COL 216 ASSIGNMENT 3

CACHE SIMULATOR

SATWIK

2022CS51150

The purpose of this assignment is the create a cache simulator with various specifications including different write-policies(write-through and write-back), write-allocations-policies(write-allocate and no-write-allocate) and evict policies(LRU eviction and FIFO eviction).

After writing the code for the cache simulator, I used the simulator to measure the performance of the cache by varying the input parameters. The trace used by me is that of a real processor and hence provides accurate data(gcc.trace).Here are the results:

1. Changing the write-policy:

There are two types of write-policies used in the simulator

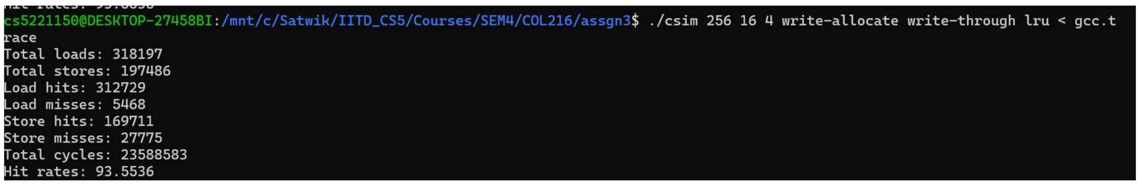
(a) Write-through
(b) Write-back

It was observed that for the same cache specifications, write-back provides better throughput(lesser number of cycles), and also has a better hit rate.

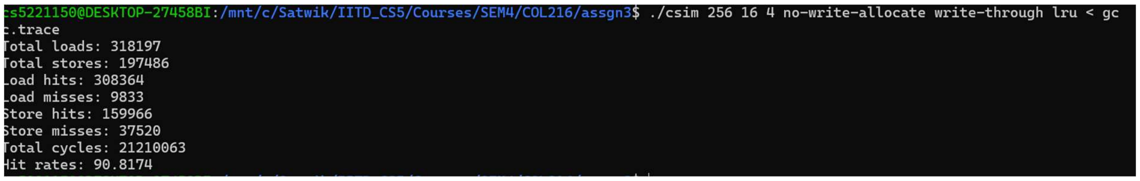
So in general, write-back cache has a better performance than a write-through cache.

Also, if we consider write-through cache, then the cache with no-write-allocate policy has a better performance than the one with write-allocate policy.

This is because the cache with no-write-allocate has significantly better lesser number of cycles despite lesser hit rate than the the cache with write-allocate policy.



```
cs5221150@DESKTOP-27458BI:/mnt/c/Satwik/IITD_CS5/Courses/SEM4/COL216/assgn3$ ./csim 256 16 4 write-allocate write-through lru < gcc.t
race
Total loads: 318197
Total stores: 197486
Load hits: 312729
Load misses: 5468
Store hits: 169711
Store misses: 27775
Total cycles: 23588583
Hit rates: 93.5536
```

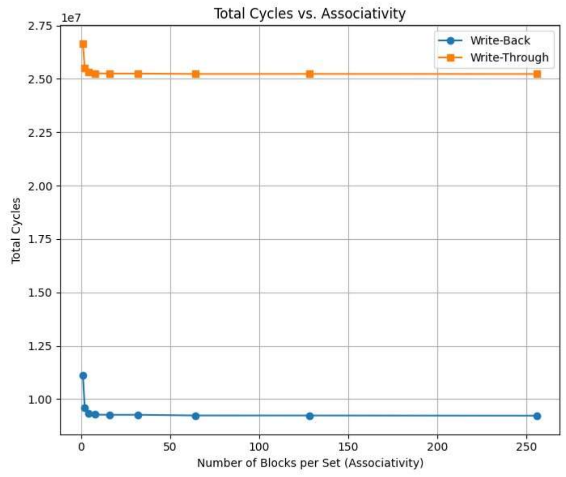


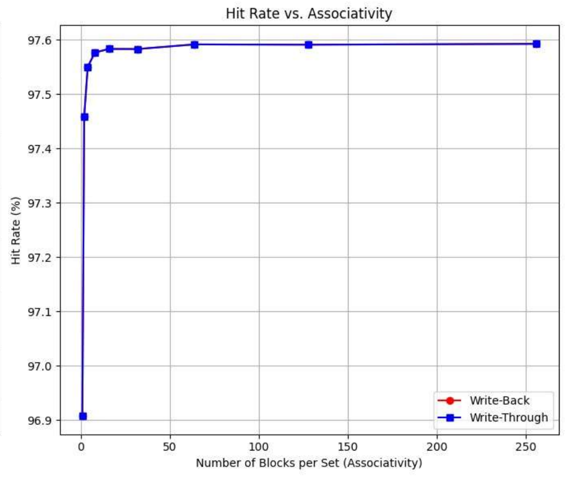
```
cs5221150@DESKTOP-27458BI:/mnt/c/Satwik/IITD_CS5/Courses/SEM4/COL216/assgn3$ ./csim 256 16 4 no-write-allocate write-through lru < gc
c.trace
Total loads: 318197
Total stores: 197486
Load hits: 308364
Load misses: 9833
Store hits: 159966
Store misses: 37520
Total cycles: 21210063
Hit rates: 90.8174
```

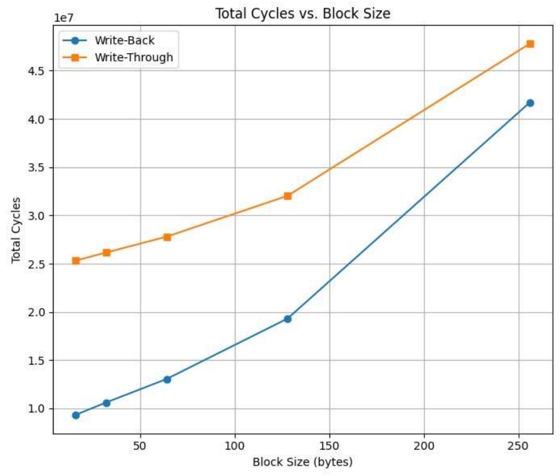
The simulator also has provisions for varying the associativity, number of sets and block size of cache. In general it is seen that increasing the associativity of the cache reduces miss rate and also decreases the number of cycles. This can be seen in the graph plotted below for different set sizes.

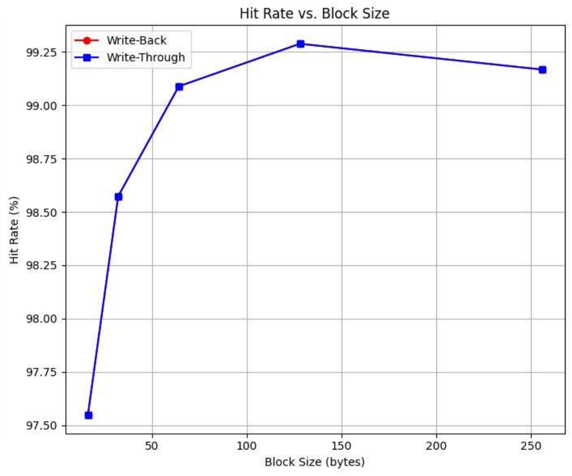
Also, increasing the size of the blocks tends to increase the number of clock cycles which is also evident in the graph plotted. However, for lower block size, the hit rate is lower than for larger block size.

So we prefer our cache to have more associativity and lesser block size to have better performance.





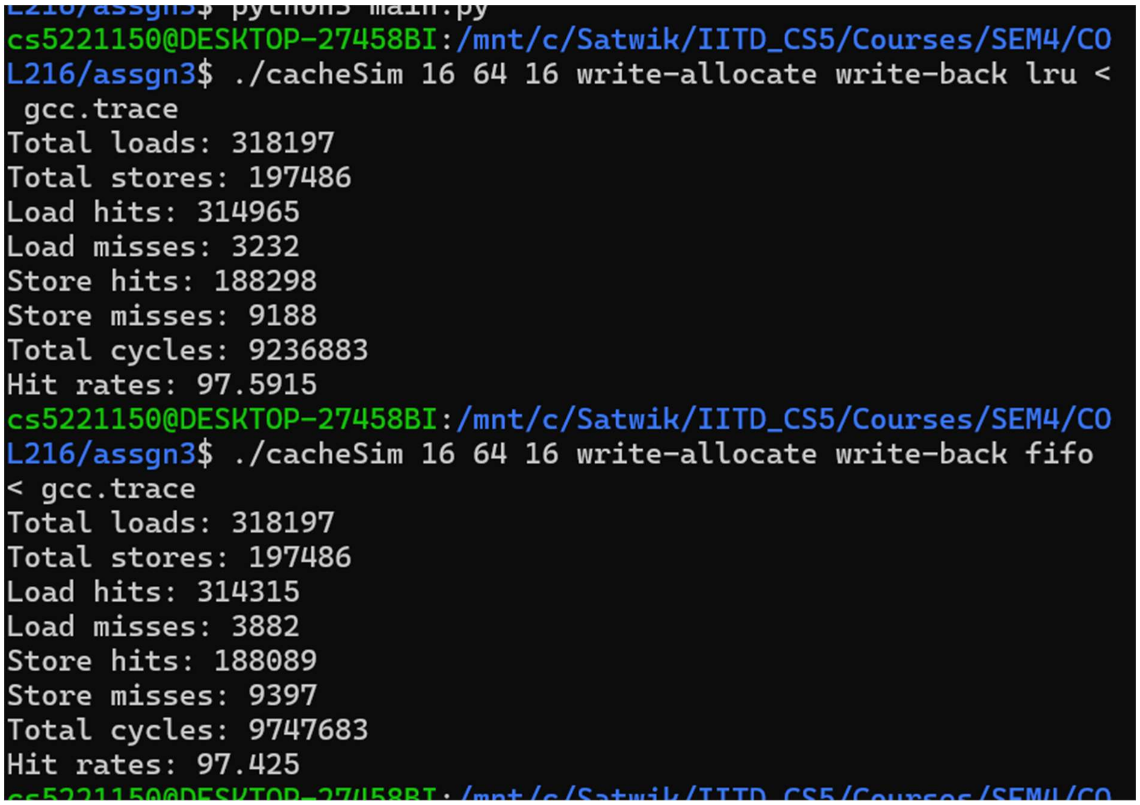




Lastly there is also provision for varying the evict policy i.e. is

(a) LRU and (b)FIFO evictions.

We observed that LRU eviction policy has lesser number of cycles and also a better hit rate and hence it is better.



```
L216/assgn3$ python3 main.py
cs5221150@DESKTOP-27458BI:/mnt/c/Satwik/IITD_CS5/Courses/SEM4/CO
L216/assgn3$ ./cacheSim 16 64 16 write-allocate write-back lru <
 gcc.trace
Total loads: 318197
Total stores: 197486
Load hits: 314965
Load misses: 3232
Store hits: 188298
Store misses: 9188
Total cycles: 9236883
Hit rates: 97.5915
cs5221150@DESKTOP-27458BI:/mnt/c/Satwik/IITD_CS5/Courses/SEM4/CO
L216/assgn3$ ./cacheSim 16 64 16 write-allocate write-back fifo
< gcc.trace
Total loads: 318197
Total stores: 197486
Load hits: 314315
Load misses: 3882
Store hits: 188089
Store misses: 9397
Total cycles: 9747683
Hit rates: 97.425
cs5221150@DESKTOP-27458BI:/mnt/c/Satwik/IITD_CS5/Courses/SEM4/CO
```

<u>Conclusion</u>

To obtain the best performance, we prefer that the cache should be write-back cache and have write-allocate policy. The eviction policy should be LRU. Also the associativity of the cache should be high. For eg- if the memory is of size 16KB, then the most optimal performance is found for a cache with block size 4, associativity 16, and number of sets 256.