Home Work -4 Aditya Garg CS22BTECH11002

Observation About the Program Given

The program-1 iterates over the memory locations in a nested loops . It increments the address by 8 64 times in the loops .

The program-2 also iterates over the memory location in a nested loops however it increments the address 64 times in the inner loop and thus jumps 8 memory locations 8 times before going to the next memory location in the case of (8,8) and jumps 16 memory locations in (16,16)

Question 1

PROGRAM -1

CASE 1 Lines: 1,2,3 ; Blocks: 2 ; Ways: 0

dword (8,8):

Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 1 2 0 0.7424 49 17 66 2 2 0 0.7424 49 17 66 3 2 0 0.7424 49 17 66

Observation -- The hit rate , number of hits , number of misses and total access remain same in all cases . The reason being the way the code works . Since the code only iterates over the address by incrementing 8 everytime "spatial locality" is being used here , however "temporal locality" fails because no data present in cache is being reused here . Benefit of Spatial locality is achieved by increasing the block size . The number of lines in cache affects the temporal locality as it helps in storing a previous required data in the cache . Since the block size remains the same , all parameters too remain unchanged

dword (16,16):

Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 1 2 0 0.7481 193 65 258 2 2 0 0.7481 193 65 258 3 2 0 0.7481 193 65 258

Observation -- The hit rate , number of hits , number of misses and total access remain same in all cases .The reason is same as mentioned above . The only difference here is iteration is longer 4 times previous one (16*16) .HOwever this doesnt change the answer as we access memory in a iterative manner and never reuse the data .

CASE 2 Lines: 3 ; Blocks: 3,4,5 ; Ways: 0

dword (8,8):

Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 3 3 0 0.8636 57 9 66 3 4 0 0.9242 61 5 66 3 5 0 0.9545 63 3 66

Observation -- The hit rate and number of hits increases as the number of blocks increase . Since the code moves on contigious memory locations , "spatial locality" is used here . Having larger block size allows to have more nearby data in the cache resulting in more hits than before .

dword (16,16):

Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 3 3 0 0.87821 225 33 258 3 4 0 0.9341 241 17 258 3 5 0 0.9651 249 9 258

Observation -- The reason here too is very similar as stated above . The longer iteration doesnt affect the answer as we access memory in a iterative manner and never reuse the data . As a result replacement of data in cache doesnt result in future cache miss .

CASE 3 Lines: 3 ; Blocks: 2 ; Ways: 0,1

dword (8,8):

Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 3 2 0 0.7424 49 17 66 3 2 1 0.7424 49 17 66

Observation -- The hit rate , number of hits , misses do not change depsite increasing the number of ways is because increasing the number of ways decreases the chance of collision allowing more temporal locality . However the program 1 utilises the "spatial locality" . NO data is reused and as a result having more number of ways do not benefit for the given program .

dword (16,16):

Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 3 2 0 0.7481 193 65 258 3 2 1 0.7481 193 65 258

Observation -- Increasing number of ways does not increase the hit rate . The reason is same as mentioned above . The only difference here is iteration is longer 4 times previous one (16*16) .HOwever this doesnt change the answer as we access memory in a iterative manner and never reuse the data .

PROGRAM -2

CASE 1 Lines: 1,2,3 ; Blocks: 2 ; Ways: 0

dword (8,8):

Lines Blocks Ways Hit rate Number of hits Number of misses Total Accesses 1 2 0 0.0303 2 64 66 2 2 0 0.0303 2 64 66 3 2 0 0.04545 3 63 66 4 2 0 0.7424 49 17 66

Observation -- The hit rate , number of hits are very low as compared to program 1 . Increasing the lines from 1 to 2 doesnt cause any change while increasing to 3 marginally increases the hit rate . This happens because unlike program 1 , program 2 doesnt follow spatial locality for the given block size and does not access neighbouring locations to the current request . The program increments address by 64 in the inner loop i.e skipping 8 double words . As a result data is stored in alternative indexes and due to lower number of lines data present in a index is replaced very often . Because of all such reason the number of misses becomes very high . However after the inner loop executes , we comes to the next location which might have still been present in cache or could be present due to the memory access in the inner loop resulting in more hits . This is only possible is the number of lines (blocks) in cache are more which is also seen in the above case . Increasing the number of lines further can improve the hit rate because of larger number of lines in the cache , the number of replacement of data would be lesser increasing the chance of cache hit .

dword (16,16):

Lines	Blocks	Ways	Hit rate	Number of hits	Number of misses	Total Accesses
1	2	0	0.00752	2	256	258
2	2	0	0.00752	2	256	258
3	2	0	0.00752	2	256	258
6	2	0	0.7481	193	65	258

Obsevation -- The hit rate and number of hits get much lower in the case when the data stored is 16 . This is because the code jumps 16*8 bytes i.e 16 words in each iteration of the inner loop . Due to the lower number of lines (index) this causes a repacement in the cache at almost every load . If the number of lines are increased in the cache the number of hits can increase due to the lesser replacement of pre fetched values . As a result after an iteration of the inner loop no data gets replaced resulting in hits for the next load instruction and this gets repeated .

CASE 2 Lines: 3 ; Blocks: 3,4,5 ; Ways: 0

dword (8,8): Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses | ---- | ------ | ---- | ------ | ---------- | ---------- | ---------- | 3 3 0 0.8636 57 9 66 3 4 0 0.9242 61 5 66 3 5 0 0.9545 63 3 66

Observation -- The hit rate and number of hits increases as the number of blocks are increased from 3 to 4 and 5 . This is because the program 2 jumps 8 words ahead . Increasing the block size allows the such jumps to be present inside the same block or minimise the replacement of previously stored data in cache ensuring to increase the hit rate . Thus despite the code having jumps of 8 words due to a larger block size , spatial locality helps to increase the hits . ALong with this , it is possible that we access the same memory address again in program 2 i.e temporal locality also comes into picture .

dword (16,16):

Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses | ---- | ------ | ---- | -------- | ---------- | ---------- | ---------- | 3 3 0 0.00751 2 256 258 3 4 0 0.00163 3 255 258 3 5 0 0.8721 225 33 258

Observation -- The data observed is different from previous case . The hit rate is very low for the number of blocks 3,4 and signficantly increases for 5 . In every iteration we move 128 bytes i.e 16 words ahaead . Since the index bits are 6 to 8 in address after every 4 iteration of inner loop tag bits change while the index bits do not change . As a reult the previously stored data is replaced . In the next ieration of outer loop we fetch the next data howvever it has been replaced . Similar pattern is observed for 4 blocks . However for 5 blocks , the data is not replaced because the index bits are 8 to 10 , by incrementing 128 bytes , the index bit doesnt change resulting in a hit . In the next ieration of inner loop the index bits changed and same pattern is repeated . Since the data fetech is 16 times the cache containing 8 lines get filled and no replacement is required . After the inner loop executes , it fetches a double word which is already present in the cache due to no replacement in the inner loop . This is why the number of hits are very high for 5 blocks .

CASE 3 Lines: 3 ; Blocks: 2 ; Ways: 0,1

dword (8,8): Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 3 2 0 0.04545 3 63 66 3 2 1 0.7424 49 17 66

Observation -- The hit rate and number of hits significantly increase by increasing the number of ways . This happens because when the number of ways was 0 , there were multiple read to the same index causing the previous stored block data to be removed which was used later . As the number of ways increase to 1 i.e 2 ways in a single set , there was no need to remove the block data to replace resulting in more hits than miss .

dword (16,16): Lines Blocks Ways Hit rate Number of hits NUmber of misses Total Accesses 3 2 0 0.07752 2 256 258 3 2 1 0.07752 2 256 268

Observation -- The hit rate and number of hits are low despite increasing the number of ways from 0 to 1 . This happens because the program access memory after 128 bytes i.e 16 double words causing the index bit to jump 4 locations . Since the number of lines is 8 , the data stored in a index is replaced in the 2nd iteration following resulting in more misses in the case of ways being 0 . When the number of ways is increased , still the jumping of index keeps happening but a data is replaced at the 4th iteration following it and since we iterate 16 times , the data is gets replaced before we fetch it in the next ieration of outer loop .

Question -2

write back -- 1 write through -- 0

write allocate -- 1 write without allocate -- 0

PROGRAM -1

dword (8,8):

Wrt HIT Wrt MISS Hit rate Number of hits NUmber of misses Write Backs Total Accesses 1 1 0.7424 49 17 0 66 1 0 0.04545 3 63 62 66 0 1 0.7404 49 17 64 66 0 0 0.04545 3 63 64 66

Observation -- The program 1 never uses any previous address again incrementing address by 8 .

If with resepct to miss we choose write allocate , it fetches a block into cache . It uses "spatial locality" here . The further store instruction on the same index results in a hit . Since for other store instructions index is not same , there is no requirement to write back to the memory (no index is fetched again)and that is why write back = 0 .

If with respect to miss we choose without allocate , it writes to the memory and doesnt load it in the cache . Because of this every store instruction results in a miss as we are not storing the block if there is a miss . Due to the first load we have 3 hits and remaining all are misses because of without allocate with respect to miss . The number of write backs are more due to the number of cache misses involving write being large .

The performance again is better in the case of write through being chosen with respect to hit and write allocate with respect to miss . Due to the nature of the code we only access memory in a iterative manner as a result first miss causes it to fetch the block . For the following store in the same block if it is a hit , we update the value in cache as well as memory resulting in write back . For every block fetched in a new index it results in a miss followd by hits due to fetching of the block due to write allocate with respect to hit . Every store instruction causes a write back to the memory either if it is a hit or miss because of the policy . However the time taken increases as compared to write back policy and write allocate policy despite the same hit rate because of the large number of memory cycles .

If we choose write through policy and write without allocate policy , number of hits decreases as compared to write allocate policy . Again since the code reaches memory in iterative manner , for the initial miss we follow write through , fetch the data in cache and update in both memory ans cache causing write backs . 3 hits are seen because of the first load instruction a block of 4 double words are fetched resulting in hit for next 1 load and 2 store . The following store instructions are never fetched to the cache resulting in more misses and every store instruction be write back to the memory due to write without allocate policy .

The number of write backs in case of write through is more because if we have a hit we write back to both cache and the memory , and if there is a miss we still write back to memory but choose to bring it in cache based on allocate policy and there are 64 write instructions .

dword (16,16):

Wrt HIT Wrt MISS Hit rate Number of hits NUmber of misses Write Backs Total Accesses 1 1 0.7481 193 65 33 258 1 0 0.01163 3 255 254 258 0 1 0.7481 193 65 256 258 0 0 0.01163 3 255 256 258

Observation -- The reasoning for write back and write allocate is similar as stated above . However the number of write backs are not 0 because of larger iterations . The address in later iterations are stored at index containing the data which needs to be replaced . Thus this need to be updated in memory resulting in write backs .

In the case of write back and write without allocate the number of hits and misses follow the same pattern is observed as mentioned for (8,8) case . Every missed write instructions results in a write back which are 254 since two writes were a hit .

For write through every write instruction results in a write back , since in case of a miss we fetch the block in cache , it results in subsequent hits due to use of "spatial loaclity" in the code implementation . As stated above , even though the hit rate is same as write back and write allocate , the number of cycles required are more due to the large number of write backs .

For write through and write without allocate , the number of hits are very low and number of write backs are large . The reasoning is similar as to program -1 . Due to write through policy every write back results in a write back . The misses are observed more due to no fetching og block upon on a cache miss . Since the code fetches next double word iteratively , this policy results in large misses .

PROGRAM -2

dword (8,8):

Wrt HIT Wrt MISS Hit rate Number of hits NUmber of misses Write Backs Total Accesses 1 1 0.7424 49 17 0 66 1 0 0.04545 3 63 62 66 0 1 0.7404 49 17 64 66 0 0 0.04545 3 63 64 66

Observation -- The program -2 jumps 64 bytes i.e 8 long words in the inner loop and moves one double word ahead in the outer loop . Thus both temporal locality and spatial locality can be utilised as stated previously .

Upon choosing write back with respect to hit and write alloacte with respect to a miss , number of hits are very high . Due to the inital misses the entire block of 4 double words is fetched in the cache , each store operation skips an index in the cache due to jumping of 64 bytes . After an iteration of inner loop , the address gets incremented by a double word and the second iteration of inner loop begins , however now the cache already contains the requested address resulting in hits . The pattern is followed for both even

and odd index positions . Since the no data has the same index but different tag in the entire code , there are no write backs to memory .

Choosing without allocate upon miss causes a large number of misses due to update only happening in memory directly rather than fetching it back to cache . The few hits are due to the initial miss of load instructions which are brought into cache . The number of large backs is equal to the number of store instructions -1 as every store instructions results in a cache miss (except the one present in cache due to the first load) causing it to be updated in memory wihtout fetching it .

Since the number of store instructions are large having a write allocate is helpful as subseqent store instruction might require address already present in the cache . This is exactly what is observed in the case of program -2 . Due to write through policy every write instructions is written back to memory resulting in the number of write back to be 64 . Even though the hit rate is same write back and write through with write allocate , write through takes more time as it needs more cycles to update in memory everytime .

The number of hits are low and number of writebacks are large for the last case having writr through and write without allocate policy incase of miss . The reasoning is similar as to program -1 . Since the code makes 8 long words jump in the inner loop , evrry store results in a cache miss causing a write back to the memory , since we do not bring the data in the memory it results in misses for future write instrutions too .

dword (16,16):

Wrt HIT Wrt MISS Hit rate Number of hits NUmber of misses Write Backs Total Accesses 1 1 0.01163 3 255 223 258 1 0 0.01163 3 255 254 258 0 1 0.01163 3 255 256 258 0 0 0.01163 3 255 256 258

Observation --

The observations are very different here . This is due to the longer iterations in the inner loop of the code . The code jumps 16 long words moving to the 4th index . Since this process is repeated 16 times in the inner loop and cache line size is 8 , the data written in block gets replaced in a single iteration of outer loop . The last tag bit toggles between 0 and 1 with a duration of half size of inner loop . As a result when trying to fetch the next double word in 2nd iteration of outerloop , the tags bits do not match resulting in a miss . This is repeated and hits are only seen initially due to the fetch of first load instruction . The number of write backs is also high because every time it a tag mismatch is found at an index , it is written back to the memory .

In the case of write back , without allocate The hit rate is same as in case of program -1 (16,16) . This can be possible because even though the way program 2 fetches the data is very different from program 1 , due to no allocate in case of miss . The future store result in a miss . All locations which have a cache miss write directly to memory resulting in high write backs .

In the case of write through with allocate the hit rate is very low as observed in write back . The reason is also same due to the small number of lines in cache . The data that is fetched in the cache gets replaced in a single iteration of outer loop as mention above . When the next double word is fetched the block has been replaced resulting in a miss . The number of write backs are equal to the number of write instructions because everytime is writes the memory is also updated .

For write through , without allocate the results are same as with allocate because the upon a miss the data is written to memory directly . Thus the subsequent store instructions also result in miss because no block

is fetched in the cache . The inital hits are observed due to the load instruction . The number of write backs are equal to the number of write instructions because everytime is writes the memory is also updated .

Question - 3

Program -3 is similar to program -1 but it also loads a data from 1024 offset of address stored in x12which causes change in the tag bit since 10th bit is part of tag . The total accesses also increase in the case due to the additional load added in the inner loop making total access to be 64*2+2 = 130 in case of dword (8,8) or 16*16*2+2 = 514 in case of dword (16,16) .

dword (8,8):

Cache	Hit rate	Number of hits	NUmber of misses	Total Accesses
Direct Mappped Cache	0.01538	2	128	130
Set Associative Cache	0.7385	96	34	130
Fully Associative Cache	0.7385	96	34	130

Observation --

Direct Mapped Cache ->

The last bit of tag toggles between 1 and 0 due to the load 1024 offset as a result the block gets replaced as soon as it was written in the cache resulting in a large number of misses . Since the code accesses memory in iterative manner it attempts to read the address incremented by 8 again but it has been replaced with the new address having tag bit 1 . As a result the number of misses are very high . The number of hits are 2 because of the first load instruction causes a block to come in cache causing hit for the next 2 instructions before tag toggle begins .

Set Associative Cache ->

Due to 2 way set associative cache the 1024 offset load at the same index doesnt cause an issue . The data is stored at the same index with different tag bit due to associativity . As a result the number of misses are very low . In every iteration the load causes a miss for 1024 offset and stores it in the same index but at a different position due to number of ways being 1 , the later 3 loads are a hit due to the entire block being fetched in the stack and this pattern is repeated for the next iteration . That is why number of hits are apporximately 3 times the number of misses .

Fully Associative Cache ->

The hit rate and number of hits of fully associative cache is similar to set associative cache because an address can be stored at any index , so the 1024 offset having different tag bit which causes an issue in direct mapped cache can be stored in a different index in fully associative cache . Since the amount of loads are almost same as the total number of double words stored in cache , the amount of replacement is very less and for every one miss we again have 3 hits in the same block . However comparing with the entire cache parallely requires more hardware adding overhead to the design as compared to set associative cache .

dword : (16,16)

Cache	Hit rate	Number of hits	NUmber of misses	Total Accesses
Direct Mappped Cache	0.06809	35	479	514
Set Associative Cache	0.7471	384	130	514
Fully Associative Cache	0.7471	384	130	514

Direct Mapped Cache ->

The hit rate although low is better than wht was obtained for (8,8) . This is observed bacause of the longer iterations , the 1024 offset address stored in the block results in a hit after few iterations of the outer loop . Every block gets immediately replaced with address having last tag bit to be 1 due to the 1024 offset load , However after 8 iterations of the outer loop the address is added by a factor of 16*8*8 since the outer loop is executed 8 times and in each iteration of outer loop 8 is incremented to the address 16 times . As a result the address equals the address stored earlier (16*8*8 = 1024) , causing a hit . For the remaining iterations every first double word accessed causes a hit followed by replacement of the entire address in block due to 1024 offset as menttioned earlier . This is why number of hits are more than (8,8) case resulting in a slightly better hit rate . Thus usage of spatial locality depends on the program written and the way it is executed .

Set Associative Cache ->

As expected the hit rate is much higher as compared to direct mapped cache . Since the code loads to different address having the same index but different tag both addresses are stored in the set of the cache . Since we iterate over the address by incrementing 8 i.e fetch the nextd the double word it is already present in cache resulting in a hit . This pattern is obsered for all the iterations . Both loads present in ineer loop result in hit due to the block being present in cache . Every one miss results in 3 hits in the set . As a result the number of hits are approximately 3 times the number of misses and the number of misses are very low .

Fully Associative Cache ->

The performance of fully associative cache are again similar to the set associative cache . The reason being the same as mentioned in the case of (8,8) . The address can be stored at any index and thus the 1024 offset address can be stored at a different index . The access time will be large due to the parallel comparison of the entire cache but only considering the hit rate , it is similar to set associative cache .