Final Report

Course: ECSE 425

Team: Bankers

Ismail Faruk
Nabila Aziz Shifat
Jiawei Ni

Project Supervisor: Professor Amin Emad

Table of Contents

Introduction

Our objective is to investigate the effects on the performance of cache memory using EduMIPS64 and Dinero cache simulator. We explored the effects on Miss Rate by varying block sizes, cache sizes, etc on direct mapped cache and different set associative caches.

Our motive behind the project was to implement multi-level caches in EduMIPS because originally EduMips has no cache and only memory. After implementing the cache we wanted to test the effect on miss rate with cache. However, we got to know later that there is already a cache simulator called Dinero and so we planned to integrate it with EduMips and created UI to test the effects on miss rate.

Dinero can only be implemented using command lines, which is based on the Linux bash script. EduMIPS64 passes the command line to Dinero, which contains the cache parameters. Bash scripting was a required knowledge to use Dinero. Also, the script would get really hard to write for a multi-level cache. Hence, we decided to change how the user interacts with Dinero and the change was successfully implemented to the base code.

Approaches

We used EduMIPS64 to run a benchmark assembly code. We then configured and passed cache configuration parameters to Dinero to simulate a cache.

We setup a new GUI framework on the existing EduMIPS64 source code to implement and operate Dinero. We created a GUI framework to setup cache parameters, without having to know bash syntax. We added java swing components to the base code, because that seemed to be the best fit of code to be added. We could then auto-generate bash scripts for Dinero following its particular syntax. With the changes we made to EduMIPS64, we wanted to reduce the need for scripting knowledge and allow users to focus on cache configuration instead.

Implementations

We tried implementing a cache in EduMIPS even though we did not use it in the end. A cache.java file is added into the core folder. Afterward, we realized that Dinero is already a cache simulator that can be used to test our cache performance. Hence, we decided to integrate Dinero cache simulator with EduMips64 instead of implementing our own cache simulator.

For upgrading UI/UX, we implemented the GUI change in *UI/Dinero Frontend.java*. The new UI framework had the capability to exploit full functionality of Dinero, upto ***5 levels*** of cache, of ***data***, ***instruction*** or ***unified*** cache type. We added new components: **JPanels**, **JCombobox**, **JCheckbox**, **JLabels**, and **JScrollPane** to make the UX/UI better. We implemented the incomplete *DineroSingleCachePanel* class, the object class of the cache panels, that instantiated the *DineroCacheOptions* object. We created static objects when dynamic action capability was required. We added “create” and “configure” action buttons to dynamically create and remove cache panels in the UI based on the users choice of cache configuration and auto-generate bash script for dinero. We added combo buttons to give limited and valid configuration options based

on generic cache constraints and the constraints of Dinero. Finally, we added a scroll bar and added auto-refresh functionality so that the frame is responsive to the user's actions. Next, we have provided a screenshot of the Dinero Frontend before and after our upgrade.

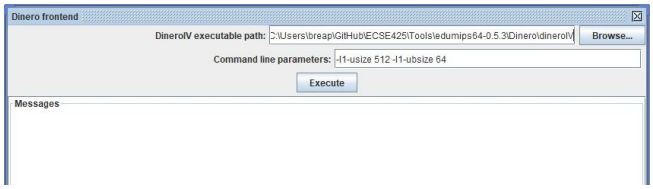

Fig 1: Old Dinero Frontend UI

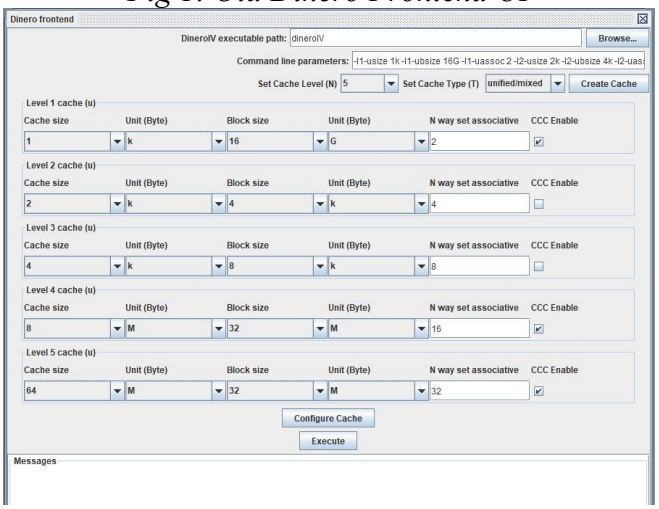

Fig 2: Upgraded Dinero Frontend UI

Experimental Setup

Dinero requires a Linux environment. Because none of our group members have Linux installed in our computers, we set up a virtual machine and installed Ubuntu to run Dinero. We prepared a sample assembly code for simulation. We ran it in MIPS and generated a memory trace file. This trace file will be used for benchmarking different caches. We used UI that we integrated with EduMips for testing.

Results

1. The Effect of Block Sizes on Miss Rate

In the first set of tests, we kept the cache associativity the same and changed the block sizes. For example, with set associativity of 1 (Direct mapping), we varied the block sizes from 4B to 256B for different cache sizes, namely 64B, 128B, 256B, and 64kB. As shown in *Fig 3*. We observed Miss rate initially decreases as block size increases. Then the Miss Rate starts rising as the block sizes continue to increase. As cache size increases miss rate decreases.

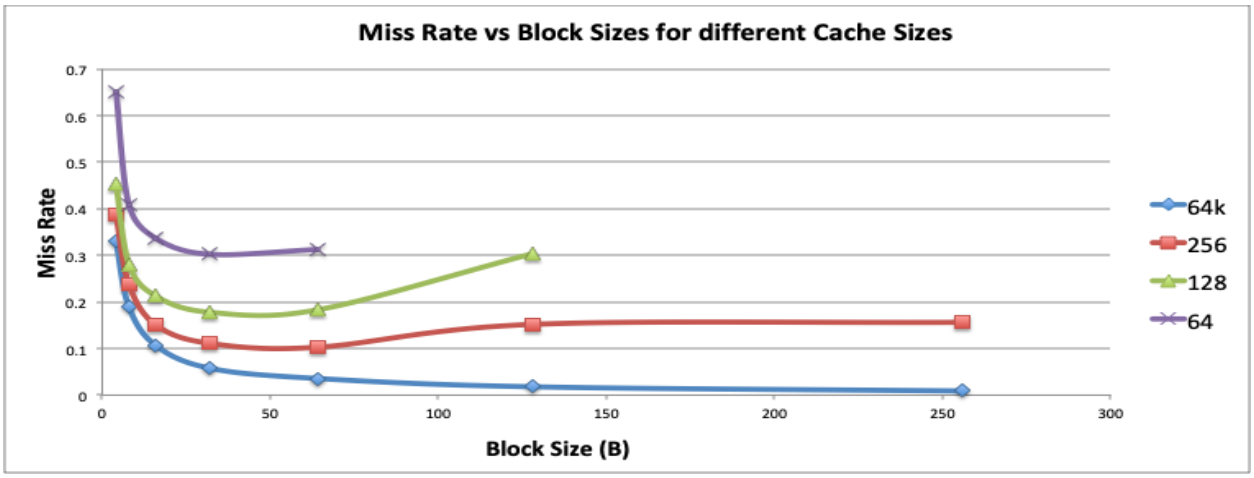

Fig. 3: Shows how miss rates behave as the block size increases for different cache sizes.

We also saw the trend in the miss rate components as block size increases using a cache size of 256B and set associativity set to 1. The result is shown in *Fig 4*, we can see that the capacity miss rate increases as the block size increases. This is the reason that the total miss rate increase at the end of the curve. As expected compulsory miss rate decreases with an increase in block sizes.

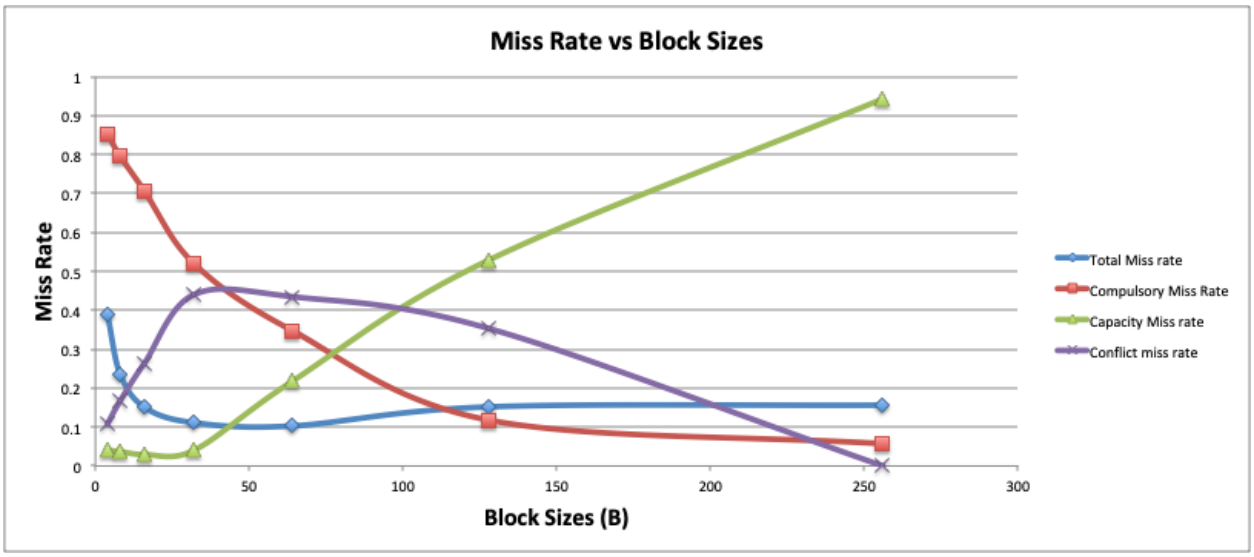

Fig 4: Shows how different types of miss rates components behave as the block size increases.

2. The Effect of Cache size on Miss Rate

We kept the block sizes the same and the set associativity is set to 1 and observed the Miss Rate as we change the cache sizes. We tested cache sizes of the range from 4B to 1kB. The result in *Fig 5*, as expected, shows that the larger the cache size, the lower the miss rate. When we increased the cache sizes to more than 512B the miss rate remained the same to 0.3308 for all other cache sizes because our program takes up a memory of 344B. As a result, the compulsory miss rate is 1 and so there are no capacity or conflict misses.

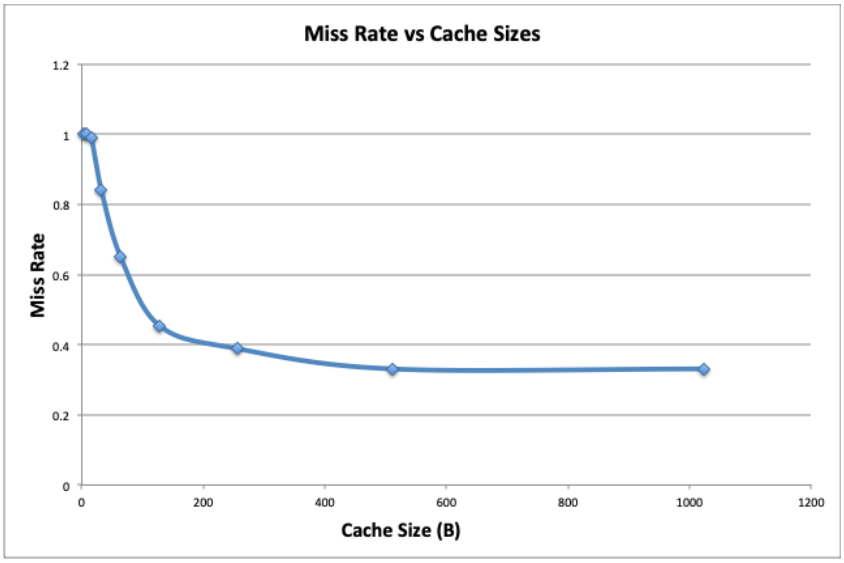

Fig 5: Miss Rate for different cache sizes

3. The Effect of associativity on Miss Rate

We increased the set associativity to compare different types of miss rates with a block size of 32B and cache size of 256B. We generally observed that the higher the associativity, the lower the total miss rate. *Fig 6* shows that compulsory miss rate generally increases with an increase in set associativity whereas capacity miss rate decreases. However, conflict misses roughly remains the same throughout the experiment.

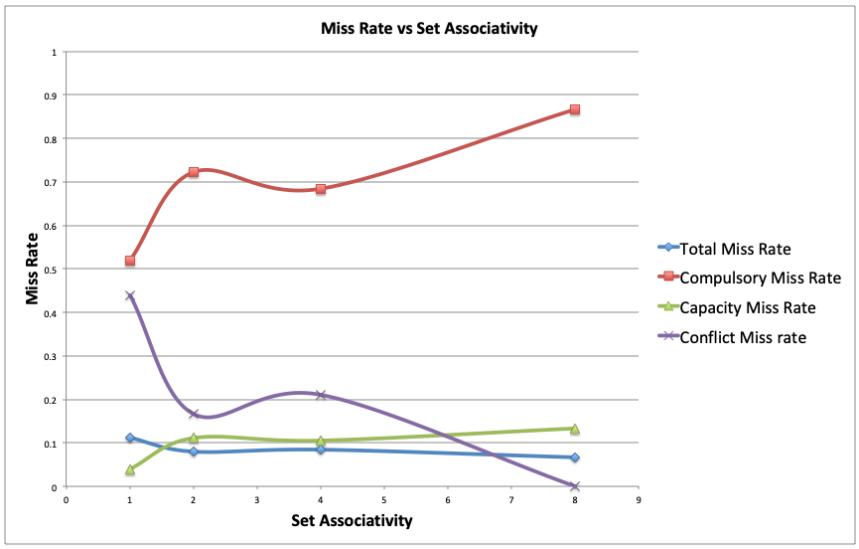

Fig 6: Miss Rate vs Cache Associativity

Post-mortem

We tried implementing a cache in eduMIPS for benchmarking. Later we realized that it was unnecessary, and that we are only required to use Dinero for cache simulation. The feature was scraped out of the testing build.

During the testing phase, we initially tested a set of caches that have sizes from 32kB to 64kB and block sizes from 4B to 64B. As we changed the size of the caches and the associativity, the miss rate didn't change at all. We eventually realized that the cache size we set is too large, so all the cache misses only comes from the initial reads. This also might be caused by the sample assembly code being too simple and not doing enough memory accesses. To solve this problem, we decided to decrease the cache size. When cache size was lower than 1kB, the miss rate started to change. We further wanted to calculate AMAT and CPU performance but we could not simulate Hit Time and Miss Penalty as these two parameters are technology dependent and needed a different simulator called CACTI.

After adding UI functionalities to Dinero Frontend, we checked the GitHub repository of EduMIPS64 - https://github.com/lupino3/edumips64. We saw that they have not implemented a GUI themselves for using Dinero. Thus, we have emailed the main author of the repository to allow us to make a pull request to add our UI upgrade to the official EduMIPS64.We hope they reply and our changes get accepted.

Project GitHub: https://github.com/ismailfaruk/ECSE425--Computer-Architecture