Inf2C Computer Systems

Coursework 2

Cache Simulator

Deadline: Wed, 28 Nov, 16:00

Instructor: Boris Grot

TA: Siavash Katebzadeh

The aim of this assignment is to write a parameterized simulator that models a cache. The simulator, written in C, will read a memory address trace and provide various statistics for the cache. An outline of the simulator is provided in `mem_sim.c` file to get you started. You are strongly advised to read up on caches in the lecture notes and course textbook, and to commence work as soon as possible.
This is the second of the two assignments for the Inf2C-CS course. It is worth 50% of the coursework marks and 20% of the overall course marks.
Please bear in mind the guidelines on academic misconduct, which are linked to from the online Undergraduate Year 2 Handbook http://web.inf.ed.ac.uk/infweb/student-services/ito/students/year2.

1 Simulator Design

The simulator will implement a parametrized cache with a configurable size, associativity and replacement policy.
A cache is a fast memory used for storing instructions and/or data that were recently accessed and, based on the principle of locality, are likely to be used in the near future. When the data is found in the cache, it saves the processor from a long-latency access to main memory.

A direct-mapped cache allows a given block to reside in exactly one location. This lack of flexibility in block placement means that if two (or more) frequently-accessed blocks map to the same cache location, they might repeatedly kick each other out, likely resulting in a low hit rate despite abundant temporal and/or spatial locality. Such behavior is called *thrashing*. Set-associative caches can reduce the likelihood of thrashing by allowing a block to reside at any one of several locations in the cache. The higher the associativity of a cache, the more options there are for block placement, and the lower the likelihood of thrashing.

Set-associative caches introduce a problem that doesn't exist in direct-mapped caches: *On a cache miss, how do we choose which entry to replace?* The answer is dictated by the cache replacement policy. In this assignment, you need to implement 3 different replacement policies:

1. FIFO: Using this policy, the cache behaves in the same way as a FIFO queue. The cache evicts the block accessed first in a given set without any regard to how often or how many times it had been accessed.
2. LRU: Discards the least recently used items first. Implementing this policy requires tracking the order in which the ways of each set have been accessed.
3. Random: Randomly selects a candidate item and discards it to make space when necessary. This algorithm does not require keeping any information about the access history.

The cache will use the following parameters:

Fixed parameters:

1. Physical addresses are 32-bits
2. Memory is byte addressable

Variable parameters: The following parameters are passed as command-line arguments to the program and populated for you by the provided skeleton code.

1. `replacement_policy`: FIFO, LRU or Random
2. `number_of_cache_blocks`: 16, 64, 256 or 1024
3. `associativity`: Should always be a non-negative power of 2 (1, 2, 4, ... , number_of_cache_blocks)
4. `cache_block_size (in bytes)`: 32 or 64

Note that if the associativity is set to **1**, the cache is effectively direct-mapped. Also the maximum valid number for associativity is the number of cache blocks; this configuration makes the cache fully associative.

For every memory reference in the trace file, you should query the cache and update the relevant hit/miss statistics; details of this are below.

1.1 Trace and Code Files

You are provided with two files: 1) `mem_trace.txt` and 2) `mem_sim.c`. The memory trace file, `mem_trace.txt`, consists of a sequence of memory addresses. The code to read this trace file is provided for you. Your simulator should consume the addresses in the trace file one after another, in order. Each line in the trace file is a 32-bit hex-encoded physical memory address. An example of the trace file is as follows:

```
8cda3fa8
8158bf94
8cd94c50
8cd94d64
8cd94c54
```

The C file, `mem_sim.c`, contains the code for:

1. reading the command-line parameters and initializing the corresponding variables
2. reading the trace file
3. outputting end-of-simulation statistics

1.2 Output Format

For your convenience, the output of the program is already provided in the *print_statistics* function. You need to populate the parameters of the function with appropriate values. The following is the full list of variables you need to populate.

1. `CacheTagBits`: number of bits required for a tag in cache
2. `CacheOffsetBits`: number of bits required for the cache block offset
3. `Cache:hits`: total number of cache hits while accessing data
4. `Cache:misses`: total number of cache misses while accessing data
5. `Cache:hit-rate`: the percentage of accesses that result in cache hits

Summary: In this assignment, you are required to write a cache simulator which is fed by a trace of memory addresses. You will need to appropriately configure the cache data structures based on parameters that are passed as command-line arguments. You will collect the cache statistics, which will be output at the end of the simulation by the provided function.

1.3 Compiling and Running the Simulator

Compile the simulator on the DICE machines with the command:

```
gcc -o mem_sim mem_sim.c -std=gnu99 -lm
```

This creates an executable `mem_sim`. Make sure that your simulator compiles and runs on a DICE machine without errors and warnings. The following are examples of invoking the simulator with valid command-line parameters.

```
./mem_sim LRU 2 256 64 mem_trace.txt
```

`mem_sim` is the executable name. `LRU` indicates you should simulate LRU replacement policy. The next parameter, `2` specifies the associativity of the cache, which is 2. Total number of cache blocks is 256 and each block is 64 bytes. `mem_trace.txt` is the name of the tracefile which is located in the same directory as `mem_sim`.

```
./mem_sim Random 1 1024 32 mem_trace.txt
```

`Random` indicates the Random replacement policy. The next parameter specifies the associativity of the cache; in this case, it's `1`, which indicates that the cache is direct-mapped. The last two parameters specify the number of cache blocks (1024) and block size (32).

Given the following input, an example output appears below. Note that in this example, the number of hits and misses are not meant to be meaningful.

```
./mem_sim FIFO 4 256 32 mem_trace.txt
```

```
input:trace_file: mem_trace.txt
input:replacement_policy: FIFO
input:associativity: 4
input:number_of_cache_blocks: 256
input:cache_block_size: 32

CacheTagBits:21
CacheOffsetBits:5
Cache:hits:20
Cache:misses:20
Cache:hit-rate:50.0%
```

1.4 Submission

You should submit a copy of your simulator by 4pm on November 28, 2018 using the following command at a command-line prompt on a DICE machine.

```
submit inf2c-cs cw2 mem_sim.c
```

Unless there are special circumstances, **late submissions are not allowed**. Please consult the online undergraduate year 2 student handbook for further information on this.
You can submit more than once up until the submission deadline. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.
Warning: Unfortunately the `submit` command will technically allow you to submit late even if you submitted before the deadline. Don't do this! We can only retrieve the latest version, which means you will be penalized for submitting late.
For additional information about late penalties and extension requests, see the School web page below. Do **NOT** email any course staff directly about extension requests; you must follow the instructions on the web page.
`http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`.

1.5 FAQ

1. You are expected to dynamically allocate memory using **malloc** for the cache data structure. Submissions that use statically allocated memory (e.g., statically declared arrays of fixed size) will be penalized. Variable-length arrays that can be sized based on a runtime param in C99 are **NOT** allowed.

2. In order to implement the Random replacement policy, you are expected to use the *rand* function. In `mem_sim.c` file, the *srand* function (which seeds the random number generator used by *rand*) has been already called in the beginning of *main* function. Do **NOT** call the *srand* function anywhere else.

3. To verify correctness of your implementation, you should write your own trace files with targeted access patterns and easily predictable hit rates. For instance, think of a trace with four accesses that uses four different addresses and has a 50% cache hit rate in a direct-mapped cache.

   To help get you started, we have provided a log file (`mem_log.txt`), which shows the details such as tag, set and offset for every memory reference in a small trace and explains the cache state (allocation, hit, miss) of that access. This file can be useful to get insight about how the cache works in different situations (adding new entry, stressing a set, ...).

Note that this file is provided only to help you to debug your code and understand how to go about designing your own test traces. Your simulator should **NOT** have any such logging in its output.

4. Your simulator will be tested with trace files different from the one provided.

5. Your implementation will be assessed on correctness and the code will be assessed on quality and readability. The latter includes using modular programming practices, quality comments, meaningful variable names and a clean presentation.

6. You can use the C library functions available from the header files that are already included in `mem_sim.c`. You may not use any library functions beyond these (i.e., do not include other C header files).

7. Your simulator must compile and run on DICE without warnings or errors when compiled with `gcc -o mem_sim mem_sim.c -std=gnu99 -lm`

8. You can submit more than once up until the submission deadline. Late submissions are not allowed and will receive a mark of 0.

2 Similarity Checking and Academic Misconduct

You must submit your own work. Any code that is not written by you must be clearly identified and explained through comments at the top of your files. Failure to do so is plagiarism. Detailed guidelines on what constitutes plagiarism can be found at: `http://web.inf.ed.ac.uk/infweb/admin/policies/guidelines-plagiarism`. All submitted code is checked for similarity with other submissions using the MOSS [1] system. MOSS has been effective in the past at finding similarities. It is not fooled by name changes and reordering of code blocks.

3 Questions

If you have any questions about this assignment, you may talk to Inf2C-CS lab demonstrators, tutor or the course instructor. Alternatively, you may also post questions on the online course discussion forum at `https://piazza.com/ed.ac.uk/fall2018/inf2ccs/home`.

November 13, 2018

[1] `http://theory.stanford.edu/~aiken/moss/`