Assignment 2 Report

Shuxin Lin(s1469577)

Internal Structure

The main function has four inputs: trace file name, cache size, associativity(number of ways) and allocation policy. If there are not enough arguments, print warning message. Otherwise, call cacheSimulator method.

cacheSimulator

The method receives four arguments and works as followed:

- Step 1: Open the trace file and check the four arguments' validity
- Step 2: Calculate the exact bit number for tag, index , block offset of one address
- Step 3: Simulate cache structure using LRUCache class
- Step 4: Initialize miss variables
- Step 5: While next line is not null, read one line at a time
- Step 6: Extract command(R or W) and address(hex) from each line
- Step 7: Extend address if the length is not 12
- Step 8: Decode tag and index from raw address (tag is a String and index is an Integer)
- Step 9: Search tag in the exact set in cache to decide if the operation hits or misses
- Step 10: Update miss variables
- Step 11: Except the condition that write misses when allocation policy is write no-allocate, update cache information based on LRU policy
- Step 13: Go back to Step 5 until the file come to the end
- Step 14: Calculate and print three miss rates

LRUCache

cacheSimulator constructs a cache structure to simulate the cache performance. The cache structure is actually a [setNum]*[associativity] array. In each set, I allocate one LRUCache instance to simulate LRU policy. LRUCache is actually a variant of linked hash table. There are two attributes in each elements of LRUCache: key and value. In this assignment, I do not actually use value but we can regard it as the actual block code. Key is tag of address, which is the ID of the block.

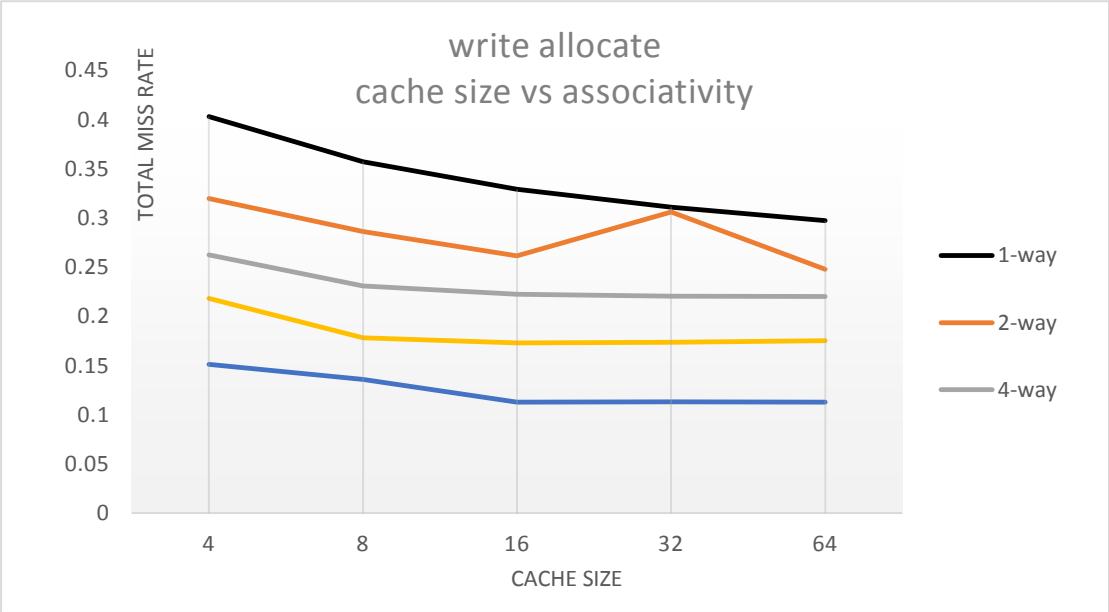
Besides, the original LRUCache class cannot meet my requirement completely. I overwrote one method removeEldestEntry like this:

```
protected boolean removeEldestEntry(Map.Entry<String,String> eldest) {
 return size() > this.capacity;   //capacity = associativity
}
```

And also I added one method isKeyExist to find out if the tag exists by comparing the tag with each keys.

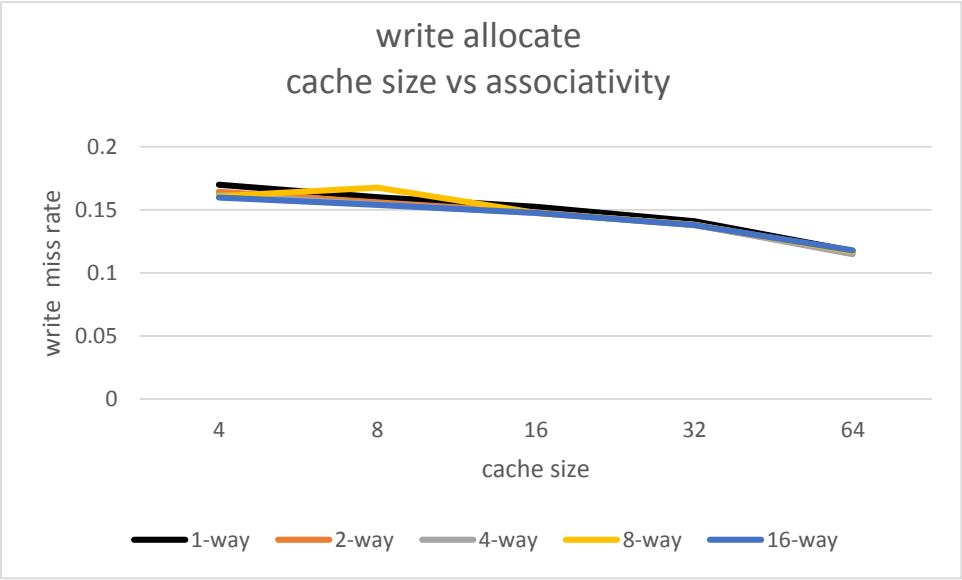
```
protected boolean isKeyExist(String key) {return this.containsKey(key);}
```

Result Summary



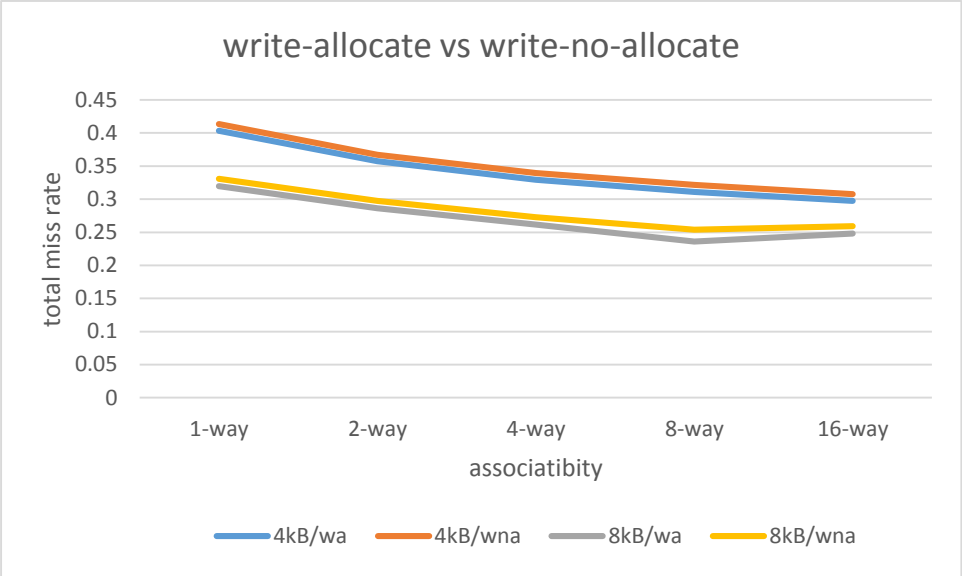
As can be seen from the graph, when the cache size increases, the total miss rate decreases. And when the associativity increased, the total miss rate decreases

Conclusion: The large cache size and associativity increases, the total miss rate decrease. But the decrease rate does not change too much if the cache size reaches 16KB. So 16KB is a good option for decreasing miss rate.



As can be seen from the graph, when the cache size increases, the write miss rate decreases. And

when the associativity increased, the total miss rate decreases.
Conclusion: The large cache size and associativity increases, the write miss rate decrease. But the decrease rate does not change too much if the associativity increases. But it may be useful when the trace file is quite large.



As can be seen from graph, when allocation policy is different, the miss rate changes. The write allocate performs better than the write no-allocate.