Inf2C Computer Systems

Coursework 2

Cache Simulation

Deadline: Thu 26 Nov 2015, 16:00

Boris Grot, Rakesh Kumar

The aim of this assignment is to write a parameterized cache simulator that reads a memory address trace and provides statistics about cache accesses and hits. The simulator is to be written in C and to get you started an outline of the simulator is provided in cache_sim.c file. You are strongly advised to read up on caches in the lecture notes/course textbook and commence work as soon as possible.

This is the second of the two assignments for the Inf2C Computer Systems course. It is worth 50% of the coursework marks and 20% of the overall course marks.

Please bear in mind the guidelines on plagiarism which are linked to from the online Undergraduate Year 2 Handbook.

1. Cache Simulator

Cache is a special memory that is used to store instructions and/or data so that the processor can access them at a very high speed. A cache that holds both the instructions and data is called a Unified Cache (Just like Multi-cycle datapath design uses the same memory to store both instructions and data). However, an alternate design, called Split Cache, makes uses of two caches: one to store only data, called Data Cache and another only for instructions called Instruction Cache. (Similar to what we have in Single-cycle datapath where instructions and data are stored in different memories).

A detailed understanding of the cache dynamic behavior is usually obtained by software simulation rather than monitoring of hardware. In this exercise you are asked to investigate the memory access behavior of a benchmark. To do this, you have to write a reasonably flexible cache simulator. Assume the following specifications in designing the cache simulator:

Fixed parameters. The following parameters always stay the same:

(i) memory addresses are 32 bits;
(ii) block size is 64 bytes;
(iii) FIFO replacement policy for fully associative caches;

Variable parameters. The following parameters are passed as command-line arguments to the cache simulator:

(i) Cache size (128B – 4KB; has to be a power of 2);
(ii) Cache Mapping: Directed Mapped (DM) or Fully Associative (FA)
(iii) Cache Organization: Unified (UC: same cache for instruction and data) or Split (SC: 2 caches – one for data, the other for instructions)

In the case of Split Cache (SC) organization, the cache size (which is passed in as a command line parameter) is split equally among both the caches (data and instructions), meaning that each cache gets one-half of the specified capacity. Both caches also use the same mapping (i.e., DM or FA).

For example, if the command-line parameters are as follows:

1) Cache size: 2KB, Cache Mapping: DM, Cache organization: UC

then there will be just one cache (due to cache organization being UC) that will contain both instructions and data:

Parameters	Values
Cache Mapping	Direct Mapped
Block Size	64B
Cache Size	2KB
Number of blocks	32 (2KB/64B)
Num bits for block offset	6 (64B block)
Num bits for index	5 (32 blocks)
Num bits for tag	21 (32-6-5)

2) Cache size: 2KB, Cache Mapping: DM, Cache organization: SC

in this case, there will the separate caches for instructions and data (due to cache organization being SC). If the address read from the trace file is for an instruction, it will go to the instruction cache; otherwise, it will go to the data cache:

Parameters	Values (Instruction Cache)	Values (Data Cache)
Cache Mapping	Direct Mapped	Direct Mapped
Block Size	64B	64B
Cache Size	1KB (2KB/2)	1KB (2KB/2)
Number of blocks	16 (1KB/64B)	16 (1KB/64B)
Num bits for block offset	6 (64B block)	6 (64B block)
Num bits for index	4 (16 blocks)	4 (16 blocks)
Num bits for tag	22 (32-6-4)	22 (32-6-4)

2. Trace and Code files

You are provided with a memory trace file *mem_trace.txt* and an outline for the cache simulator in *cache_sim.c.*

The memory trance file consists of a sequence of memory accesses and it is an input to the cache simulator. For each memory access the trace file provides: a) whether its an instruction or data access and b) 32-bit memory address in hex. The addresses in trace file are byte addresses and not the block addresses. An example of the trace file is as follows:

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

where the letter “I” indicates that its an instruction access, “D” denotes a data access, and is

followed by a 32-bit address. In the case of a Unified Cache (UC), both instruction and data accesses go to the same cache. For a Split Cache (SC), memory accesses marked as I (e.g. 8cda3fa8, 8158bf94) go to the instruction cache whereas, the ones marked D (e.g. 8cd94c50, 8cd94c54) go to the data cache.

The file cache_sim.c already contains the code for:
1) reading the command-line parameters and initializing the corresponding variables
2) reading the trace file

The statistics you need to collect are:
(i) Number of accesses
(ii) Number of hits
(iii) Hit rate

Your output should have the following format:

1) For a unified cache:

```
U.accesses: 1000
U.hits: 200
U.hit rate: 0.200
```

2) For a split cache:

```
I.accesses: 1000
I.hits: 200
I.hit rate: 0.200

D.accesses: 1000
D.hits: 200
D.hit rate: 0.200
```

Note that the hit rate should have exactly 3 digits of precision, which you can get by using the `%1.3f` format specifier in the `printf` function.

Summary: in this assignment, you are required to write a cache simulator which reads a trace of memory references and reports 3 statistics (accesses, hits and hit rate). The cache configuration is determined by parameters that are passed as command line arguments. The command-line parameters are cache size, cache mapping (DM/FA), and cache organization (Unified/Split).

Hint: You can write your own 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) to verify your simulator before using the trace (mem_trace.txt) provided to you. Your simulator will be tested with trace files different form the one that is already provided to you.

3. Compiling and Running the simulator

Compile the simulator on the DICE machines with the command:

```
gcc -o cache_sim cache_sim.c
```

This creates an executable cache_sim. To run the simulator you need to pass command-line parameters as follows:

```
./cache_sim 1024 dm uc
```

where cache_sim is the executable name. 1024 is cache size in bytes, dm tells that the cache is direct mapped and uc signifies a unified cache.

The parameters can take following values:
cache size : 128 – 4096 (has to be a power of 2)
cache mapping : dm (Direct Mapped) or fa (Fully associative)
cache organization : uc (Unified cache) or sc (Split Cache)

Make sure that your simulator compiles and runs on a DICE machine.

4. Submission

You should submit a copy of your simulator before 4pm on 26 Nov 2015, using the command

```
submit inf2c-cs 2 cache_sim.c
```

at a command prompt on a DICE machine. Unless there are special circumstances, late submissions are not allowed. Please consult the online Undergraduate Year 2 Handbook for further information on this.

5. Similarity Checking and Plagiarism

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 system. MOSS has been effective in the past