**CS320 – Fall 2021: Project 2**

**BrightSpace submission due: Monday, November 22nd at 11:59pm**  
**Project demos: lab session on Tuesday, November 23rd and TA’s office hours on Nov 23 and Nov 30**

The goal of this project is to measure the effectiveness of cache subsystem organizations using traces of memory instructions obtained from realistic programs. Each trace contains memory instructions with two values provided for each instruction: a flag indicating whether this is a load or a store (L stands for a load, S stands for a store), and the byte memory address targeted by this instruction. Three traces are provided.

Your goal is to write a program in C or C++ that would use these traces to measure the **cache hit rate** of various data cache organizations and prefetching techniques (note: we are not estimating the instruction cache performance in this project, only the data cache). Specifically, the following cache designs have to be implemented.

1. **[10%] Direct-Mapped Cache.** Assume that each cache line has a size of 32 bytes and model the caches sized at 1KB, 4KB, 16KB and 32KB
2. **[20%] Set-Associative Cache**. Again, assume that the cache line size is 32 bytes and model a 16KB cache with associativity of 2, 4, 8 and 16. Assume that the least recently used (LRU) replacement policy is implemented.
3. **[20%] Fully-Associative cache.** Assume that each cache line is 32 bytes and the total cache size is 16KB. Implement Least Recently Used (LRU) and hot-cold LRU approximation policies. For the hot-cold LRU approximation policy the initial state of all hot-cold bits should be 0 corresponding to the case where the left child is “hot” and the right child is “cold”. Furthermore, the policy should be utilized (and updated) for **all** accesses, including placing the initial blocks into the cache as well as replacements once the cache is full.
4. **[10%] Set-Associative Cache with no Allocation on a Write Miss.** In this design, if a store instruction misses into the cache, then the missing line is not written into the cache, but instead is written directly to memory**.** Evaluate this design for the same configurations as in question (2) above.
5. **[20%] Set-Associative Cache with Next-line Prefetching.** In this design, the next cache line will be brought into the cache with every cache access. For example, if current access is to line X, then line (x+1) is also brought into the cache, replacing the cache’s previous content. Evaluate this design for the same configurations as in question (2) above. Note that prefetched blocks **should** update the LRU order of the corresponding set meaning that the prefetched block should become the most recently used block in its set.
6. **[20%] Prefetch-on-a-Miss.** This is similar to part (5) above, but prefetching is only triggered on a cache miss. (Prefetched blocks should update the LRU order as in part 5).

**Materials on BrightSpace**

On BrightSpace, there is a tar/gzipped archive called project2.tar.gz, which contain the following materials:

* sample\_output.txt – Sample output file with comments
* traces/ – Directory containing three trace files
* correct\_outputs/ – The correct output for each of the provided traces

To access these materials, download a copy from BrightSpace, cd into the directory where you placed the tar/gzipped archive and issue the following command:

tar -xzvf project2.tar.gz

This will create a new directory (named project2) containing the files listed above.

**Submission requirements**

**Please submit ONLY the following files and pay close attention to naming. Remove any trace files or test output files. Make sure your Makefile builds a correctly named executable. Ensure that you have included an ASCII text file named README (not README.txt, just README, exactly like that; no lowercase!)**

You will need to submit your source code, so that we can compile it and test for correctness. You must ensure that your program compiles and runs correctly on the remote.cs.binghamton.edu lab machines. **Your score will be a zero if your project fails to compile.** For checking your code, we will be using the same three traces that have been provided to you, plus one more trace that you will not have access to.

The code that you submit should compile into a single executable called **cache-sim** with a simple make command. This executable should run all of the caches on the given trace, which will be specified via command line options as follows:

./cache-sim input\_trace.txt output.txt

Where:

* input\_trace.txt – file name of file containing the memory trace
* output.txt – file name of file to write output statistics

The output file should have the following format: (an example text file is on BrightSpace with comments, which should not be output by your program)

x,y; x,y; x,y; x,y; x,y;

x,y; x,y; x,y; x,y;

x,y;

x,y;

x,y; x,y; x,y; x,y;

x,y; x,y; x,y; x,y;

x,y; x,y; x,y; x,y;

Where each x,y; pair corresponds to the number of cache hits (x) and the total number of accesses (y) of one of the cache configurations. The first line provides the results for the direct mapped caches, second line for set associative, the third line for the fully associative cache with LRU replacement, the fourth line for the fully associative cache with hot-cold replacement, the fifth for the associative caches without store allocation, the sixth line for associative caches with next line prefetching and the seventh line for associative caches with next line prefetching only for cache misses. The numbers within each line should be separated by a single space.

Submissions will be checked using a script that will compare your output file to the correct output file using the UNIX diff tool, so if your output does not **EXACTLY** match the correct output, the grading program will mark your output as incorrect. The TA will have to check such submissions by hand, which will result in deducting points.

**Submission Rules**

You must submit all of the following:

1. All source code
2. A Makefile
3. A README, which minimally contains your name, BU-ID (everything before the @ in your Binghamton University e-mail) and your B Number. Other things to include might be: what works/what doesn't, things you found interesting, etc.

These materials should be turned in as follows: (using Hoda’s name and BU-ID as an example)

Hoda’s e-mail is hnaghibi@binghamton.edu so her BU-ID is hnaghibi

1. Create a new directory whose name is your BU-ID:

mkdir hnaghibi/

1. Copy all relevant files into this new directory
2. Create a tar/gzipped archive whose name is also your BU-ID from the directory as follows:

tar -czvf hnaghibi.tar.gz hnaghibi/

(The tar command should output the name of all archived files. Make sure there are no .o files, executables, traces, outputs, etc. in this list before submission. Make sure that your submission is not empty.)

1. Submit tar/gzipped archive to BrightSpace