Georgia Institute of Techology
School of Computer Science
CS 4290/6290: Spring, 2015
Project 4: Cache Design
Due: Friday, April 3rd, 2015 at **11:55pm**

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:

The best way to understand cache memories 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 can model any caching system with 2^C bytes of total data storage, having 2^B-byte blocks, and with sets of 2^S blocks per set (note that S=0 is a direct-mapped cache, and $S = C - B$ is a fully associative cache)
b. The memory addresses are 64-bit addresses
c. The cache is byte-addressable
d. The cache implements the write-back, write-allocate (WBWA) policy. There is an additional dirty bit for each block required
e. The cache controller implements one of two storage policies (ST): BLOCKING or SUBBLOCKING
    - The BLOCKING cache will wait until the entire cache block is filled before returning to the processor the data
        - The hit time will be $0.2*2^S$ cycles

        - The miss penalty will consume $0.2*2^S+50+0.25*2^B$ cycles
        - There is 1 valid bit per block of storage overhead required
    - The SUBBLOCKING cache will retrieve and store a subset of the cache block. Either the first half of the block or second half will be requested and stored.
        - The hit time will be $0.2*2^S$ cycles
        - The miss penalty will consume $0.2*2^S+50+0.25*2^{(B-1)}$ cycles for the access
        - Accesses to the other half of the block will be a miss until requested and populated.
        - There is 2 valid bits per block of storage overhead required, one for each offset

f The cache controller implements one of two replacement policies (*R*): least-recently-used (LRU), or not-most-recently-used-FIFO (NMRU_FIFO)
    - The LRU cache chooses the least recently used block for replacement
        - There are 8 bits per block of storage overhead
    - The NMRU_FIFO cache will choose the oldest block, excluding the most recently used one
        - There are 4 bits per block of storage overhead

g The simulator also has a **victim cache** that is fully associative and contains 2^V most recently replaced blocks or subblocks, as appropriate:
    - **The VC uses the LRU replacement policy (irrespective of the policy the main cache implements).**
    - If there is a miss to the main cache and a hit in the victim cache (say for block *X*), block *X* is placed in the main cache. Then, the block that *X* replaces (say block *Y*) is placed in the victim cache in place of *X*. Block *Y* is now the most recently used block.

h Remember, all hit times and miss penalties need to be in integer, and should be round up to the nearest integer.

i In general, (*C, B, S, V, ST, R*) completely specifies the caching system

Explanation of Functions you need to fill in:

void setup_cache(uint64_t c, uint64_t s, uint64_t b, uint64_t v, char st, char r);
Subroutine for initializing the cache. You many add and initialize any global or heap variables as needed. st can be either BLOCKING or SUBBLOCKING, r can be either LRU or NMRU_FIFO (these are defined in cachesim.hpp).

void cache_access(char type, uint64_t arg, 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 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 outputs the following statistics after completion of the run:**

a number of accesses to the cache
b number of reads
c number of read misses to the main cache
d number of read misses to the main cache that also miss in the VC
e number of writes
f number of write misses to the main cache
g number of write misses to the main cache that also miss in the VC
h total number of misses (d+g)
i cache miss rate, which equals (total *number of misses*)/(*number of accesses*)
j the average access time (AAT)
k Total number of bits needed for overhead storage, such as valid and dirty bits, LRU and NMRU-FIFO
l Total ratio of storage overhead (total number of overhead storage bytes/total number of data storage bytes)

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 for the entire cache system's storage (including all tag storage, valid bits, data storage and overhead storage)
2 The cache should have the lowest possible AAT
3 The victim cache size must be less than the main cache's associativity (V <= S)

You may vary any parameter (*C, B, S, V, ST, R*).

Validation Requirement

Four sample simulation outputs will be provided on the website 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. You are required to hand in this validated output with your project (see grading).

What to hand in via T-Square:

1 Output from your simulation showing it matches 100% all the statistics in the validation outputs posted on the website for each validation run
2 A document 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 The commented source code for the simulator program itself
4 Remember that your code must compile and run on the reference machine.

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 justified each cache with graphs or tables and a persuasive argument

REFERENCE MACHINE:

CS STUDENTS: We will use **shuttle3.cc.gatech.edu** as the reference machine for this course. (https://support.cc.gatech.edu/facilities/general-access-servers)

ECE STUDENTS: We will use **ecelinsrv7.ece.gatech.edu** as the reference machine for this course. (http://www.ece-help.gatech.edu/labs/unix/names.html).

Before submitting your code ensure that your code compiles on this machine, and generates the desired output (without any extra printf statements). Please follow the submission instructions. If you do not follow the submission file names, you will not receive the full credit.

NOTE: It is impractical for us to support other platforms such as Mac, Windows, Ubuntu etc.