Computer Architecture - CS2323. Autumn 2024

Homework-4 (Cache experiments)

Name : Ahmik Virani

Roll Number : ES22BTECH11001

L1: (8, 8)

Q1)

Program 1

Varying lines keeping blocks = 2, ways = 0

Lines value	Hit Rate	Hits	Misses	Total Accesses
1	0.7424	49	17	66
2	0.7424	49	17	66
3	0.7424	49	17	66

Varying blocks keeping Lines = 3, ways = 0

Blocks value	Hit Rate	Hits	Misses	Total Accesses
3	0.8636	57	9	66
4	0.9242	61	5	66
5	0.9545	63	3	66

Varying ways keeping Blocks = 2, Lines = 3

Ways value	Hit Rate	Hits	Misses	Total Accesses
0	0.7424	49	17	66
1	0.7424	49	17	66

Program 2

Varying lines keeping blocks = 2, ways = 0

Lines value	Hit Rate	Hits	Misses	Total Accesses
1	0.0303	2	64	66
2	0.0303	2	64	66
3	0.04545	3	63	66

Varying blocks keeping Lines = 3, ways = 0

Blocks value	Hit Rate	Hits	Misses	Total Accesses
3	0.8636	57	9	66
4	0.9242	61	5	66
5	0.9545	63	3	66

Varying ways keeping Blocks = 2, Lines = 3

Ways value	Hit Rate	Hits	Misses	Total Accesses

0	0.04545	3	63	66
1	0.7424	49	17	66

First lets understand what each program does

Program 1 : This is analogous to accessing an array in sequential manner.
Program 2 : This is analogous to accessing the array in a periodic but not sequential manner. An example is given in the answer to the questions

Q1)
Program 1
As we increase lines, but still the hit rate is not going up. The reason for this is because we are always using new memory location, therefore the only thing that determines the hit rate for program 1 is the blocks (increasing blocks increases hit rate), because this means that we are increasing the number of array values as mentioned in the analogy that are being stored. Basically through spatial locality. Also, increasing the associativity does not matter, that is because we are not really accessing the same tag. We are doing sequential access, which means that we are always accessing a new index, so keeping more tags of the same index does not really matter, here only spatial locality matters.

Program 2
Here there is no spatial locality, nor is there temporal locality, we are always accessing a memory location that is some distance away. Also there is no temporal locality as we are always accessing a new value. And this is also the reason why we have lower hit rate for program 2 than for program 1 in most of the cases.
For example when y = 8:

1. Iteration 1 : index = 0, 8, 16, 24, ….
2. Iteration 2 : index = 1, 9, 17, 25, ….
3. So on

Now, if we increase the number of indexes, i.e. the lines, it does not really affect. However, when we increase the block size and can accommodate, more values. Which means that once we do iteration 1, we can accommodate 0->7 in one row, then 8-> 15 in one row and so on. Thus in iteration 1 we will have misses, however the subsequent iterations will be hits. Also note that this requires us to have at least lines value as 2^3 as the number of iterations are 8. If we run this in lines = 2^2 instead of 2^3, then again we will get more miss. Now here associativity is increasing hit rate because we can store value for 2 iterations. Which means that iteration 0 will be all misses, however because set associativity is 2, we do not need to replace any thing old that we may use again, instead we can just store it in a different row as set associativity = 2.

Q2)
Program 1

Write-Policies	Hit Rate	Hits	Misses	Total Accesses
Write Back Write Allocate	0.7424	49	17	66
Write Back No Write Allocate	0.04545	3	63	66
Write Through Write Allocate	0.7424	49	17	66
Write Through No Write Allocate	0.04545	3	63	66

Program 2

Write-Policies	Hit Rate	Hits	Misses	Total Accesses
Write Back Write Allocate	0.7424	49	17	66

Write Back No Write Allocate	0.04545	3	63	66
Write Through Write Allocate	0.7424	49	17	66
Write Through No Write Allocate	0.04545	3	63	66

Here we can see that write allocate has a higher hit rate than no write allocate, this is because we are accessing the bit that we are writing to. Basically write allocate governs the hit rate. When write allocate, we are writing to the cache when there is a write miss, and since we are using the value which is being accessed close by i.e. through the spatial locality since the no. of words stored in one index is high, we are just storing everything and then using them later.

Q3)

Program 3

Associativity	Hit Rate	Hits	Misses	Total Accesses
32-entry 4-word direct mapped	0.01538	2	128	130
32-entry 4-word fully associative	0.7385	96	34	130
32-entry 4-word 2-way set associative	0.7385	96	34	130

Program 3 : This is analogous to accessing an array in alternating sequential manner. Eg first 1, 128, 2, 129, …

Again here, there is no spatial and temporal locality, however this time we are accessing memory locations which are very far apart, unlike program 2.
In the first iteration we access indexes 0, 1, 2, 3, 4, 5, 6, 7 and 128, 129, 130, 131, 132, 133, 134, 135. Thus when we do 2 way set associativity, we can store 1->7 and 128->135 together, whereas when we store them in direct mapped, they will overwrite each other at the same index and repeatedly just change the entry which we will use in the near future. 2-way set associative is enough. Fully associative is basically random, which means we can just store it if we find space, thus since we have space to store it, it is as good as 2-way set associative.

<u>L1: (16, 16)</u>

Q1)

Program 1

Varying lines keeping blocks = 2, ways = 0

Lines value	Hit Rate	Hits	Misses	Total Accesses
1	0.7481	193	65	258
2	0.7481	193	65	258
3	0.7481	193	65	258

Varying blocks keeping Lines = 3, ways = 0

Blocks value	Hit Rate	Hits	Misses	Total Accesses
3	0.8721	225	33	258
4	0.9341	241	17	258
5	0.9651	249	9	258

Varying ways keeping Blocks = 2, Lines = 3

Ways value	Hit Rate	Hits	Misses	Total Accesses
0	0.7481	193	65	258
1	0.7481	193	65	258

Program 2

Varying lines keeping blocks = 2, ways = 0

Lines value	Hit Rate	Hits	Misses	Total Accesses
1	0.007752	2	256	258
2	0.007752	2	256	258
3	0.007752	2	256	258

Varying blocks keeping Lines = 3, ways = 0

Blocks value	Hit Rate	Hits	Misses	Total Accesses
3	0.007752	2	256	258
4	0.01163	3	255	258
5	0.9574	247	11	258

Varying ways keeping Blocks = 2, Lines = 3

Ways value	Hit Rate	Hits	Misses	Total Accesses
0	0.007752	2	256	258
1	0.007752	2	256	258

Program 1

Here now the hit rate is again the same on increasing lines, and the reason too remains same, i.e. because it is sequential access and everytime we are accessing a new index. Only spatial locality is there but no temporal locality. Which means that only the blocks define the hit rate of this program. And as seen before, increasing the number of blocks increases the hit rate. Also we can see that associativity does not really affect and again the same argument holds, that you are just increasing the number of entries for the same index, but since you are accessing sequentially it does not really matter, you can just overwrite without causing any problem or future miss of the same data as this data being overwritten is not going to be used again.

Program 2

Hit rate remains constant with lines with the same reason as explained for the L: {8, 8} case. You are always accessing new entries and those entries are not having spatial locality for low block value(2^2).

Again the block case remains the same,

1. Iteration 1 : index = 0, 16, 32, 48, ….
2. Iteration 2 : index = 1, 17, 33, 49, ….
3. So on

However here we can see that the hit rate only is high when blocks = 2 ^ 5. This is because for small block values, when we are running the iterations, the values will be overwritten (this happens because example when blocks = 2^4, first store from 0->15, then 16->31 and so on. However this loop runs 16 times so we need 16 slots to store, but we only have space for 8 lines, thus we need double the space for this which means we need 2^5), and we cannot take advantage of spatial locality. However when each index can store 32 values, we will store index 0 and get hit in 16, then store 32 and get hit in 48. Also this high size in iteration 2, we continue getting hits for index 1, 17, and so on. Thus here, we need 8 lines

because each line we are storing for 2 sets i.e. for set 0->15 and 16-> 31 in the same index, thus gives 2 hits, so in 16 iterations we need only 8 lines to satisfy.

Q2)

Program 1

Write-Policies	Hit Rate	Hits	Misses	Total Accesses
Write Back Write Allocate	0.7481	193	65	258
Write Back No Write Allocate	0.01163	3	255	258
Write Through Write Allocate	0.7481	193	65	258
Write Through No Write Allocate	0.01163	3	255	258

Program 2

Write-Policies	Hit Rate	Hits	Misses	Total Accesses
Write Back Write Allocate	0.01163	3	255	258
Write Back No Write Allocate	0.01163	3	255	258
Write Through Write Allocate	0.01163	3	255	258
Write Through No Write Allocate	0.01163	3	255	258

Program 1

Here again same explanation as before, when we allocate space, we are basically allowing that it can be stored in the cache and reused, but without allocating, we have to access from memory everytime, thus hit rate of write allocate is better.

Program 2

Now here there is no change with write policy. The reason is here there are 32 entries and 4 blocks. Which means that since here unlike program 1 we are not taking advantage of spatial locality, we need to replace in middle of every loop, and in turn again replace in start of next loop. So allocating is pretty much same as not allocating because we are not taking advantage of locality, but instead just replacing what will be used next. Thus if we just increase the lines, then we can see the difference. (When I increased the lines to 2^6, with allocate gave a hit rate of 0.7481 whereas no write allocate gave same 0.01163)

Q3)

Program 3

Associativity	Hit Rate	Hits	Misses	Total Accesses
32-entry 4-word direct mapped	0.06809	35	479	514
32-entry 4-word fully associative	0.7471	384	130	514
32-entry 4-word 2-way set associative	0.7471	384	130	514

Just like the previous explanation, again we just need 2 way set associative as we just need to avoid that extra replacement that is being caused due to lack of space. As shown above it can be done by directly increasing lines, or also by increasing set associativity. (The explanation is same as previously explained). Effect of just increasing the lines->Increasing lines to 2^6 is giving performance 0.8113 hit rate.