EE382M – 15: Assignment 1

Professor: Lizy K. John
TA: Jee Ho Ryoo
Department of Electrical and Computer Engineering
University of Texas, Austin

Due: 11:59PM September 21, 2014

1. Introduction and Goals

The goal of this assignment is to study the performance evaluation of multilevel caches in multicore processors. Using a trace driven cache simulator, you will conduct a simple performance analysis, and eventually, understand performance implications of different cache organizations.

In this assignment, you are asked to implement a multi-level cache simulator. This assignment will be graded based on correctness. But in a later assignment, you will be judging the runtime efficiency of the single core cache simulator that you develop here. So it is always important to use efficient algorithms and data structures in your code.

1.1 Graded Items

The following items need to be completed by the deadline

1. Single core 1-level cache simulator
2. Single core 2-level cache simulator
3. Multicore 2-level cache simulator

1.2 Overview

The assignment asks you to write a multicore 2-level cache simulator in steps. Figure 1 shows the general structure of the memory subsystem you will be implementing. The multilevel cache

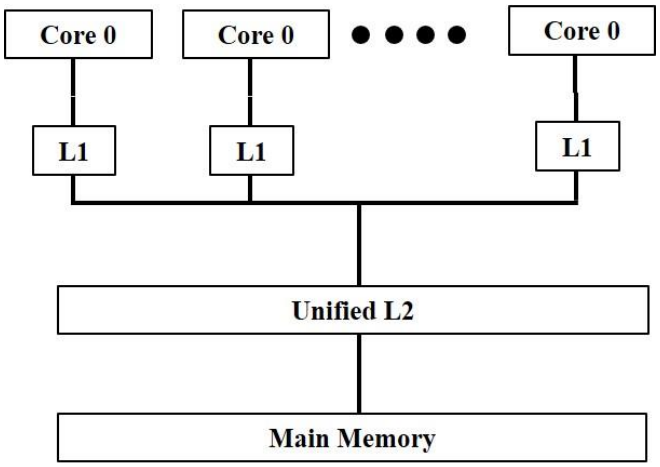

Figure 1: Overview of Memory Subsystem

simulator will have private data caches. They will be connected to the unified L2 cache. The L2 is the Last Level Cache (LLC) in this lab. The main memory is where L2 misses will be serviced. In this assignment, we will assume that there is a magic cache coherence protocol that you do not have to implement.

2. Single Core 1 Level Cache Simulator

2.1 Writing a single core one level cache simulator

In this section of the assignment, you are designing a single core 1 level cache simulator. Each core will have its own data cache. They will be connected to the main memory to handle misses.

Table 1 shows the parameters your simulator must be able to accept. Any value in the range indicated must be handled by the simulator in order to get full credit. All caches will be inclusive caches with a perfect LRU replacement policy. They are write-allocate, write-back caches. The parameters listed in

Table 1: Single Core 1 Level Cache Parameters

L1 capacity (powers of 2 in KB from 1KB to 64KB)
L1 associativity (1 to 32)
L1 cacheline size (powers of 2 in bytes from 64B to 1KB)
L1 hit latency (0-2)
L1 miss latency (2-20)

Table 1 will be read into the simulator from a single configuration file. All numbers will be powers of two, and your simulator is only responsible for handling those in the range specified.

The time at which the requests to the cache arrive is captured in the input trace. This timing information has to be taken into account. How to control the requests and replies when multiple requests arrive at the same time is explained in Section 4.2. Also, the input trace will be generated in a way that no multiple requests to the same cacheline will occur while the same cache line is currently being processed in the memory hierarchy. For simplicity, this cache simulator will have infinite bandwidth between caches, cores and the main memory. Therefore, every component can process as many requests as possible

2.2 Implementation Details

A query of a cache for misses and hits can be assumed to happen on the exact same cycle that a request is received. The latencies in Table 1 represent the number of cycles it takes a request to reach the next level of the memory hierarchy on a miss.

For the filling of cache lines into the caches on a miss, you can assume that these happen instantaneously after the correct number of latency cycles have elapsed. Metadata for the cache (dirty bits, lru, etc) will be updated when the cache has completed that request. For a hit, this is the same cycle that the request is received. For a miss, this happens when the cache line is filled.

2.3 Input trace

The input trace will have full 64 bit addressing and will be in the following format:

<cycle>,<R/W>,<address>

<cycle>: cycle number in hexadecimal format
<R/W>: 1 if read, 0 if write
<address>: address in hexadecimal format

You will be given a sample trace file and it is your responsibility to generate your own trace files that test/validate your simulator.

2.4 Output format

At the end of the simulation, your code must be able to output the hit rates, latency, references, and the Average Memory Access Time (AMAT). This metric is calculated by the accumulated sum of all memory request latencies divided by the total number of requests. Total latency is the accumulated sum of all memory request latencies while L1 references is the total number of references made to L1. Please note that latencies incurred by writebacks are not to be factored into AMAT calculation. At the end of the simulation, your code must be able to print the hit rates (with 2 decimal places), latency, references (as whole numbers) and the Average Memory Access Time (AMAT) with 2 decimal places.

% cachesim_l1…
L1 hit rate: 0.87
Total latency: 3000
L1 references: 150
AMAT: 20.00
%

Template files, sample input traces, and sample configuration file are compressed and are in the a1_2 directory in *a1.tar.gz* on the course website. The template file only has a skeleton where you have to implement the single core 1 level cache simulator. A sample configuration file is also provided with the lab template. You are free to change any value, but do not change the parameter name. A sample makefile is provided, but you are encouraged to modify it to suit your need. However, you are not allowed to change the output executable name, *cachesim_l1*. The command line argument is the following:

% <your cachesim directory>/a1_2/cachesim_l1 <config file> <trace-file >

Your simulator must be able to handle any parameter values listed in Table 1.

3. Single Core 2 Level Cache Simulator

3.1 Writing a single core 2 level cache simulator

In this section of the assignment, you are designing a single core 2 level cache simulator. Each core will have its own data cache. They will be connected to the unified L2 to handle misses. The unified L2 will be connected to the main memory. Table 2 shows the additional parameters your simulator must be able to accept.

Table 2: Single Core 2 Level Cache Parameters

L1 capacity (powers of 2 in KB from 1KB to 64KB)
L1 associativity (1 to 32)
L1 cacheline size (powers of 2 in bytes from 64B to 1KB)
L1 hit latency (0-2)
L1 miss latency/L2 hit latency (2-20)
L2 miss latency (100-400)
L2 capacity (powers of 2 in MB from 1MB to 16MB)
L2 associativity (1 to 32)
L2 cacheline size (powers of 2 in bytes from 64B to 1KB)

3.2 Implementation Details

Additional three scenarios in this section are:

L1 Hit (L1 hit latency cycles)
L1 Miss -> L2 Hit (L1 miss latency cycles)
L1 Miss -> L2 Miss (L1 miss latency cycles + L2 miss latency cycles)

Please load the cache line into both the L1 and L2 cache on an L2 miss. Also, please note that you are note that on an L1 cache miss, you are not allowed to check the L2 cache on the same cycle as the L1 cache miss. You must wait L1 miss latency cycles before the L2 cache receives the request from the L1 cache.

3.3 Input trace

The same input trace as Section 2 will be used.

3.4 Output format

At the end of the simulation, your code must be able to print the hit rates (with 2 decimal places), latency, references (as whole numbers) and the Average Memory Access Time (AMAT) with 2 decimal places.

% cachesim…
L1 hit rate: 0.87
L2 hit rate: 0.87
Total latency: 4000
L1 references: 300
L2 references: 40
AMAT: 23.25
%

Template files, sample input traces, and sample configuration file are compressed and are in the a1_3 directory in *a1.tar.gz* on the course website. The command line argument is the following:

% <your cachesim directory>/a1_3/cachesim_l2 <config file> <trace-file >

4. Multicore 2 Level Cache Simulator

4.1 Writing a multicore 2 level cache simulator

In this section of the assignment, you are designing a cache simulator. Each core will have its own data caches. They will be connected to the unified L2 to handle misses. The unified L2 will be connected to the main memory.

Table 3 shows the additional parameters your simulator must be able to accept in this sec.

Table 3: Multicore 2 Level Cache Parameters

Number of cores (1 to 16)
L1d capacity (powers of 2 in KB from 1KB to 64KB)
L1d associativity (1 to 32)
L1 cacheline size (powers of 2 in bytes from 64B to 1KB)
L1 hit latency (0-2)
L1 miss latency/L2 hit latency (2-20)
L2 miss latency (100-400)
L2 capacity (powers of 2 in MB from 1MB to 16MB)
L2 associativity (1 to 32)
L2 cacheline size (powers of 2 in bytes from 64B to 1KB)

4.2 Arbitration policy

There will be times when multiple requests arrive at the shared resource at the same time. You process them in the order by the following policy:

1. *Core 0* will get the highest priority and the priority will decrease subsequently until *Core N* which will get the lowest.
2. If the above policy cannot resolve it, then use the line number of the request in the trace file. A lower line number will have the higher priority. This situation can occur when a multiple requests are issued in the same cycle by the same trace.
3. If replies and requests require the same shared resource in the same cycle, replies always get the higher priority.

4.3 Implementation Details

Refer to Section 3.3.

4.4 Output format

At the end of the simulation, your code must be able to output the geometric mean of hit rates of different caches from all cores. In an *N* core system, you will have *N* L1 caches, so your code must report one L1 hit rate. Also, print the latency and references (as whole numbers) and the Average Memory Access Time (AMAT) with 2 decimal places.

% cachesim…
L1 hit rate: 0.87

L2 hit rate: 0.87
Total latency: 4000
L1 refeferences: 300
L2 references: 35
AMAT: 22.54
%

Template files, sample input traces, and sample configuration file are compressed and are in the a1_4 directory in *a1.tar.gz* on the course website. The command line argument is the following:

% <your cachesim directory>/a1_4/cachesim <config file> <trace-file 0> … <trace-file N-1>

In the command line example above, there are *N* number of cores. Each core will have its own trace file.

4. Assignment Guidelines

4.1 Coding guidelines

You can write your assignment in C/C++. The grading will be done with g++ on ECE LRC machines. If your code does not compile on these machines, you will lose 20% of the total grade automatically even if it is completely functional.

5. Submission Instructions and Grading Guidelines

5.1 Submission instructions

Your assignment must be compressed in tar.gz format. Please use the following command to tar your submission:

% tar –zcvf <your last name>-a1.tar.gz <your working directory>

There will be a submission link available on the course website, so please submit your tarball by deadline. Your submission directory must have the same directory structure as the template directory structure. Please remove all your temporary and input trace files. You are only required to submit your code and makefile. Make sure that the name of your working directory is named *<your last name>-a1*.

5.3 Grading guidelines

Your code will be compiled on LRC machines. LRC machine access information can be found on the ECE IT page (http://www.ece.utexas.edu/it/remote-linux). The grade will be based on correctness. Partial points will be given if appropriate comments are written in the code and late submissions will lose 20% of the total grade each 24 hours after the deadline.

5.4 Assignment questions

If you have any general questions, please post it under *a1* label on Piazza. The course teaching staff will not answer general questions sent to personal email addresses.