CS 4290/6290, ECE 4100/6100 - Spring 2017

Dr. Conte
Project 1 - Cache Simulator

Checkpoint Due : 13^{th} **February 2017, 11:55 PM**
Final Due: 20^{th} **February 2017, 11:55 PM**

Rules

- **This is an individual assignment. ABSOLUTELY NO COLLABORATION IS PERMITTED**. All cases of cheating will be reported to the Dean of Students.
- The due date at the top of this assignment is final. **It will not be changed**. Late assignments will not be accepted.
- For any questions, please use Dr. Conte's or one of the TA's office hours. If you cannot make it to office hours, please email the TA's.
- This is a tough assignment that requires a good understanding of concepts before you can start writing code. **Make sure to start early**.
- Read the entire document before starting. There may be critical pieces of information and hints along the way.
- Unfortunately, experience has shown that there is a very high chance that there are errors in the project description. **It is your responsibility to check the website often and download new versions of this project description as they become available**
- A makefile with the frontend will be provided. You will only need to fill in the empty functions and any other subroutines you may need. **Make sure all your work is written according to the C99 or C++11 standards, using only the standard libraries.**

1 Project Description

Caches are complex memory and often difficult to understand. One way to understand them is to build them, however we do not have time for that in this class. You will instead write a date cache simulator that can simulate SPEC benchmark traces, and then run design experiments on these workloads to find the best possible cache for each of the given workloads. We have provided you with:

- **cachesim.**{**h/hpp**} - Header file containing declarations of the statistics struct (more information ahead)
- **traces/** - The **SPEC** workloads inside the **traces** directory.
- **solution.log** - The solution generated by the TA's. You must debug your simulator until all the test cases match.
- **run_script.sh** - A script to run the test cases.

2 Simulator Specifications

Cache Layout - A single, sub-blocked L1 cache that also has a victim cache (VC). Detailed specifications are below:

- The simulator should model a cache with:
  - 2^C bytes of data storage
  - Having 2^B byte blocks
  - With 2^S blocks per set
  - Command line parameters C, B and S respectively can be used to modify these values
- Memory addresses are 64-bit long.
- The cache is **byte** addressable.
- The cache implements a write-back, write-allocate (WBWA) policy.
- The least recently used (LRU) cache replacement policy is used.
- Valid bits are set to **0** when the simulation begins.
- The cache has a **victim cache** that can hold V blocks. It has the following properties:
  - Is fully associative
  - Uses the first in first out (FIFO) replacement policy
  - Command line parameter V can be used to specify the size of the VC
- The cache implements **sub-blocking** where each sub-block is 2^K bytes in size. The command line parameter K dictates the size of the sub-blocks.
- Cache accesses are either READ or WRITE type. A READ event is a memory load operation of 1 byte to an address, and a WRITE operation is memory store operation of 1 byte to an operation.
- In general, (C, B, S, V, K) completely specifies the simulator specifications.

Cache Optimizations - Details of the victim cache and sub-blocking optimizations are below:

1. **The Victim Cache**
   - When there is a cache miss, first the victim cache is checked. If the block is present in the victim cache, the LRU block of the L1 cache set (the "victim") is replaced with the VC block that was just found. If the block is not present in the VC, it is fetched from memory. The victim block from the L1 cache is put in the VC. Remember that the victim cache uses a FIFO replacement policy.
   - When a block (the "victim") is evicted from the cache, it is written to the VC. Blocks evicted from the VC are written, if dirty, to memory.
2. **Sub-blocking**
   - Each block is divided into sub-blocks, each of 2^K bytes.
   - There is only **1** dirty bit for the **entire block**.
   - When there is a sub-block miss, the cache fetches all the sub-blocks from the **missed sub-block to the end of the entire cache block**. For example, say there are 4 sub-blocks in each block of an empty cache, and the CPU requests sub-block 3 of some block, the cache will fetch both sub-block 3 and 4 when servicing the CPU's request.

3 Implementation

3.1 details

Your Task is to write a simulator that will simulate the behavior of the above described cache by performing one cache access at a time, till you use all the addresses in the input trace file. Make sure to meet the following requirements:

1. We should be able to use **make** to build and compile your submission.
2. The executable should be called **cachesim**.
3. Make sure that the following command line arguments are supported by your simulator:
   - C The size of the cache is 2^C bytes
   - B The size of each block is 2^B bytes
   - S The cache has 2^S ways
   - V The victim cache can hold V blocks
   - K Each sub-block is 2^K bytes
   - i The input trace file

   Please note that the default values for the cache are given in the cachesim.h file we have provided. The default input should be **stdin** so that we can use the '<' operator to pipe a trace file into your simulator.
   For example, we should be able to call the simulator like this:
   `./cachesim -C 16 -B 4 -i traces/astar.trace`
4. The cache average access time is computed as per the following formula:
   $AAT = HT + MR_{L1} * MR_{VC} * MP$
   $HT = 2 + 0.1 * 2^S$
   $MP = 100$

   The increase in AAT due to sub-block misses can be computed by adding the sub-block misses to victim cache misses when computing the victim cache miss rate. So the final computation of victim cache miss rate will be:

   $$MR_{VC} = \frac{\text{(number of vc misses + number of sub-block misses)}}{\text{number of vc acceses}}$$
5. The trace files are in the following format:
   r {Address}
   w {Address}
   .....
   'r' specifies a **load** operation and 'w' specifies a **store** operation at the **'address'**.

3.2 Some Comments

- On eviction, make sure that you write back only the valid and dirty sub-blocks.
- On a sub-block miss, only invalid sub blocks are prefetched till the end of the block which is being accessed.
- When the victim cache size is 0, victim cache misses should also be zero.

4 Statistics (the output)

The output from your final cache is the statistics that your cache calculates for each workload. Here is the list of fields inside the `cache_stats_t` struct and their description:

1. accesses - Total number of accesses to the cache
2. reads - Number of reads
3. read_misses - Number of read misses
4. read_misses_combined - Number of read misses that also miss in the VC
5. writes - Number of writes
6. write_misses - Number of write misses
7. write_misses_combined - Number of write misses that also miss in the VC
8. misses - Total number of misses
9. write_backs - Number of write backs
10. vc_misses - Total number of misses in the VC
11. subblock_misses - Number of sub-block misses. That is a hit on the block, however a miss on the sub-block
12. bytes_transferred - Number of bytes transferred to (writeback) or from (miss repair) memory
13. hit_time - The hit time for the L1 cache
14. miss_penalty - The time taken for a memory operation
15. miss_rate - The miss rate for the cache simulation
16. avg_access_time - The average access time (AAT) of the overall cache system

Please note that sub-block misses are only incremented when there is a block hit, however the sub-block the processor wants is not present in the cache. They are not incremented on a cache miss.

5 Experiments

The below sub-sections detail the experiment requirements:

5.1 Validation Requirements

Sample outputs will be provided on T-square. You must run and debug your simulator till it **completely** matches all the statistics in the validation outputs posted on T-square. **Do not proceed to the next section without meeting this requirements**.

5.2 Cache Design

For **each** trace in the trace directory, design a cache subject to the following constraints:

(a) You have a total budget of 64KB = (65,536 * 8 bits) for the cache system's storage, including the tag storage, valid bits, dirty bits, cache data storage, both for the L1 and victim caches. Storage for LRU bits are excluded from this budget.

(b) The cache should have the lowest possible average access time (AAT).

(c) You can vary any parameter *(C, B, S, V, K)* to any values, subject to the following constraints:

- B = [3 .. 7]
- C = [B .. 30]
- S = [0 .. (C-B)]
- V = [0 .. 8]
- K = [1 .. (B-1)]

5.3 Comparison of Optimization Techniques

Suppose you only have the budget to implement one optimization. You need to decide whether your system should use **sub-blocking** or a **victim cache**. Explain why you would choose one technique over the other? What assumptions did you make to get to your decision, and what are your insights on how each optimization works?

6 What to hand in via T-square

Checkpoint

A preliminary cache simulator exhibiting basic functionality is due at the half way mark. You will need to submit a cache simulator that works for all test cases where the victim cache size is 0. That is, V = 0. You need to submit the following files for the **checkpoint**:

- All the source files for your simulator
- A working `Makefile` that the TA's can use to build your simulator

Final Deadline

- **All your source code**, including the `Makefile`. Make sure your code is well formatted, well documented and does not contain any **dead** code. Your code must compile on a recent version of Linux with the GNU compiler collection.
- **A document** with the design results of the experiments for each trace file, with a **persuasive** argument of the choices that were made. Make sure to include a **separate** section with your analysis of the **third** experiment (section 5.3) requirement. Make sure this document is in **PDF** format.
- Submit a single **tarball**, which contains both your code and the report with the following naming convention: {gtlogin}.tar.gz.

7 Grading

CHECKPOINT This will count for **5%** of the project grade.

DUE DATE - The following is the breakdown for rest of the **95 percent** of the available points:

- 0 % - Your simulator does not match any validation outputs
- 50 % - Your simulator matches all the validation outputs
- 90 % - Your simulator matches all the validation outputs, and you found the best cache for each trace.
- 95 % - The project you have handed in is award winning quality. For example, you have graphs, tables and persuasive arguments to justify each of your best caches.

NOTE: No late assignments will be accepted. Make sure you submit your project before the deadline.