Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

HEAPLab Project Report

Performance Estimation for TAFFO via LLVM-MCA

Author:
Marco Prosdocimi

Supervisors:
Stefano Cherubin
Daniele Cattaneo

September 13, 2020

Abstract

This project aims to use LLVM-MCA to compare the fixed-point code produced by TAFFO with the original floating-point code for all loops in a given program to obtain a static prediction of the possible speedups gained by TAFFO.
LLVM-MCA is a tool that simulates the inner behavior of the CPU to estimate the performance of a machine code snippet.
TAFFO is an autotuning framework, based on LLVM 8, which tries to replace floating-point operations with fixed-point operations as much as possible.

1 Introduction

The purpose of this project is to build a tool that, by using the LLVM compiler infrastructure, finds loop bodies in a C program, compiles them with and without TAFFO, and uses LLVM-MCA to check if TAFFO is able to improve the execution time. The tool uses LLVM-MCA in order to have a static prediction of the possible speedup given by TAFFO. Up to now, possible speedups were measured only at run time.

2 Design and Implementation

The tool consists of two different parts: the Loop Finder and the LLVM-MCA analysis.

2.1 Loop Finder

The Loop Finder uses LLVM APIs to search loops in the code and then add an Assembly inline comment then the loop's header and to the exit block. The Assembly comments are ”LLVM-MCA-BEGIN” and ”LLVM-MCA-END”; LLVM-MCA uses them to identify the region of code to analyze. In the event of nested loops, the Loop Finder marks only the innermost ones. Before running Loop Finder, loops are transformed into their canonical form: the latter has a pre-header and a single exit block. To realize that, I used the Loop Simplify LLVM pass that runs before Loop Finder. Unfortunately, even by using Loop Simplify, I was not always able to transform loops into a form

that allows them to have a single exit block.
In the current version of the tool, these kinds of loops are skipped.

2.2 LLVM-MCA analysis

LLVM-MCA is a performance analysis tool that uses information available in LLVM (e.g. scheduling models) to statically measure the performance of machine code in a specific CPU. I modified the main procedure of LLVM-MCA so that it can show the total machine cycle count for all marked regions. The original version of LLVM-MCA was designed to print different analyses for all different regions marked in the code. Having a general estimate of machine cycles, instead of several evaluations for individual regions, will make it easier to evaluate a possible speedup. In the file "llvm-mca-mod.c" I underlined the original LLVM-MCA code and the few lines that I have changed.

2.3 Outline of the operation performed by the tool

The execution of the tool follows these main steps:

1. By using clang, it compiles the code and emits two LLVM IR files.
2. To generate the TAFFO IR file, it runs all TAFFO passes, and then the Loop Finder pass.
3. To generate the non TAFFO IR file, it executes only the Loop Finder pass.
4. The two IR files are compiled using LLC.
5. The modified version of LLVM-MCA is used to compute the cycle count on both of the Assembly files
6. For both of the files, the tool prints the results of the analysis.

3 Experimental procedure and evaluation

To validate the operation of the tool, I ran several benchmarks. I performed the analysis using the tool on an AMD A8-6600K x86 64 bit CPU, and I compared the speedup statically estimated by the tool with the real speedup measured at run time.

3.1 Polybench benchmarks

PolyBench is a collection of benchmarks used to validate experimental compilers. It consists of 30 different tests which have a single loop that dominates all the execution times.
By looking at the results in table 2, it can be observed that the predictions are not very accurate. This behavior could be due to the inability of LLVM-MCA to predict cache behavior. In the future, it will be necessary to repeat the tests using machines without cache, such as some microcontrollers.

3.2 AxBench benchmarks

AxBench is a suite of benchmarks use to evaluate applications for approximate computing. For these benchmarks, instead of using the shell script tool, I wrote a Makefile that compiles the code with and without TAFFO and generates two different assembly files. Then I used the modified version of LLVM-MCA on both Assembly files to estimate the machine cycles. Even in this case, the tool's previsions are not very accurate. The biggest problem lies in the impossibility of the current version of the tool to analyze all kinds of loops, as expressed in section 2.1.

Table 1: Comparison of the speedup predicted by the tool and the measured speedup for AxBench

Test Name	Predicted speedup	Real speedup
blackscholes	0.747	1.176
fft	3.254	0.998
jmeint	1.130	0.629
sobel	2.430	0.0639
inversek2j	2.300	0.995

4 Conclusions and future developments

The tool I created is useful if used as a new component of the TAFFO tool's chain, but it certainly needs some improvements. The challenge in managing some specific kinds of loops – those that don't have a single exit block after the execution of Loop Simplify – makes the tool not entirely usable. In the future, it will be necessary to change the "Loop Extractor" LLVM pass in order to add support to these kinds of loops that are currently skipped. This problem is presented in some AxBench; thus, not all benchmarks result are fully reliable yet. The best results when using TAFFO is achieved with machines without FPU (Floating Point Unit). In the future, it will be useful to repeat all benchmarks in this type of machine.

Table 2: Comparison of the speedup predict by the tool and the measured speedup for Polybench

Test Name	Predicted speedup	Real speedup
correlation.c	0.885	1.840
covariance.c	0.8896	1.919
2mm.c	1.164	2.153
3mm.c	1.187	2.598
atax.c	0.985	2.248
bicg.c	1.016	1.807
doitgen.c	0.809	3.157
mvt.c	0.953	2.232
gemm.c	1.126	0.822
gemver.c	1.021	1.268
gesummv.c	1.138	1.740
symm.c	0.953	1.076
syr2k.c	1.126	0.696
syrk.c	1.096	0.918
trmm.c	1.024	1.996
cholesky.c	1.027	2.925
durbin.c	0.786	0.991
gramschmidt.c	0.963	1.235
lu.c	1.026	1.277
ludcmp.c	0.861	0.826
trisolv.c	0.858	2.491
deriche.c	0.664	1.429
floyd-warshall.c	0.974	2.370
nussinov.c	0.867	1.130
adi.c	0.924	0.936
fdtd-2d.c	0.852	2.720
heat-3d.c	0.915	3.107
jacobi-1d.c	0.934	1.696
jacobi-2d.c	0.956	1.860
seidel-2d.c	0.886	3.123

5 Appendix A Machine cycles measured by using the tool

Table 3: Machine cycles estimated by using the tool for Polybench benchmarks (MC means machine cycles)

Test Name	Total number of MC	Total number of MC with TAFFO
correlation.c	11379	12851
covariance.c	7086	7910
2mm.c	18940	16261
3mm.c	20772	17490
atax.c	8660	8791
bicg.c	14088	13855
doitgen.c	11557	14282
mvt.c	10634	11150
gemm.c	13633	12105
gemver.c	10101	9892
gesummv.c	10776	9467
symm.c	12764	13384
syr2k.c	13609	12082
syrk.c	10509	9582
trmm.c	7641	7458
cholesky.c	10705	10425
durbin.c	4789	6089
gramschmidt.c	11949	12399
lu.c	10705	10425
ludcmp.c	13660	15862
trisolv.c	4884	5688
deriche.c	14938	22465
floyd-warshall.c	16627	17054
nussinov.c	8506	9807
adi.c	7968	8615
fdtd-2d.c	13527	15870
heat-3d.c	14331	15655
jacobi-1d.c	5874	6283
jacobi-2d.c	7883	8562
seidel-2d.c	8700	9811

Table 4: Machine cycles estimated by using the tool for AxBench benchmarks (MC means machine cycles)

Test Name	Total number of MC	Total number of MC with TAFFO
Blackscholes	6402	8571
fft	36718	11281
jmeint	1720	1521
sobel	27094	11149
inversek2j	102512	44571