**Programming Assignment 6 -- CS429 Fall 2013**

Due Date: 24 November 2013

**Purpose**

The purpose of this assignment is to give you experience with caches and evaluation of them. Also, I want to know if this victim cache idea helps cache behavior.

In addition, a major part of C programming, in the real world, is being able to take someone else's code and fix it or extend it. Also, programs always have to run faster. Also, to experience a late-date change in the program specifications.

**Motivation**

At [cache.tar](http://www.cs.utexas.edu/users/peterson/src/cache.tar)is a tar file of a C program and its Makefile that is a general cache simulator. This cache simulator takes a description of a set of caches (in the description file) and a trace file (of memory references), and computes the number of hits and misses for the described caches for the given memory trace. This allows us to easily compare different cache designs.

The current cache simulator has the ability to define a "property" to have a "value", by saying "property=value". For example,

{ name=lru, line\_size = 64, entries = 1024, ways=4, policy=lru }

Caches are described by their line size and number of entries. All caches are set-associative. We can model a direct-mapped cache by setting the number of ways to 1; we can model a fully associative cache by setting the number of ways equal to the number of entries. In addition, we can specify a replacement policy (LRU, FIFO, LFU, or RANDOM). If we have a LFU (least frequently used) policy, we need to define a decay interval.

After the cache is described, the memory trace is simulated, and statistics are printed for each cache design. For example:

lru: 256 entries of lines of 64 bytes; 64 sets of 4 ways, write-back, LRU 200000 addresses (133899 Fetch, 39430 Load, 26671 Store) 205335 hits (99%), 80 misses, 80 memory reads, 0 memory writes 15 dirty cache lines remain

**Program Specification**

Download the cache simulator and modify the code so that it can also simulate a victim cache. And we want the program to run no slower with this new function than it ran before.

To specify that a victim cache is to be simulated, add another option to the description of a cache. Add an ability to say "victim = n" to a cache description, to define a victim cache of n cache lines.

If a cache is defined to include a victim cache, create an n-entry fully-associative victim cache. Any cache line that is evicted from the base cache is put in the victim cache. A memory reference that is not in the base cache causes a search of the victim cache.

If the memory reference is in the victim cache it is used to satisfy the request for the cache line. To satisfy the request, a new victim must be found in the base cache, and then evicted from the base cache (into the victim cache) while the desired entry from the victim cache is moved back to the base cache. (The desired cache line, in the victim cache, is swapped with another line, in the base cache.)

If the memory reference is not in either the base cache or the victim cache, we get it from memory and put it in the cache.

**Example**

Suppose we have a cache description (lru\_definition):

{ name=lru, line\_size = 128, entries = 1024, ways=4, policy=lru }

or

{ name=lru, line\_size = 128, entries = 1024, ways=4, policy=lru, victim=0 }

For memory trace file test\_trace0, we get the following:

> cachesim lru\_definition test\_trace0 lru: 1024 entries of lines of 128 bytes; 256 sets of 4 ways, write-back, LRU 200119 addresses (149800 Fetch, 38425 Load, 11894 Store) 202606 hits (99%), 1638 misses, 1638 memory reads, 111 memory writes 99 dirty cache lines remain

Adding a victim cache is just changing the definition to

{ name=lru, line\_size = 128, entries = 1024, ways=4, policy=lru, victim=4 }

and now we get

> cachesim lru\_definition test\_trace0 lru: 1024 entries of lines of 128 bytes; 256 sets of 4 ways, write-back, LRU 200119 addresses (149800 Fetch, 38425 Load, 11894 Store) 202606 hits (99%), 1638 misses, 1638 memory reads, 109 memory writes 99 dirty cache lines remain 4 victim cache hits

**A Change in the Output Specifications**

There were questions raised by the analysis team about the statistics printed above, and to keep the customer happy, we want a change in the output to:

200119 addresses (149800 Fetch, 38425 Load, 11894 Store) lru: 1024 entries of lines of 128 bytes; 256 sets of 4 ways, write-back, LRU lru: 204244 accesses, 202606 hits (99%), 1638 misses, 1638 miss reads, 111 miss writes lru: 99 dirty cache lines remain

This separates out the trace statistics (the first line) from the description of the cache (the second line), and it's primary statistics (the 3rd line) and final state (the 4th line). We added the total number of accesses to the cache, since this is not the same as the number of addresses in the trace; some of those addresses span two cache lines, and so we get more cache accesses than addresses.

We want the same level of information about the victim cache, so if we run the same as the above with a 4-entry victim cache, we get

200119 addresses (149800 Fetch, 38425 Load, 11894 Store) lru main cache: 1024 entries of lines of 128 bytes; 256 sets of 4 ways, write-back, LRU lru main cache: 204244 accesses, 202606 hits (99%), 1638 misses, 1634 miss reads, 112 miss writes lru main cache: 99 dirty cache lines remain lru victim cache: 4 entries of lines of 128 bytes; fully associative, write-back, FIFO lru victim cache: 642 accesses, 4 hits (0%), 638 misses, 638 miss reads, 110 miss writes lru victim cache: 1 dirty cache lines remain

where we have printed the same information and statistics for the victim cache as for the main cache.

**Another set of Cache Code**

The employee that created the cache code that you were originally given (cache.tar above), had been assigned to create a cache simulator for multiple caches (like L1, L2, and L3), but left the company before finishing that project. The code you were given (cache.tar) was the code that he had checked into the corporate source control system. Since then, we've found code on his workstation that was not checked into the code database. This newer code is at [new\_cache.tar](http://www.cs.utexas.edu/users/peterson/src/new_cache.tar).

**Test Cases**

The cache simulator takes valgrind memory traces as input. You can produce your own trace files by:

valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes {command to trace} ...

for any command that you want to trace. There are sample trace files in [peterson/memory\_trace](http://www.cs.utexas.edu/users/peterson/memory_trace). Be careful, these can get large quickly. Our samples are between 3 and 6 Megabytes.

**Extensions**

Extensions to consider:

* How would you extend this code to allow chains of caches, such as an L1, L2, and L3 cache structure. What if it were a tree or network of caches instead of a simple chain, like multiple L1s which feed into multiple L2s which talk to multiple L3s?
* If you did extend it to a tree or network of caches, how would you handle cache coherence?
* The current code uses memory references as a measure of time -- time is counted in memory refrences. What if you introduced a separate sense of time, and defined the amount of time it takes to move items from one cache to another. How would that change the simulation?
* The current code does not actually move the memory data from memory to the cache and back. What would be gained, if anything, in doing that?

**Submission**

You need to submit files to create an executable called cachesim for program6. More specifically, for submission you should use

turnin --submit zxy program6 Makefile \*.h \*.c

Due Date: 24 November 2013