EE382N – 21: Assignment 3

Professor: Lizy K. John
TA: Shuang Song
Department of Electrical and Computer Engineering
University of Texas, Austin

Due: 11:59PM March 25th, 2018

1. Introduction and Goals

The goal of this assignment is to study the performance evaluation of multilevel caches in single core 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 exclusive 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. It is always important to use efficient algorithms and data structures in your code.

1.1 Graded Items

The following item need to be completed by the deadline

Report for PartA (20 pts)
Single core 3-level exclusive cache simulator for PartB (80 pts)

2. Understanding and evaluating cache implementation of ChampSim (PartA)

In this assignment, we are using simulation infrastructure provided by the 2nd Cache Replacement Championship (http://crc2.ece.tamu.edu/). This is a cycle accurate trace-based simulator for a microarchitecture study. It models multicore system with shared last level cache and detailed DRAM. Each core is superscalar processor with branch predictor, out-of-order execution, private L1 and L2 caches, data prefetchers. In this assignment, we are only focusing on a single core system with multilevel cache hierarchies.

ChampSim models non-blocking caches, i.e. the cache does not freeze if there is a miss. Future accesses to the cache can happen while the miss is being serviced. There is a maximum number of outstanding misses that cache can serve. In the ChampSim, cache miss block is not installed into cache until the correct number of cache miss latency cycles pass. Data requests are sent in the format of Packet, which contains very detailed information such as data address, data value read from the memory, PC of the instruction trigger this data, at which cycle should this data gets filled into cache, etc. Communication between different level of caches is made through these packets. There are queues to save these packets. For example, a cache read miss in upper level cache will trigger a read request to the lower level, which is modeled as upper level generating a

packet containing the data read request and inserting the packet to the corresponding queue of the lower level cache.

Compile and run simulations

The simulator should be able to be compiled using following command

./build_champsim.sh lru

This will create a binary called bimodal-no-no-lru-1core in the bin directory.

To run and test your simulator, refer to scripts/run_champsim.sh. An example run command could be

./bin/bimodal-no-no-lru-1core -warmup_instructions 1000000 -simulation_instructions 9000000 -traces bzip2_10M.trace.gz

Reference output of unmodified ChampSim code is given in file bzip_out.txt.

Please answer following questions

2.1 In the ChampSim, a cache is associated with multiple queues, which are WQ (write queue), RQ (read queue), PQ (prefetch queue), MSHR, and PROCESSED (processed queue). Read the simulation infrastructure, find out when a queue entry is allocated and deallocated in each queue.

2.2 Read the concept of alpha, beta, gamma tests in Black and Shen and the concept of "Performance Signatures Dictionary" as in Bose-Conte paper and create a performance signatures dictionary for detecting the modeling errors for this cache simulator. Create various tests to test the correctness of the cache simulator.

The following table is just an example. I am looking forward to seeing your creativity. Be creative.

Test Objective	Test Case	Expected Output	Cycles
Associativity (LLC)			
Replacement Policy (LLC)			
Cache size (LLC)			
……			

You need to write your own memory access traces in input file. Your input file should follow a strict format, such that each line only consists of two elements. First element can be either 0 or 1, indicating read or write; second element is virtual memory address in decimal format. Each test case can be a separate file (e.x., Test1.txt). An example input.trace is provided.

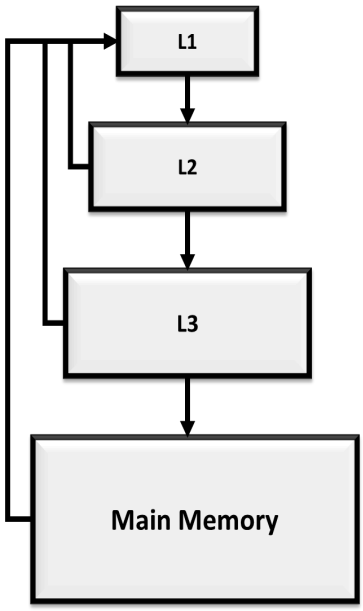
This trace file can be converted to the formatted needed for the simulator using the trace generator/convertor we provide. The trace generator is under *tracer/* directory. This is a synthetic trace generator. The trace generator only generates memory instruction traces, with all load instructions loading data into register0 and all store instructions reading data from register1.. Further information can be obtained via README file.

You would get bonus points if you find a bug in the simulator!

3. Implementing exclusive caches (PartB)

This section is about exclusive cache implement atop ChampSim. In the current ChampSim implementation, if data is missed in all levels of cache, after the data is returned from main memory (DRAM), every level of caches would allocate a cache block to fill the missing data (i.e., inclusive cache). You are asked to modify ChampSim code to implement exclusive cache hierarchies, where data is guaranteed to be in at most one level of caches.

In this exclusive cache implementation, you are asked to fill DRAM served data into L1 directly. That is to say, if L1 sees a cache miss, and the data also misses in L2 and LLC, only L1 cache is filled with the data. Thus, during the initial simulation period when all caches are empty, only L1 cache is filled with data. After L1 gets full and data block get evicted to make room for new block, evicted blocks from L1 are installed into L2. Similarly, L2 evicted block is installed into LLC. Figure on right illustrates data block movement across the memory hierarchy. When a data block misses in L1 but hits in L2, the data block will be installed to L1 and get invalidated in L2 to maintain exclusiveness. Similarly, a cacheline hit in LLC would lead to data block invalidation in LLC and promotion to L1.



Prior researchers have proposed multiple techniques to implement exclusive caches. The one we are going to implement is called Demote technique. Except for brief description mentioned above, you are encouraged to read related chapters in academic papers. For example, the introduction section in Gill's work, entitled "*On multi-level exclusive caching: offline optimality and why promotions are better than demotions*". You may find more links to exclusive caches in this paper as well.

You can assume all prefetchers are disabled so that you don't need to consider prefetching effects.

We only provide you one trace file for testing. To test your code, you are encouraged to either generate your own traces using the trace generator we provide, or use other single core traces provided by the 2nd Cache Replacement Championship. More information can be found on their website http://crc2.ece.tamu.edu/.

4. Assignment Guidelines

4.1 Coding guidelines

You may only change cache inclusiveness features, and maintain other features such as non-blocking cache, write allocate, same cache miss penalty cycles in both inclusive and exclusive caches, all kinds of parameters in header file, etc. You may not change build_champsim.sh, main.cc and ooo_cpu.cc. These files would be replaced with TA's files for grading purpose. As far as I know, there is no need to modify these files to

implement exclusive caches. In the cache.cc, you should not modify **check_hit2** cache function, which is called during grading. Your code should be able to be compiled using build_champsim.sh.
Except for the constraints mentioned above, you may make as many changes to the source and header files as you need.

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 eid>-a3.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 which can be compiled using build_champsim.sh file. Make sure that the name of your working directory is named *<your eid>-a3*.

5.3 Grading guidelines

Your code will be compiled on Linux machines. 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 10% of the total grade each 24 hours after the deadline.

5.4 Assignment questions

If you have any general questions, please post it under *a3* label on Piazza.