COE 1541 Project 2: Cache Simulator

Spring 2020

Due: Friday, 4/24/2020

1. Introduction

This is an individual project.

In this project, you will build a highly parameterized and configurable cache. You will be given an input stream of cache accesses, and output 1) the finish time of each read access, 2) the hit and miss rate of each cache layer, 3) cache status image of all layers.

2. Basic cache model

The cache model you will implement is required to be:

1) Hierarchical and inclusive: all the data in the lower-level cache are required to be in the higher-level cache (e.g. all the data in L1 cache have to be in L2 cache).
2) Set-associative: Multiple cache lines are mapped to the same cache set. When cache receives an access, it needs to first use the index bits in the data address to find the target cache set, and then compare the tag bits of the address with the tag bits of all the valid entries (cache lines) in the target cache set. The position of index bits and tag bits in an address is shown as following (also in the textbook), the number of bits used for indexing depends on the total number of cache sets.

Tag	Cache Set Index	Block Offset

3) Using **LRU** as replacement policy.
4) Memory access latency is 100 more cycles than the latency of the last level cache.

3. Configurable parameters

The following parameters of the cache model should be left configurable:

1) Number of cache layers, e.g., 1, 2, 3
2) Size of each layer of cache in bytes, e.g. 32K for L1, 2M for L2, 4M for L3.

3) Access latency of each layer of cache in cycles, e.g. 1 cycle for L1, 50 cycles for L2.
4) Block size in bytes, this should be consistent among all cache layers, e.g. 64.
5) Set associativity of each layer of cache, e.g. 4 for L1, 8 for L2.
6) Write policy and allocation policy. Write policy includes write back, write through, and write eviction. Allocation policy includes write-allocate and non-write-allocate. To simplify the project, we only require write-back with write-allocate, and write-through with non-write-allocate.
    a. For write-back+write-allocate, data is only written to memory when it's evicted from cache. If a write access hits in cache, it can just write the data into cache and mark it as dirty. Hitting in higher level cache will bring this line into lower level cache, and only writes to the copy in lower level cache. If a write access misses in cache, it will allocate space in cache for this new cache line (may cause cache eviction if the set if full) and then write the data into the new allocated space in cache.
    b. For write-through+non-write-allocate, data is written to all levels of cache (if hits) and also memory. For example, for a 2-level cache, when there is a write access to addr0, you need to first check L1, if addr0 is in L1, write to L1 (If not found, nothing needs to be done); then check L2, if addr0 is in L2, write to L2 (If not found, nothing needs to be done); then write the data to memory. In this case, no new space is allocated in any level of cache.
7) Max number of outstanding misses (only for the *access-under-misses* version), i.e. the number of data reads that cache is waiting for memory to respond. When the maximum is reached, cache cannot serve new accesses anymore.

4. Description of cache operations

You will build a cache hierarchy defined by a given setup configuration. The input to this cache hierarchy is a stream of memory accesses with the format of, "op address arriving_time ", e.g.:

r 12345_{10} 1
w 65432_{10} 3
w 54321_{10} 7

The caches are initially empty, and filled by the input accesses. The cache should contain all the control bits as discussed in class: valid and dirty bits.

For input memory access streams, first implement a *sequential* version, meaning that the next memory access does not start until the previous access is returned (including misses). Then implement an *access-under-misses* version. In this version, while a cache is waiting for a miss to return, it can continue to serve subsequent requests. As subsequent requests may also miss the cache, there is an upper bound of how many outstanding misses this cache can handle. Implementation wise, you may use a buffer to save those outstanding misses until they return from lower level caches/memory. A cache is single-ported, meaning that it can be probed or filled one at a time. For example, when a miss returns and if the cache needs to be filled, it needs to wait till after the current request finishes probing due to a structure hazard.

5. **Inputs**
   A stream of memory accesses with the aforementioned format

6. **Outputs**
   1) The finish time of each read access. Note: For cache miss, we don't count the latency of filling each level of cache. The access is treated as "finished" after checking each level of cache and the memory access.
   2) Cache status image of all layers (you can output dirty entries only)
   3) Hit and miss rate of each cache layer.