Uppsala University

Bachelor's Thesis

Static Multi-Versioning for Efficient Prefetching

Author:
Per Ekemark

Supervisor:
Alexandra Jimborean

Examiner:
Olle Gällmo

Reviewer:
David Black-Schaffer

A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Computer Science

in the

Research Group Name
Department of Information Technology

May 2014

UPPSALA UNIVERSITY

Abstract

Faculty for Science and Technology
Department of Information Technology

Bachelor of Computer Science

Static Multi-Versioning for Efficient Prefetching

by Per Ekemark

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...

Contents

List of Figures

List of Tables

Abbreviations

CFG	Control Flow Graph
DAE	Decoupled Access-Execute

Physical Constants

Speed of Light	c	$=$	$2.997\ 924\ 58 \times 10^{8}$ ms^{-S} (exact)

Symbols

a	distance	m
P	power	W (Js^{-1})
ω	angular frequency	rads^{-1}

Chapter 1

Introduction

General introduction...

1.1 Background

- Decreased power consumption through DVFS
  - Power equation
  - Breakdown of Dennard scaling
- Decoupled access/execute
  - Finding tasks suitable for optimisation
  - Breaking them down in small enough chunks
  - Creating the access version

1.2 Approach

- Multiple access versions
  The idea is to generate more than one access version. These are to be generated from different rule sets. The generated access versions are evaluated in a runtime system in order to determine which version that produces the best results for a specific task. That version is selected to be used for that task while others may use different access versions if they prove to be better.

Chapter 2

Related Work

- Fix the code [1]
- Towards More Efficient Execution [2]
- Polyhedral model and affine code
  (Not strictly required for this thesis.)

Chapter 3

Methodology

A few introductory words...

3.1 Access phase generation

When generating the access phase there are three goals to keep in mind, inserting prefetches at suitable locations, preserving the Control Flow Graph (CFG), including all instructions it depends on, and removing all side effects. Assuming that a task suitable for Decoupled Access-Execute (DAE) has been found [possible reference to prev. work] it must first be copied in order to create separate access and execute phases to be run sequentially. Once this has been done the instructions that are required to be in the access phase are identified and prefetches are inserted. Other instructions are removed.

3.1.1 Control Flow Graph skeleton

To find out which instructions that must be kept in order to leave the CFG in a working state following steps are taken:

1. Find and select all instructions that directly defines the CFG.
2. Recursively find and select the instructions that produces a value that previously selected instructions requires in order to execute.
3. For every load that is found all stores that could be writing to that location are added to.

At this point all instruction that must be kept are known. If any of these instructions may cause side effects on data used outside of the access phase the task must be disqualified from DAE in order not to affect the results of the program in any way. Disqualification will occur when any function calls are selected and when selected stores touches memory that is not guaranteed to be local.

Comments:

- If a load is required for the CFG the corresponding store might not be found if its address is calculated using different data (at compile time) compared to the load. (Might be a pathological example, but can it be detected?)
- Step 3 are currently not working. It does find the stores but some of these stores are not identified as local stores. Which mean that the entire function is disqualified from prefetching. **Meanwhile, the search for stores has been turned off.**

3.1.2 Prefetches in general

There are a few policies that are always valid when generating prefetches.

- It is unnecessary to prefetch locally allocated data, in part because that data can be expected to already be in the cache, in part because the access and execute phases may not share local memory locations.
- If it has been decided that a prefetch is to be attached to a load then the instructions required to compute the pointer address must be selected to be kept in the access phase, similar to what was done in steps 2 and 3 of the identification of the CFG skeleton.
- When a prefetch instruction is inserted this should be done as soon as the pointer address is available in order to enable more aggressive dead code elimination. The prefetch itself must also be selected to be saved from deletion.

As with the CFG skeleton it is possible to run into situations where computing the pointer for a prefetch require modification to externally visible memory. If this is the case that prefetch, and its dependencies along with it, must not be selected in order to keep the correctness of the program.

3.1.3 Multiversioning

Multiple versions to the access phase are being produced by generating several instances of the access phase. While generating the instances different rules are applied to determine which of all possible prefetches that are to remain.

One set of rules stem from limiting the number of indirections there may be before a prefetch are considered to be too costly. In essence this means that when a prefetch are considered for insertion it is determined which instructions that are required to calculate the address to prefetch. In case the number of loads among these dependencies exceed a threshold the prefetch is disqualified. Several access versions can be created by varying the threshold.

TODO:

- Access version generation techniques and rules
  - Multiversioning (More rules?)
- Evaluation model
  - Power model
  - Measurement method
- Environment setup
  - LLVM 3.4
  - SPEC CPU2006

Chapter 4

Experiments

... or Evaluation

Chapter 5

Conclusion

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] A. Jimborean, K. Koukos, V. Spiliopoulos, D. Black-Schaffer, and S. Kaxiras. Fix the code. don't tweak the hardware: A new compiler approach to voltage-frequency scaling. In *GCO'14*, February 2014.

[2] K. Koukos, D. Black-Schaffer, V. Spiliopoulos, and S. Kaxiras. Towards more efficient execution: A decoupled access-execute approach. In *ICS'13*, June 2013.