Conrado Uraga

Project 3 Cache Simulator

This is a Cache simulator that simulates a cache read for L1, L2, and/or L3. At the startup, if the user inputs the incorrect amount of arguments or just –h as the first argument, it will tell the user how to use the program. Once the user specifies how the sizes, replacement policy, and trace file, it will run the program. This works for simulating only L1, only L1 and L2, and all three caches. Once it calculates the hits and misses for specified caches, it will print them out and then free every memory that was allocated. Run time should not take more than 10 seconds even when doing all Fully associative caches and reading 1 MB of traces.

The way this program works is rather simple. Each cache is essentially a struct called the_cache in the cache_sim.h. It records how many misses, hits, memory accesses, contains the type of replacement algorithm, and the actual data structure for the replacement algorithm(only if it's a fully associative cache will actually use it), a link to the next cache level(if any), and then it holds a dynamically multi-dimensional array of cache_array structs. Cache_array struct contains the validity bit, the tag, the index, the offset, and the actual data structure for the replacement algorithm (only for set-associative cache).

For the replacement algorithm, I chose to go with double linked list for the LRU policy. When we call insert, it goes through the whole list and if the item is found, we delink it and update it to move to the head. If not, it still goes to the head, a rather simple procedure. Once we want to evict the least recently used item, we call getLRU and it returns the tail of the list. For FIFO, I created an array queue. The array is dynamically allocated and takes the size of the associativiy of the cache if it's an n-way set associative cache or it takes how many lines(E) of a fully associative cache. This is all taken care of in the initialize set function which checks what kind of cache it is.

Once we're done creating the cache, we call the respective functions to fill the cache which are: direct, SA, FA. The way we record the misses and hits are easy. Cold misses if the valid bit was 0. If the valid bit was 1 and the tag didn't match, we record a miss. If the tag was a match, we record a hit. To find the capacity miss and conflict miss, we have a Parallel L1 cache that is a FA cache with an LRU policy. Since a FA cache has total misses and cold misses computed, we can easily get the capacity misses. It's just TA(total misses)-L1'sCM(cold misses). Once we have that, L1's total misses-L1's capacity miss- L1's cold misses will give us the conflict misses.