notes in Computer Science 715*, pages 398–416. Springer-Verlag, 1993.

[4] Louis-Noël Pouchet. PoCC - The Polyhedral Compiler Collection, 2009. URL http://www.cse.ohio-state.edu/~pouchet/software/pocc/.

[5] Tobias Grosser, Hongbin Zheng, Raghesh A, Andreas Simbürger, Armin Grösslinger, and Louis-Noël Pouchet. Polly - polyhedral optimization in llvm. In *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11)*, Chamonix, France, April 2011.

[6] Tobias Grosser. *Enabling Polyhedral Optimizations in LLVM*. Diploma thesis, University of Passau, April 2011. URL http://polly.llvm.org/publications/grosser-diploma-thesis.pdf.

[7] A. RAGHESH. *A Framework for Automatic OpenMP Code Generation*. PhD thesis, INDIAN INSTITUTE OF TECHNOLOGY, MADRAS, 2011.

[8] Tobias Grosser. Polly - polyhedral optimization in llvm, 2011. URL http://polly.llvm.org.

[9] Kevin Streit, Clemens Hammacher, Andreas Zeller, and Sebastian Hack. Sambamba: A runtime system for online adaptive parallelization. In Michael F. P. O'Boyle, editor, *CC*, volume 7210 of *Lecture Notes in Computer Science*, pages 240–243. Springer, 2012. ISBN 978-3-642-28651-3.

[10] Kevin Streit, Clemens Hammacher, Andreas Zeller, and Sebastian Hack. Sambamba: A runtime system for online adaptive parallelization, 2012. URL http://www.sambamba.org.