Matthew Kozak

EGRE 426, Lab 5

11/21/2019

Implementation

The program for this lab assignment was chosen to be written in C, as VHDL was recommended. The very first thing implemented was command line argument configuration in order to specify the parameters necessary to chance the cache for testing, discussed later. The program specified how many parameters it needs at runtime. Next, file intake was managed. The program will check if the address file specified by a ‘.txt.’ is actually present in the directory of the executable. If it is not, it will inform the user. To finalize the initial error checking, the code checks the associativity and the block size to confirm that both are equal to a power of 2. If either of these are not a power of 2, the program will let the user know what needs to be resolved.

The initializations of the code begin with determining the bits necessary for an offset. This is done by taking the natural logarithm of the block size input by the user, then dividing that by the natural logarithm of 2. The result of this rounded up provides an integer representation of the number of bits required to represent the offset required. The same formula is used to determine the bits for the index. However, when calculating this value, the program will also remove a number of bits based on the number of bits needed for the associativity. For example:

* For ‘addresses.txt’, there are 10000 elements, which requires 12 bits to be represented
* If an associativity of 2 is selected, this reduces the result by 1
* If an associativity of 4 is selected, this reduces the result by 2, etc.

This result (‘b\_min’ in the code) is then used to produce a mask called ‘shift\_temp’. This mask is used against the address read in to determine the index. After these values are calculated, the arrays to hold the tag, valid bit, count (for LRU), and the index (e.g. data for Step 1 cases) are generated, and cleared to be all zeros.

The body of the code begins with a main ‘for’ loop that is waiting until the end of the file has been reached. Each time it increments to the next line of the file using ‘fscanf’. The tag is the most significant btis, without the index and the offset, so the entire address (‘curr\_addr’) is shifted right by the sum of number of bits needed for the index and the number of bits needed for the offset. The current index is the current address ANDed with the mask created, then shifted over by the number of offset bits to extract just the index. The current block address is calculated as the current address divided by the block size

Another ‘for’ loop is implemented to work through the associativity. Each time a new address is read in, the loop checks all values in that index/set to see if the tags match. If any of them produce a hit, the hit count is incremented. Additionally, if the setting is LRU and not random, the count for that element is reset to one, and the count for the other elements in the index/set are incremented by 1. If there is a miss, a flag is set and a separate method is implemented. This could have been implemented din its own function for improved clarity.

Finding a miss will first increment the miss count, then go through two different actions based on whether LRU or random placement was selected. If random was selected, the program will check the index/set and see if any elements are invalid (valid bit == 0) and set a flag if so. A while loop will then generate a random value for an element in that set that is not valid. It will set that value as valid, add the tag in, and clear the flag for the while loop.

If LRU is selected, the miss handler will increment the count of all valid elements in the index/set by 1. Then, each element in the index/set is checked and if the valid is not set, then the new tag will be placed at that location, and the count will be reset to 1 for that element. If no elements are invalid, then the program will look for the element with the highest count value and place the new tag there.

Finally, the code will print out the cache table if Step 1 in the report is currently being tested (number of elements == 16). For all cases, the code prints the cache size in KB, the number of reads, hits, misses, and the rates for the hits and misses based on the number of reads.