Andre Yates
4/30/2018
cs211

Cache Simulator Algorithm

I used a struct, containing an integer value and a 2d array of integers, to represent my sets and each the 2d array of ints to represents the lines of the set an int variable the index. I then used an array structs to represent all of the sets contained by my cache. First I extract all of the data that was inputted into the command line and use that to formulate all of the variables needed for my cache like the cache size, number of sets, number of bits per set, tag, and block offset, and the number of lines per set. Next, I start to read the file and do all of my operations on the cache. I check the program counter to see if it is at the end of the file. If it is not then I will read the rest of the data from the text file like the operation (read or write) and the actual address. I then extract all of my tags and set index from the address and then read or write to the cache using this information. I have a function that loops through the cache and searches for the tag that we extracted from the address in the set also specified by the address. If the function returns a 1 then the address is found in the cache and this is a hit and if it is not found in the cache it returns a 0 and results in a miss. After this, the corresponding variables are incremented. Memory read values are incremented if the search results in a miss and afterward, the address is loaded into the cache and since this is a write-through cache simulation the memory write variable is incremented if the operation was “write”, otherwise it is left alone.

I insert my addresses with a function that takes the replacement policy, the info of the address you want to insert, and the cache you want to insert into. My line is represented by a row of 3 columns. One column for the valid bit, the tag number and the place or age of the line. The function finds the set you want to insert into with the set Index and then if the replacement policy is “fifo” it will look for the line with place equal to 0 and place the new address in this line. The line with place equal to 0 represents the line that was inserted first. After inserting it sets the place of the newly inserted address to E which is the total number of lines in a set or in this case the youngest line in the queue. After this all of the places of the other lines in the set, except for the new line, are decreased by one setting the line 1 to line 0, line 2 to line 1, etc. which advances the queue. If the replacement policy is lru then the load function looks for the place that has the biggest value which would be E and then sets that value to 0 which represents the most recently used line. After it is loaded and set to 0 1 is added to all of the places of the other line. The bigger the spot the least used the line is. Lru also inserts data differently then fifo. In fifo the data is just inserted and the place value increases as the number of filled lines in the set increases creating a natural queue as the set is filled but the lru insert changes the new lines place value to 0 and increases all the other ones by one to stay with the lru convention previously explained.
Prefetching has its own separate values for hit, miss, etc. and occurs after the non-prefetching. It just takes the address and adds the block size to it and checks if the resulting address is in the cache or not and collects the data accordingly.

Questions

1.

Based on my data from my program it seems like the cache hits increase and the memory reads decrease. I think this is due to the concept of temporal and spatial locality. Memory entered around the same time and in the same place is most likely to be used together so if we load in the data asked for it is likely that the data next to it is also going to be asked for in the future. If we add this extra data into the cache before the CPU asks for it when it does it will result in more hits even if it is asking for the first time. With prefetching, there is a higher chance the data you are looking for will already be in the cache and result in a hit and allows the CPU not to have to go all the way to main memory to read the data.

2.

It would not be too difficult to add a cache layer to my program. What I would have to do is create another array of sets and call it my L2 cache. Then when checking for an address in the cache I would first check for it in the L1 cache and if it is not there then I would check for it in the L2 cache which would act like my cache simulator now and look for the address in the memory if it is not found in the L2 cache. If it is found in the L2 cache it will put it in the L1 cache as well.

3.

To make a cache that does not write to memory immediately I would take the data that has to be written and write it straight into the cache and then create an array of this data to keep track of the addresses that have to be inserted into main memory at some point. Then, when the time calls for it, I would take the array and load all of the addresses into main memory at the same time and empty the array so I can add more data and I am not putting old data into main memory.