COL216 MAJOR ASSIGNMENT

CHIRAG MOHAPATRA
2018CS50403

In this assignment , we had to simulate a cache as specifed in the description .
As according to piazza , the cache provides block level access .

My cache is an array of cache sets . Each cache set has two different categories : high priority and low priority blocks . Each block has a tag , data and a dirty bit .

Cache : A vector of cache sets

Cache set : Contains a vector of high priority blocks and a vector of low priority blocks .

Cache block : A tag , data and a dirty bit

Both the high and low priority blocks follow the LRU(Least Recently Used) Method .

Also , a block addressed for the second time after first call is promoted to high priority . If not accessed for T requests , it returns back to the Low Priority group and enters at the front of the group(as following LRU method) .

I have set the sizes of high and low priority blocks at run times i.e they are variable and are modified accoridng to need . This is to make efficient use of memory since static allocation may lead to memory being unused for long time .

The dirty bit in a cache block represents that the data has been overwritten after inserting the block from main memory . So , when the block gets out of cache , the main memory is updated .

Format of main memory:
Array of integers(all initially 0) ordered by block address , where block address is tag*number of sets + set_index

Input is given in for 1,W,2 or 0,R with any extra spaces where 0 and 1 refer to block addresses here .

Some test cases I have tested are:

S.No	Input	Remarks
1	16 2 2 4 1,W,1 5,W,5 6,W,6 1,R 0 , W , 0 5,R 7,W,7 3,W,3 5,W,50 7,R	Please run the program for complete details of output . Hit Ratio : 0.4 , some salient features of this case are: 1,R : 1 is promoted to high priority Since T is 4 , we observe that after 3,W,3 is executed , then 1 comes down to low priority . So movement between high and low priority is successfully checked . Also , we can observe that LRU is successfully maintaained .
2	16 2 2 4 0,W,0 0,W,1 8,W,8 0,R 2,W,2 3,W,3 4,W,4 16,W,16 8,R 10,R	Hit Ratio : 0.2 Some salient features: Observe the use of main memory with 8,R since we read 8 which means that the implementation of write-back policy is correct . At 16,W,16 , the block 4 is removed and at 8,R , the block 0 is removed , application of LRU is correct .
3	16 2 2 4 0,W,1 4,W,2 8,W,3 16,W,4 20,W,5 24,W,6 28,W,7 0,R 4,R	Demonstrating the use of main memory .Here , all the requests are from one set itself and we can easily check the correct working of our write-back policy with this example .

	8,R 16,R 20,R 24,R 28,R	
4	4 2 2 1 0,W,1 0,R 4,W,2 4,R 0,W,1 0,R 4,W,2 4,R	Clearly demonstrates the movement of blocks between high and low priority groups . We have only 1 set with 2 blocks and T=1 , so there is alternation between the two blocks among the groups .
5	4 2 2 10 0,W,1 4,W,2 0,R 4,R 0,W,3 4,W,4 0,R 4,R	Clearly demonstrates the LRU aspect in the high priority group since we can see the changes in order of the two blocks . Also shows usefulness of dynamic sizes of the high and low priority groups since we can see that most operations are happening solely in high priority group here .
6	4 2 2 10 0,W,1 4,W,2 0,R -4,R 0,W,3 4,W,4 0,R 4,R	Terminates after 0,R since -4 is an invalid memory input . The input memory should be >= 0 .

7	16 2 2 4 0,W,1 4,W,2 8,W,3 12,W,4 0,R 4,R 8,R 12,R 16,W,4 20,W,5 24,W,6 28,W,7 16,R 20,R 24,R 28,R	Specially doctored case to check effects of cache associativity . Also , easily simulates working of main memory and write-back policy .
8	16 1 2 4 0,W,1 2,W,2 4,W,4 0,R 6,W,6 2,R 8,W,8 4,R 6,R 8,R	Doctored case to check effect of block size
9	16 1 4 4 0,W,3 21,W,5 1,W,8 7,W,7 1,W,9 8,W,9	Hit Ratio = 0.214286 Step-by-step explanation : 1. 0,W,3 : Write miss and block added to set 0 low priority(tag 0) with data 3 . 2. 21,W,5 : Write miss and block added to set 1 low priority(tag 5) with data 5 . 3. 1,W,8 : Write miss and block added to set 1 low priority(tag 0) 4. 7,W,7 : Set 3(tag 1) , Write miss 5. 1,W,9 : Write hit , 1 moves to high priority in set 1

	17,W,6 17,W,7 13,W,1 16,W,3 9,W,6 8,R 20,R 21,R	6. 8,W,9 : Set 0(tag 2) , Write miss 7. 17,W,6 : Set 1(tag 4) , low priority , miss 8. 17,W,7 : Write hit , 17 moves to high priority in set 1 9. 13,W,1 : Set 1(tag 3) added , 1 comes down to low priority in set 1 10. 16,W,3 : Set 0(tag 4) , Write miss 11. 9,W,6 : Set 1(tag 2) , 21 is removed from set as Lru policy dictates 12. 8,R : Read hit , reads data 9 , 8 moves to high priority in set 0 , in set 1 , 17 comes down to low priority 13. 20,R : Set 0(tag 5) , reads data 0 , read miss 14. 21,R : Set 1(tag 5) added , 13 removed as Lru(and high and low priority settings) . Read data 5
10	4 1 1 1 10,W,10 0,W,10 1,W,9 2,W,8 3,W,7 4,W,6 5,W,5 6,W,4 7,W,3 8,W,2 9,W,1 1,R 11,W,9 0,R 1,W,10 0,W,9 2,R 11,W,8 1,R 2,W,10 1,W,8 3,R 11,W,7 2,R 3,W,10 2,W,7	Simulating a real program on cache . Here , I have represented bubble sort , a well-known sorting algorithm . The memory 10 contains n(10 here) while 0 to 9 contain our array which is initially 10,9,8,7,6,5,4,3,2,1 (inputed initially in first 10 lines) . After this , sorting takes place , 11 is a temporary variable used for swapping . Finally , we can see that the memory becomes 1,2,3,4,5,6,7,8,9,10 which is sorted in ascending order as was our purpose . The cache statistics are: Number of Accesses = 236 Number of Reads = 90 Number of Read Hits = 36 Number of Read Misses = 54 Number of Writes = 146 Number of Write Hits = 96 Number of Write Misses = 50 Hit Ratio = 0.559322 We successfully simulated bubble sort using our cache!!

	4,R 11,W,6 3,R 4,W,10 3,W,6 5,R 11,W,5 4,R 5,W,10 4,W,5 6,R 11,W,4 5,R 6,W,10 5,W,4 7,R 11,W,3 6,R 7,W,10 6,W,3 8,R 11,W,2 7,R 8,W,10 7,W,2 9,R 11,W,1 8,R 9,W,10 8,W,1 1,R 11,W,8 0,R 1,W,9 0,W,8 2,R 11,W,7 1,R 2,W,9 1,W,7 3,R 11,W,6 2,R 3,W,9	

	2,W,6 4,R 11,W,5 3,R 4,W,9 3,W,5 5,R 11,W,4 4,R 5,W,9 4,W,4 6,R 11,W,3 5,R 6,W,9 5,W,3 7,R 11,W,2 6,R 7,W,9 6,W,2 8,R 11,W,1 7,R 8,W,9 7,W,1 1,R 11,W,7 0,R 1,W,8 0,W,7 2,R 11,W,6 1,R 2,W,8 1,W,6 3,R 11,W,5 2,R 3,W,8 2,W,5 4,R 11,W,4 3,R	

	4,W,8 3,W,4 5,R 11,W,3 4,R 5,W,8 4,W,3 6,R 11,W,2 5,R 6,W,8 5,W,2 7,R 11,W,1 6,R 7,W,8 6,W,1 1,R 11,W,6 0,R 1,W,7 0,W,6 2,R 11,W,5 1,R 2,W,7 1,W,5 3,R 11,W,4 2,R 3,W,7 2,W,4 4,R 11,W,3 3,R 4,W,7 3,W,3 5,R 11,W,2 4,R 5,W,7 4,W,2 6,R 11,W,1	

	5,R 6,W,7 5,W,1 1,R 11,W,5 0,R 1,W,6 0,W,5 2,R 11,W,4 1,R 2,W,6 1,W,4 3,R 11,W,3 2,R 3,W,6 2,W,3 4,R 11,W,2 3,R 4,W,6 3,W,2 5,R 11,W,1 4,R 5,W,6 4,W,1 1,R 11,W,4 0,R 1,W,5 0,W,4 2,R 11,W,3 1,R 2,W,5 1,W,3 3,R 11,W,2 2,R 3,W,5 2,W,2 4,R	

	11,W,1 3,R 4,W,5 3,W,1 1,R 11,W,3 0,R 1,W,4 0,W,3 2,R 11,W,2 1,R 2,W,4 1,W,2 3,R 11,W,1 2,R 3,W,4 2,W,1 1,R 11,W,2 0,R 1,W,3 0,W,2 2,R 11,W,1 1,R 2,W,3 1,W,1 1,R 11,W,1 0,R 1,W,2 0,W,1	

Trying to gain some statistics . We try to adjust the different parameters of our program to get some statistics for our cache(Note : these statistics are input-dependent) .

Cache Associativity

Increasing cache associativity will increase our hit ratio , however it comes as a cost since we have to do more comparisons and so larger sets would be cumbersome and slower .
This increase is not so apparent however .

Consider test case 2 , then :

Cache associativity	Hit Ratio
1	0.1
2	0.2
4	0.2
8	0.3

Consider test case 3 , then:

Cache associativity	Hit Ratio
2	0
4	0
8	0.5

Our hit ratio might increase on increasing cache associativity(however , this is input dependent) since blocks have less chance of being deleted from cache .
However , there is an increase in number of comparisons .

Consider this doctored test case 7:

Cache associativity	Hit Ratio
2	0
4	0.25
8	0.5

Effect on test case 10(bubble sort):

Cache associativity	Hit Ratio
1	0.559322
2	0.559322
4	0.737288

Block size

Increasing block size decreases the number of sets and the number of blocks(since cache size is fixed) . Less blocks stored means hit ratio might decrease .

Consider test case 8:

Block size	Hit Ratio
1	0.5
2	0.3
4	0
8	0

Increasing block size means larger elements are stored in blocks , however , the hit ratio greatly reduces since the overall effect is less number of blocks .

T

If a block in high priority group is not accessed for T accesses , then it is demoted to low priority . The effect of change in T won't affect the hit ratio or final output per se , however this is input dependent . The main thing that would change is the inner workings of cache .

Very high T:- Most blocks would be in high priority
Low T:- Most blocks in low priority

Ideally for a practical system , T should have an appropriate mid value that ensures that blocks remain in high priority for proper amount of time .

The effect of T can be seen in the test cases provided .

Cache size

This has positive aspects mostly since cache accesses are much faster than memory accesses and so , increase in cache size would make the computer faster as a whole . However , cache memory is also highly expensive and it is very costly to increase cache size . Thus , cache size is not very large generally . Our efforts are all to get maximum efficiency at low cache sizes .

Effects of cache size on test case 7(associativity 2):

Cache size	Hit Ratio
16	0
32	0.25
64	0.5

Increase in cache size obviously increases the hit ratio .

Effect of cache size on bubble sort(associativity 1):

Cache size	Hit Ratio
4	0.559322
8	0.745763
16	0.949153

The effect of increasing cache size are apparent in this case .

Conclusion

The cache implemented has been tested and it runs correctly! I have analysed various test cases to show the nuances of te working of the cache .

I also analysed the various parameters of our cache and their effects in a practical scenario .

The size of high and low priority groups is determined at run time since the test cases show that this is more effective than static allocation as the running is affected by different cases and values of T .

Overall , the simulation of cache has been successful!