PROJECT REPORT

Computer Architecture TDT4260

Trond Snekvik
trondesn@stud.ntnu.no

Christopher Benjamin Westlye
chriwes@stud.ntnu.no

Kaj Andreas Palm
kajpalm@gmail.com

Raymond Selvik
raymondd@stud.ntnu.no

Contents

Abstract

A modern day computer has a CPU that have a tremendous processing power. However, the transmission rate on the RAM is still relatively low. This can be solved by adding a memory cache between the CPU and RAM. The performance of the cache is determent by how many memory hits the CPU does in the cache. If it does not hit anyone, it has to access it from the RAM anyway. If the the cache as a prefetcher that will determine what memorylocation that is going to be fetched next, the Memory misses will be tremendously decreased.

Chapter 1

Introduction

The performance of modern day CPUs is much higher than what can be accommodated by the bus connecting the RAM. Much of the performance time is thus used to access the memory of the RAM and load it into the CPU. This delay can be reduced by adding cache memory and a prefetching scheme to the CPU. Memory fetches can then be done locally within the CPU unit, leading to increased performance. The cache fetches information from the RAM the same way as the CPU. If the prefetched data differs from what was needed, the CPU needs to access the RAM anyway. The worst case scenario is then that the cache will have no effect.

The goal of this prosject is to make a prefetcher that has a algorithm that will decrease memory misses by the CPU. The soulution presented in this report is based on pattern regnonition implemented as a two dimensional vector structure.

The cache can be designed in many ways. It can for example be direct mapped which will have fastest hit times, but a fully associative cache will have the best trade off if there are many misses. The cache assumes the program to be deterministic, but does not know how the memory is mapped in a computer program. The performance can then be increased by adding a prefetcher. A prefetcher is a program that determines what the next memory call from the CPU will be, and load the memory location into the cache. The prefetcher can be designed by that fact that a computer program is containing many arrays and repetitive function which will have a known memory location relatively to each other.

The goal of this project is to make a prefetcher that has a algorithm that will decrease memory misses by the CPU. The solution presented in this report is based on pattern recognition implemented as a two dimensional vector structure.

Chapter 2

Background

The prefetcher is based on the report on itslearning named Managing Shared Resources in Chip Multiprocessor Memory Systems by Magnus Jahre. In order to predict

Chapter 3

Prefetcher Description

The prefetcher is an attempt to improve the Delta-Correlating Prediction Tables (DCPT) approach by Granaes, Jahre and Natvig. The original DCPT algorithm is described in the "Background" section. The main weakness of this approach is (arguably) that it only bases the prefetch address on the first similar delta pattern it finds. This makes the prefetcher extremely vulnerable to alternating patterns and irregularities in general. If the access pattern makes a sudden leap in an otherwise regular pattern, not only will the prefetcher miss the irregular access, it is guaranteed to miss the next access after that when the pattern goes back to normal. This example is illustrated in Table **??**. Access number 6 breaks the pattern, and misses. Because the next fetch is based on access 6's delta, this also misses. This problem could be solved by a more democratic approach, where the most common "next delta" is used instead of the previous.

This democratic approach is what the prefetcher described in this report is meant to implement. To achieve this, the structure of the reference table needs a major change.

The prefetcher uses vectors to log observed fetch sequences, and ranks them based on how often the actual fetches correspond with the predicted sequences. This is achieved by logging combinations of three requests, thus making a link between two fetches and what we expect to be referenced after these. Every time the two requests are recognized, the third one is prefetched. If this is correct behaviour, the score of that particular sequence is raised. The next fetch is then always assumed to be the third fetch in the sequence that has been correct the most times.

The access sequences that occur the least frequent are more prone to being removed in favour of new sequences. This is done by continually raising a threshold. Sequences that are recognized and proven to be correct have their value increased. If entries fall under the threshold, they are removed when the cache is full and space is needed.

Access	Address	Delta	Fetch issued	Previous result
1	...	...	...	...
2	1000	10	1010	Hit
3	1010	10	1020	Hit
4	1020	10	1030	Hit
5	1030	10	1040	Hit
6	1050	20	1070	Miss
7	1060	10	-	Miss
8	1070	10	1080	-
9	1080	10	1090	Hit

Chapter 4

Methodology

Simulator

The framework provided with the assignment was used for simulation. Our code was at first uploaded to the Kongull cluster, but this took a lot of time due to long queue times. The framework was therefore installed on one of our computers, running on Linux CentOS. This worked fine, but most of our other computers couldn't run the simulator because the GCC/G++ versions were too new.

The simulator used was the modified M5 simulator provided with the assignment.

Chapter 5

Results

Chapter 6

Discussion

Chapter 7

Conclusion