Understanding Caches

Issued : 2nd March 2015

Due : 23rd March 2015 at 4.00pm

This assignment represents the second practical component of the Computer Architecture module of Inf3. This practical contributes 20% of the overall mark for the module. It consists of a programming exercise culminating in a brief written report. Assessment of this practical will be based on the correctness and the clarity of the solution (see more details below), and on the completeness and clarity of the written report. This practical is to be solved individually to assess your competence on the subject. Please bear in mind the School of Informatics guidelines on plagiarism. You must submit your solutions before the due date shown above. Follow the instructions provided in Section 2 for submission details.

In this practical, you are required to build a simulator to explore Caches. You are strongly advised to read up on cache architectures in the course textbook and commence work as soon as possible.

Furthermore, you are required to document your experiments and inferences from these experiments, along with the structure of your simulator and any additional information in a hand written report and submit it before the deadline.

1. Experiments

A detailed understanding of the cache's dynamic behaviour is usually obtained by software simulation rather than monitoring of hardware. In this exercise you are asked to investigate the memory access behaviour of a pair of benchmarks (gcc and mcf) which are part of the SPEC2006 benchmark suite. To do this, you will have to write a reasonably flexible cache simulator. Assume the following specifications in designing the cache simulator: (i) memory addresses are 48 bits long (addresses that are shorter in length should be zero-extended); (ii) data in cache is organized in blocks of 32 bytes; (iii) write through cache; (iv) LRU replacement policy for set associative caches; (v) consider both write allocate and write no-allocate for cache misses.

You must write a program which accepts a trace file as input and reports upon its behaviour. The cache configuration (number of sets, associativity and allocation policy) should be specified using command line switches. The output of the simulator should be the **total miss-rate, memory read miss-rate and memory write miss-rate**. You are required to provide a readme file with the cache simulator with a description of it's compilation and execution procedure.

As part of the experiments, you are required to generate the miss-rate by **varying the cache size from 4 Kilobytes to 64 Kilobytes** by varying both the number of ways and the number of sets in the cache. To do this, first assume a direct-mapped cache and increase the number of sets so that the cache size varies from 4 KB to 64KB doubling the cache size each time. Do the same for a 2-way, 4-way, 8-way and 16-way set associative cache. Represent the results in appropriate graphs. Also present your inference from these experiments in your report.

The trace file for cache simulation contains the sequence of memory operations (Read -R, Write -W) and the addresses they access. An example of the trace file is as follows. The trace file consists

of raw addresses and not block addresses. Therefore, you will have to decode the block addresses from these raw addresses in order to be used in your cache simulator.

R 8cda3fa8
R 8158bf90
W 8cd94c50
R 8cd94c60
W 8cd94c60

To summarize, in this experiment you are required to write a cache simulator which reads a trace of memory references and generates 3 miss-rates (total, memory reads and memory writes). The cache configuration is determined by parameters that should be read into the simulator via command line switches. The parameters are the associativity, the number of sets in the cache, and the allocation policy (write allocate, write no-allocate). Present your inference from these experiments in your written report. Make sure that your simulator is runnable on a DICE machine.

2. Format of your submission

Your submission should clearly indicate (in both the report and the code) which cache features you have simulated completely and which you have only partially completed.

You should submit a copy of your simulator and a soft copy of your report (compressed file titled <cw2-name_matricno.tar.gz>) before 4pm on 23 March 2015, using the command

```
submit car 2 cw2-name_matricno.tar.gz
```

at a command-line prompt on a DICE machine. Make sure that your simulator is accompanied by a readme file detailing the compilation and execution procedure of your simulator.

The report should contain the following details:

1. Written description of the internal structure and workings of your simulator (around 1 page depending upon complexity), explaining clearly which parts are complete and which incomplete.

2. A report detailing the experiments you ran and giving a critical summary of the results (around 1-2 pages including results). You should provide enough information to make the experiments repeatable. Present your data clearly and concisely. Also present any inferences you have obtained from the results of these experiments, with reasons for these inferences.

3. A **separate** readme file stating which parts are complete and how to run the simulator.

3. Marking Scheme

- Direct-mapped cache – 24%
  - Write allocate – 12%
    - 4% each for total miss rate, memory read miss rate and memory write miss rate
  - Write no-allocate – 12%
    - 4% each for total miss rate, memory read miss rate and memory write miss rate

- Set-associative cache – 48%
  - Write allocate – 24%
    - 2% each for total miss rate, memory read miss rate and memory write miss rate of each associativity value
  - Write no-allocate – 24%
    - 2% each for total miss rate, memory read miss rate and memory write miss rate of each associativity value

- Clarity of code within the simulator(s) – 6%
  - This includes a modular structure of the code, consistent indentation, and comprehensive comments

- Report on the Simulator – 10%
  - 6% for stating internal structure – this should be an overview of the major components structured in a logical manner
  - 4% for readme file
    - 1% for stating which parts are complete and which incomplete
    - 3% for instructions on how to use the simulator and clarity of readme as a whole

- Report on Experiments – 12%
  - 2% on stating which experiments were run successfully and which failed
  - 5% on critical summary of the results and clarity of presentation
  - 5% on inferences drawn from the results and reasons for these inferences

Your mark will then be scaled to be worth 20% of the course.

The marker will run your simulator to verify the reported miss rates. In the case of incomplete/inaccurate simulator, efforts will be made to give partial credit for what work has been completed. Clear comments explaining the important tasks (and where they are missing/failing) will increase your chances of receiving credit.

N.B. Marks will be deducted if you significantly go over the page limit on reports for either the simulator or the experiments.

4. Working in Stages

This section contains some suggestions as to how you might tackle the work in stages. There is no obligation to follow these, but your report must make it clear exactly what your simulator can handle.

Stage 0

Find the code for your simulator from the first assignment and re-familiarise yourself with it sufficiently that you can use it for this assignment. (Alternatively you can write a new simulator from scratch)

Stage 1

Restrict your simulator to direct mapped caches (in other words, the set associativity must be 1).

You can test this out with hand generated traces. Verify that your simulator is doing what you think. Write trace files with easily predictable hit rates (e.g. think of a four access trace which uses four different addresses and has a 50% hit rate).

Stage 2

Upgrade your simulator to cater for genuinely set-associative caches (i.e. with set size greater than 1). You could start by assuming some fixed scheme for resolving clashes (e.g. random) and then build in the ability to simulate LRU. You can use hand generated traces which access the same set in the cache but are to different cache blocks. Verify which blocks are getting replaced and when. This will help you verify the LRU policy for set-associative caches.

Stage 3

Once you have verified your cache simulator, start running the required experiments by fixing the associativity, and changing the cache size and allocation policy. Repeat the same for different associativity values. Don't forget to add the results and your inferences from it in your report.

5. Reporting Problems

Send email to jcanore@inf.ed.ac.uk for any issues regarding the assignment.