# 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**

# Chapter 1

# Introduction

The performance of a modern day microprocessor is much higher than that of typical memory. Much of the computational time is thus used to access the memory of the RAM and load it into the CPU. This is a growing bottleneck in a time where microprocessors are still increasing in performance.

A prefetcher reduces this bottleneck by predicting which instructions are addressed next. Memory fetches are attempted to be done before the memory is needed by the microprocessor, leading to a decreased time where the processor is stuck in a waiting state. If the prefetched memory addresses differ from what was needed, the processor needs to access the memory anyway. This is the worst case scenario, in which the cache has no effect on performance.

The goal of this project was to make a prefetcher that would increase the performance of a microprocessor. The idea was also to run the prefetcher in a simulator environment with equal hardware specifications for everyone, so that everyone could work with a common interface and the results would be comparable.

The prefetcher 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 3. 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

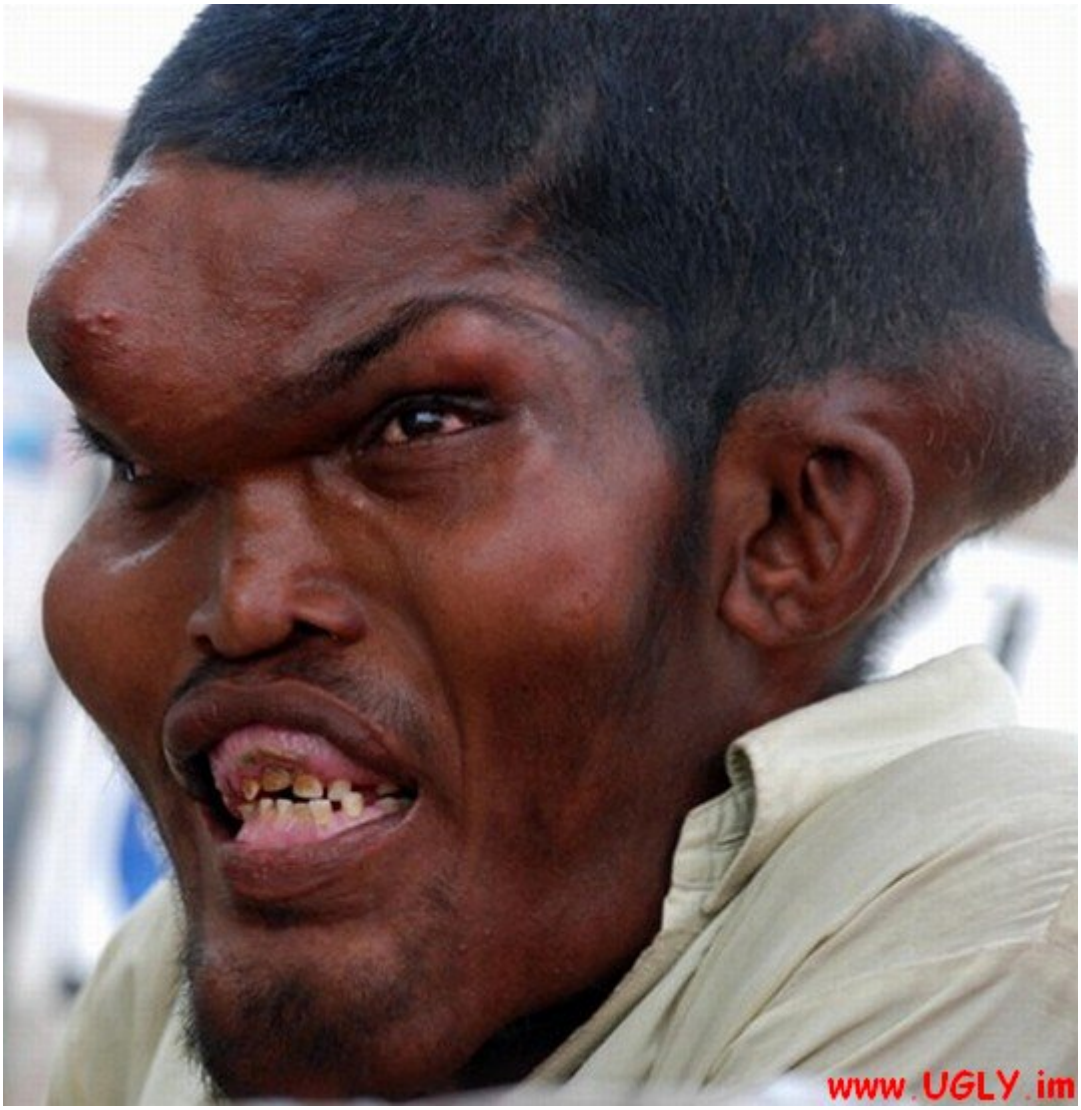
# Results

| | Speedup | IPC | Accuracy | Coverage | Identified | Issued |
|---|---|---|---|---|---|---|
| ammp | 0.999 | 0.082 | 0.060 | 0.000 | 13765492 | 32526 |
| applu | 1.014 | 0.523 | 0.681 | 0.068 | 662896 | 233366 |
| apsi | 1.015 | 1.507 | 0.628 | 0.020 | 60110 | 3777 |
| art110 | 0.999 | 0.122 | 0.519 | 0.006 | 1728334 | 182419 |
| art470 | 0.999 | 0.122 | 0.519 | 0.006 | 1728334 | 182419 |
| bzip2$_{g}raphic$ | 1.055 | 1.390 | 0.945 | 0.320 | 46207 | 31750 |
| bzip2$_{p}rogram$ | 1.021 | 1.515 | 0.962 | 0.128 | 10422 | 7394 |
| bzip2$_{s}ource$ | 0.997 | 1.705 | 0.964 | 0.209 | 10327 | 7282 |
| galgel | 0.998 | 0.443 | 0.388 | 0.008 | 249284 | 7166 |
| swim | 0.978 | 0.669 | 0.309 | 0.019 | 2033005 | 144134 |
| twolf | 0.997 | 0.423 | 0.632 | 0.001 | 1366 | 862 |
| wupwise | 1.059 | 0.791 | 0.343 | 0.187 | 279083 | 236426 |

# Chapter 6

# Discussion

As shown in table REF, the prefetcher



# Chapter 7

# Conclusion