Lab report COL 216

Mohit Dua 2022CS51745

PART A

The motive of part A is that given a trace file we have to simulate a cache with the given parameters and output the number of loads, stores, hits, missess and the number of cycle times for the corresponding simulation.
To do this I have made two classes

- The FIFO class -

  It simulates a fifo cache set of given size evicting in fifo order
- The LRU class-

  It simulates a lru cache set of given size evicting in lru order

By making classes I can simulate a set by a class, so the whole cache can be simulated by a vector of different classes each representing a set such that the size of the vector will be the number of sets and the vector can be used to represent the cache.
Now running the code outputs the loads, hits and cycle time. The number of loads and hits should come out same for almost everyone but the calculation of the number of cycles used depends on an indivisual's interpretaion.
The interpretation for the number of cycles used by me for various cases are as follows
here 100(*) = 100(block size /4)

1. write through and write allocate
   - for a load hit we just load the data from the cache so it takes 1 clock cyle
   - for a load miss we first load the data from the memory to the cache and then pass it on from the cache so it takes 100(*) + 1 clock cycles
   - for a store hit we first write in the main memory so 100 clock cyles then write in the cache so 1 clock cycle so in total 101 clock cycles
   - for a store miss we first write in the main memory and then load it in the cache so 100 + 100(*) clock cyles

2. write through and no-write allocate
   - for a load hit we just load the data from the cache so it takes 1 clock cyle
   - for a load miss we first load the data from the memory to the cache and then pass it on from the cache so it takes 100(*) + 1 clock cycles
   - for a store hit we first write in the main memory so 100 clock cyles then write in the cache so 1 clock cycle so in total 101 clock cycles
   - for a store miss we just write in the main memory so 100 clock cycles

3. write back and write allocate
    - for a load hit we just load the data from the cache so it takes 1 clock cyle
    - for a load miss we first load the data from the memory to the cache and then pass it on from the cache so it takes 100(*) + 1 clock cycles
    - for a store hit we just write in the cache so 1 clock cycle
    - for a store miss we first bring the data from the main memory and then write it in the cache so 100(*) + 1 clock cycles
    - In case of write back there is an additional parameter called the write-backs which measures the number of times we write back to the memory in case of a miss when a memory block is evicted which is not counted in the cycle times of load and store misses so if in the case of an eviction the dirty bit is 1 we write back to the memory which takes 100(*) clock cyles. This is addedd to the total number of clock cycles additionally

4. write back and no-write allocate
    - or a load hit we just load the data from the cache so it takes 1 clock cyle
    - for a load miss we first load the data from the memory to the cache and then pass it on from the cache so it takes 100(*) + 1 clock cycles
    - for a store hit we just write in the cache so 1 clock cycle
    - for a store miss we just write directly in the main memory so 100 clock cycles
    - In case of write back there is an additional parameter called the write-backs which measures the number of times we write back to the memory in case of a miss when a memory block is evicted which is not counted in the cycle times of load and store misses so if in the case of an eviction the dirty bit is 1 we write back to the memory which takes 100(*) clock cyles. This is addedd to the total number of clock cycles additionally

The clock cycles for tasks such as checking hit/miss , checking the dirty bit etc. Are ignored since in the assignment pdf only cycle times for load and stores are considered.

PART B

Now considerting part A simulates a cache correctly and returns the number of clock cyles and the number of hits and misssess, the task for part B is to calculate the best cache configuration for a given trace file. To do this I just run the command for part A by varying the parameters in a loop, calculating 3 parameters : hit rate, clock cyles and cache size to be considered to compare caches.
The effectiveness is considered as the number of clock cycles used + (0.1)*cache_size. The best configuration is given as the output.

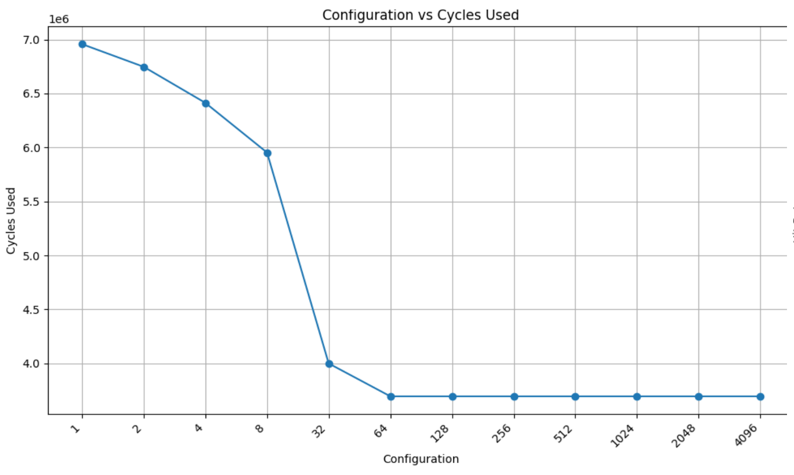
Reason for choosing the given parameter for effectiveness:

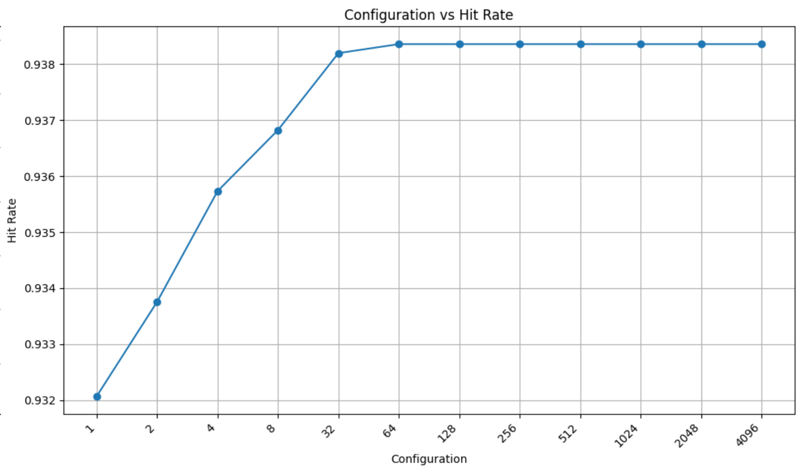
According to the problem statement given in the pdf the access time for the memory and the cache is constant and is not dependent on the size of the caches. This creates a huge problem as in reality the cache size affects the access time and this is the reason as the cache size becomes larger the design the access time becomes slower taking more number of cycles for read and write also the cost of the cache increases, but in our case just taking a larger cache will give better effectiveness in all aspects including hit rate and the number of cycles used.
This is the reason why the term for the cache size is added in the term to measure effectiveness. Hit rate does not work as a good parameter for taking in account the effectivenesse of a cache as a bigger cache will always have better hit rate but the number of cycles takes in account the hit rate and the miss penalty and so is dependent somewhat on the size of the cache, hit rate and very much on the specification of the cache namely write-back/write-through write-allocate/no-write-allocate lru/fifo.
But to account for the cache size as for the reasons mentioned above the term (0.1)*cache_size is added. This in no way accounts for the effect of cache_size in real caches but is just a term to add the significance of smaller cache size.
The best configuration after running the tests on all the given trace files is
Cache Size: 64 sets
Associativity: 1024
Block Size: 4 bytes
Write Policy: write-allocate
Write Strategy: write-back
Replacement Policy: LRU

Effect of varying parameters on cycle time and hit rate

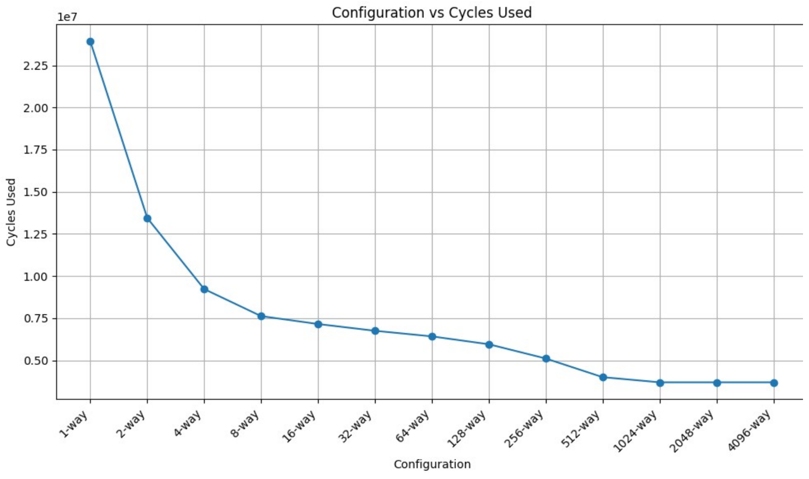
These are some plots which represent the effect of changing one parameter while keeping others constant. These are plotted by simulating the cache on the gcc.trace and varying the given parameter and keeping others same as that in the configuration of the best cache.

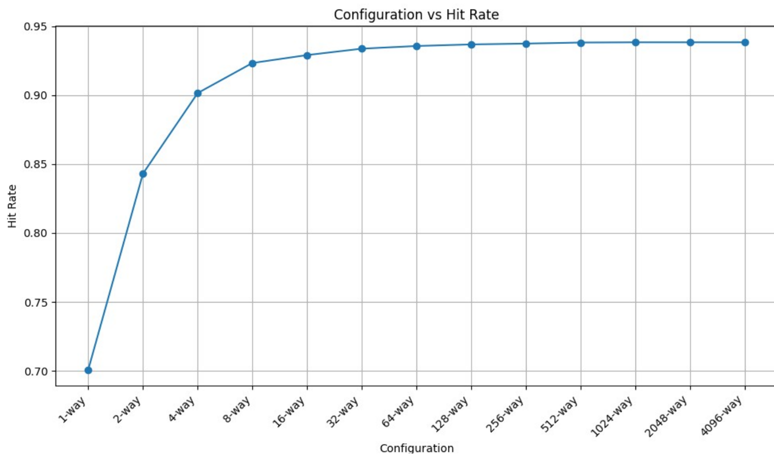
- Cache Size



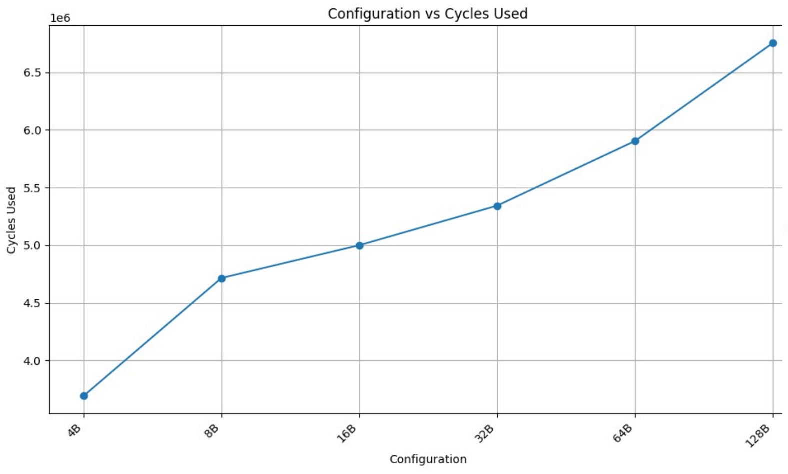


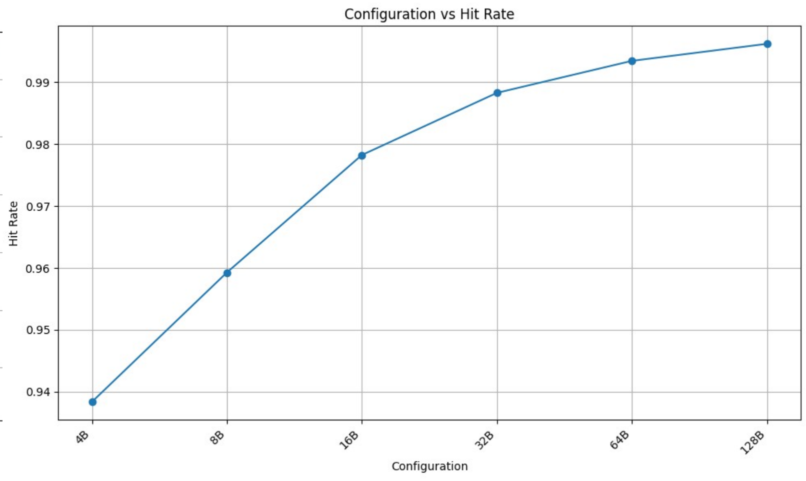
- Associativity



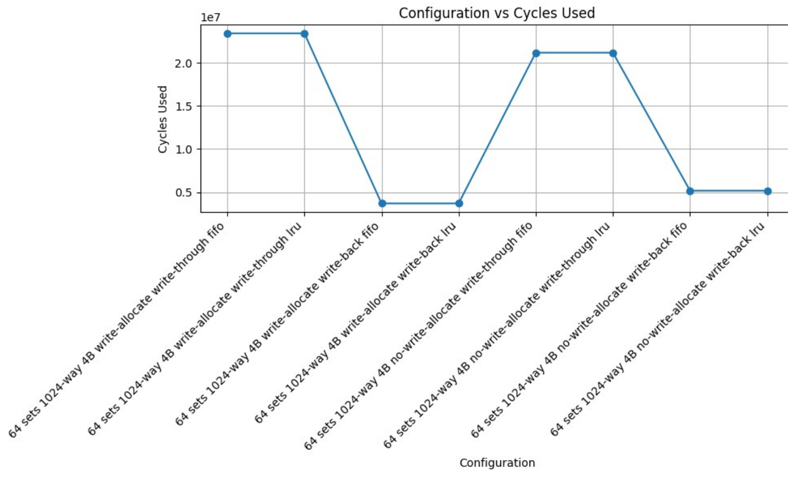


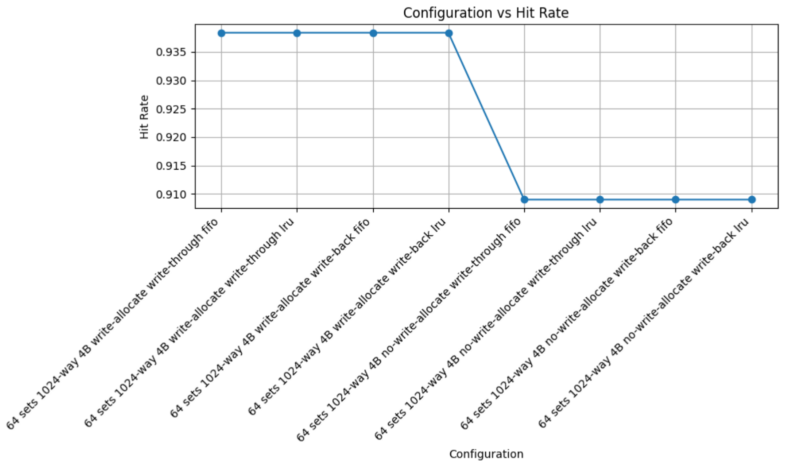
- Block Size





- Effect of other parameters





Scripts Used for testing and plotting

part_b.cpp is the code to return the best cache configuration for any trace file.
it iteratively runs the code multiple times for various configurations and returns the best configuration.
partb_2.cpp is the code for running the simulator by changing specific parameter and writing its result in a file named analysis.txt which is then read by the plotter.py which plots the graphs. This is helpful in visually comparing the effect of change in a parameter and helps in finding the best configuration.