Georgia Institute of Techology

School of Computer Science

CS 4290/6290, ECE 4100/6100: Fall, 2013 (Prof Conte)

Project 1: Cache design

Version 1.2

Rules

This is the first project for the course --- here are some rules:

1. Sharing of code between students is viewed as cheating and will receive appropriate action in accordance with University policy.

2. It is acceptable for you to compare your results, and only your results, with other students to help debug your program. It is not acceptable to collaborate either on the code development or on the final experiments.

3. You should do all your work in the C or C++ programming language, and should be written according to the C99 or C++98 standards, using only the standard libraries.

4. Unfortunately experience has shown that there is a very high chance that there are errors in this project description. The online version will be updated as errors are discovered. *It is your responsibility to check the website often and download new versions of this project description as they become available*.

5. A Makefile with the frontend will be given to you; you will only need to fill in the empty functions and any additional subroutines you will be using. You will also need to fill in the statistics structure that will be used to output the results.

Project Description:

Cache memories are hard to understand! One way to understand them is to actually build a cache. We do not have time to do this in this class. However, writing a simulator for a cache can also make caches easier to understand. So in this project, you will design a parametric cache simulator and use it to design data caches well suited to the SPEC benchmarks.

Specification of simulator:

Cache simulation capabilities:

a. The simulator should model a 2-level caching system with 2^{C1} and 2^{C2} bytes of data storage, having 2^{B1}-byte and 2^{B2}-byte blocks, and with sets of 2^{S1} blocks per set and 2^{S2} blocks per set, in level 1 and level 2 respectively (note that S=0 is a direct-mapped cache, and $S = C - B$ is a fully associative cache). Also note that the values are restricted to C2>=C1, B2>=B1 and S2>=S1.

b. The memory addresses are 64-bit addresses.

c. Caches are byte-addressable.

d. Each cache implements the write-back, write-allocate (WBWA) policy. There is an additional *dirty bit* for each tag in the tag store.

e. Each cache implements least-recently-used (LRU) as the replacement policy. The LRU cache chooses the least recently used block for replacement.

f. There is 1 valid bit per block of storage overhead required. The valid bits are set to 0 when the simulation begins.

g. The following are used to calculate AAT, hit time, and miss penalty for L1 and L2 (HT means Hit Time, MR means Miss Rate, and MP means Miss Penalty):

    - AAT = HT1 + MR1 * MP1
    - HT1 = 2 + 0.2*S1
    - MP1 = AAT for L2 = HT2 + MR2 * MP2
    - HT2 = 4 + 0.4*S2
    - MP2 = 500

h. L2 cache performs stride prefetching to reduce compulsory misses. The prefetching works as the following when L2 misses:

    - Define the function Block_Addr(X) = address of X with offsets bits of X to zero.
    - On a L2 miss to address X, calculate d = Block_Addr(X) - Last_Miss_Block_Addr.. Then set Last_Miss_Block_Addr = Block_Addr(X).
    - If d == Pending_Stride then prefetch the next K blocks. This means that we bring blocks with block addresses X + i * Pending_Stride, for i = 1..(K) from memory to L2 cache. K is given as a simulation parameter.
    - Note that prefetch does not contribute to AAT. Any prefetch are *not* included in the calculation of the miss rate of L2.

- A prefetched block becomes an LRU (least recently used) block in its set. (Hint: you can make the timestamp of the prefetched block equal to min(timestamps of all other blocks in the set) – 1.)

- Finally, regardless of whether d matches Pending_Stride or not, set Pending_Stride = d.

- Vary K in the range [0, 4].

i. In general, (*C1*, *B1*, *S1*, *C2*, *B2*, *S2*, *K*) completely specifies the caching system

Explanation of Provided Framework

We are providing you with a framework to build the cache simulator. You must fill in the following functions in the framework:

```
void setup_cache(uint64_t c1, uint64_t b1, uint64_t s1, uint64_t c2,
uint64_t b2, uint64_t s2, uint32_t k);
```

Subroutine for initializing the cache. You many add and initialize any global or heap variables as needed.

```
void cache_access(char rw, uint64_t address, cache_stats_t* p_stats);
```

Subroutine that simulates the cache one trace event at a time. Type can be either READ or WRITE, which is each defined in cachesim.hpp. A READ event is a memory load operation of 1 byte to the address specified in arg. A WRITE event is a memory store operation of 1 byte to the address specified in arg.

```
void complete_cache(cache_stats_t *p_stats);
```

Subroutine for cleaning up memory and calculating overall system statistics such as miss rate or average access time.

Statistics (output): The simulator must output the following statistics after completion of the run:

a. number of accesses to L1 (either hit or miss)

b. number of reads issued by CPU

c. number of read misses to L1

d. number of read misses to L2

e. number of writes issued by CPU

f. number of write misses to L1

g. number of write misses to L2

h. number of write backs to the main memory

i. number of prefetched blocks

j. number of successful prefetches (count first-time-hits on prefetched blocks)

k. the average access time (AAT)

The output of these variables should be handled by the driver code, and you only need to fill in the structure cache_statistics_t, defined in cachesim.hpp.

Experiments:

First validate your simulator (see the section on *validation requirement* below).

THE EXPERIMENTS ARE AS IMPORTANT AS HAVING A WORKING SIMULATOR!

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

1. You have a total budget of 48KB and 192 KB respectively for L1 and L2 cache system's storage respectively (including all tag storage, valid bits, data storage and overhead storage but exclude bits used for LRU and for prefetch registers)

2. The cache should have the lowest possible AAT

You may vary any parameter (*C1, B1, S1, C2, B2, S2, K*). K in the range [0, 4].

Validation Requirement

Four sample simulation outputs will be provided on T-Square by the TAs. You must run your simulator and debug it until it matches 100% all the statistics in the validation outputs posted on the website.

What to hand in via T-Square:

1. You should turn in exactly two files, “writeup.pdf” and “cachesim.cpp” (file names matter!). Submit them as they are, and do not compress them. You will lose points if you do not follow the submission rule.

2. A document (“writeup.pdf”) with the design results of the experiments for each trace file, with a *persuasive* argument of the choices that were made. (An argument may be as simple as an explanation of the search procedure used to find the designs and a statement about why the procedure is complete.) This argument should include output from runs of your program. (*There are multiple answers for each trace file, so I will know which students have "collaborated" inappropriately!*)

3. “cachesim.cpp” that implements your cache simulator. Do not submit any other source files. You are only allowed to change “cachesim.cpp” to implement the simulator. Make sure everything runs correctly with the clean template and your “cachesim.cpp” before submission!

4. Remember that your code must compile and run on a current variant of Linux (preferably Ubuntu) running on an x86 architecture (i.e., Intel or AMD).

5. **Note that late projects will not be accepted.**

Grading:

0%   You do not hand in anything by the deadline
+50%   Your simulator doesn't run, does not work, but you hand in significant commented code
+20%   Your simulator matches the validation outputs (5% each)
+20%   You ran all experiments and found a cache for each trace
+10%   The project hand in is award quality. For example, you followed the submission rule, justified each cache with graphs or tables and a persuasive argument.

Hints

There is no data store (or values) modeled in your simulator, just a tag store.