COMPENG 4DM4: Computer Communication Networks

Lab 3: Caches

Shanzeb Immad - 400382928 - immads

Josh Umansky - 400234265 - umanskyj

Part 4:

4.1:

Within this part, the paper and pencil analysis was done on the provided code snippet. The first part requested to identify the bit length of the index, offset and tag. To get the offset bit, the log base 2 of the cache line size was calculated, $offset\ bit\ =\ log_2(64)\ =\ 6\ bit$. Next the number of cache lines was calculated by taking the cache size 8 * 1024 and dividing it by the line size. This gives a result of 128, and if the base 2 log of that is done, the index bit is revealed to be 7. The remainder of bits left after 6 and 7 is subtracted by the total 32 equals 19, which is the size of the tag bits.

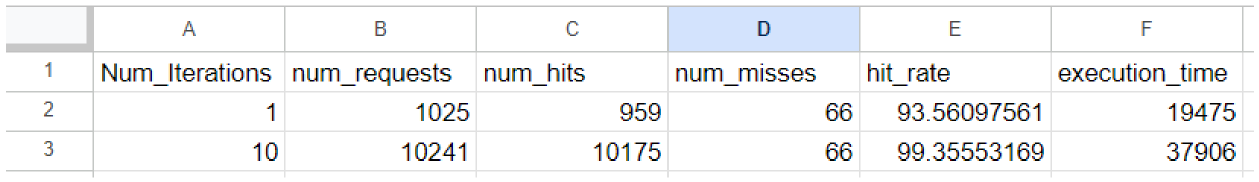
For part b, calculations were done to obtain the hit rate of the code with iterations being 1 and 10. For iteration one, the following was done, # of cache lines = 4096/64 = 64. Since each cache line is 64B and each integer is 4B long, each cache line can hold 16 integers. This goes on for mempool[0] being a miss but the next 15 addresses will hit, then the cycle will repeat at mempool[16] being a miss. This means that throughout the entire 1024 memory accesses, there will be 64 misses and the hit rate can be calculated for a value of 0.9375.

When the iteration becomes 10, the hit rate increases. This is due to the fact that after the first iteration, the cache lines are all populated so no more misses would occur besides the original 64. The total memory access becomes 10 * 1024 = 10240, and the total hit number is 10240 - 64. With this the hit rate is observed to increase to the value of 0.9938.

For these memory pool runs, the locality is of type spatial. This follows the definition given in lecture 6 stating that the accessing data locations are near each other within the memory.

4.2:

Within this part, the simulation is done to observe the impact of the iteration count on the caches. Figure 1.a below shows the simulation result and all values almost perfectly match everything analyzed within the paper and pen calculations. Both the results proved the fact that once the first iteration occurs, no more misses will happen because the cache line is filled. The trend that the hit rate will increase also deems to be true from both the analysis and the simulation.



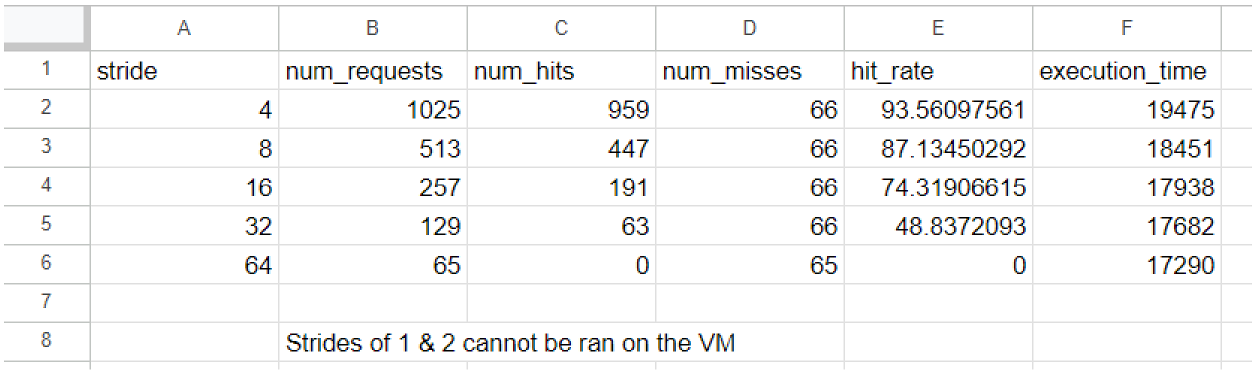
	A	B	C	D	E	F
1	Num_Iterations	num_requests	num_hits	num_misses	hit_rate	execution_time
2	1	1025	959	66	93.56097561	19475
3	10	10241	10175	66	99.35553169	37906

Figure 1.a: Simulation result for Part 4

Part 5:

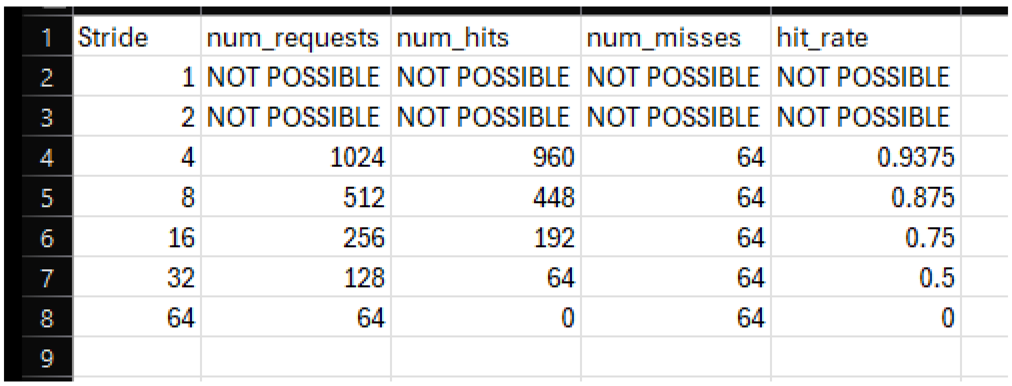
For the analysis, the only thing being changed for each run is the value for stride, this means that the number of memories accessed will either increase or decrease but the value for the cache line size stays the same. This leads to the number of integers per line staying the same of 16 but only certain integers would be looked at. The mentioned changes lead to the number of lines changing as well as when stride equals 4, the same results from part 4, iteration 1 is collected. With the increase of the stride, the number of cache lines stays the same, which results in the same number of misses for all strides, but the request number decreases exponentially which also exponentially decreases the hit rate. The results for stride equal to 1 or 2 is not possible as that would give an incrementation of a fraction and a memory access at a fractioned number is not possible.

The results from the simulation exactly match the paper and pencil analysis showing the exponential decrease of the hit rate and how the number of misses stay constant throughout the changes in stride.



	A	B	C	D	E	F
1	stride	num_requests	num_hits	num_misses	hit_rate	execution_time
2	4	1025	959	66	93.56097561	19475
3	8	513	447	66	87.13450292	18451
4	16	257	191	66	74.31906615	17938
5	32	129	63	66	48.8372093	17682
6	64	65	0	65	0	17290
7						
8		Strides of 1 & 2 cannot be ran on the VM				

Figure 2a: Experimental results for spatial locality



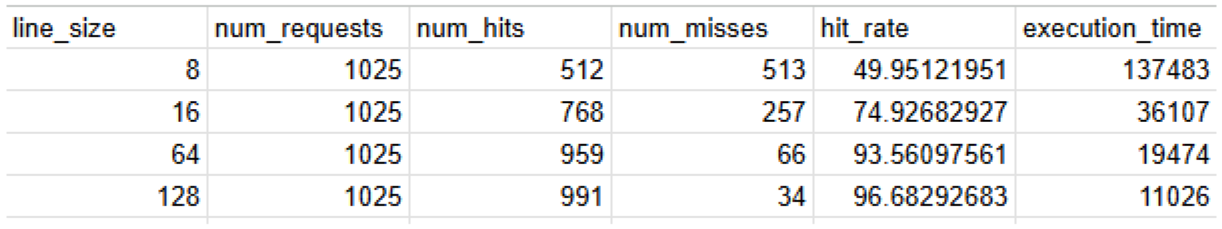
1	Stride	num_requests	num_hits	num_misses	hit_rate
2	1	NOT POSSIBLE	NOT POSSIBLE	NOT POSSIBLE	NOT POSSIBLE
3	2	NOT POSSIBLE	NOT POSSIBLE	NOT POSSIBLE	NOT POSSIBLE
4	4	1024	960	64	0.9375
5	8	512	448	64	0.875
6	16	256	192	64	0.75
7	32	128	64	64	0.5
8	64	64	0	64	0
9					

Figure 2b: Analytical result for spatial locality

Part 6

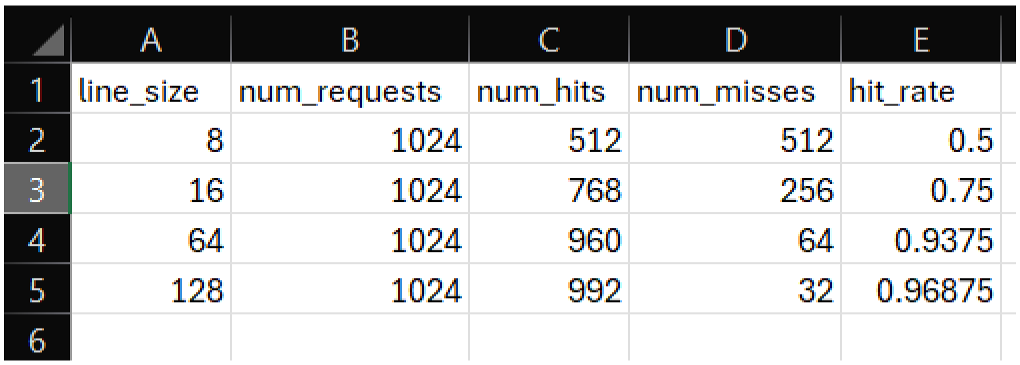
Similar to the last test, the same simulation is run but now the effect of the cache line size is observed to view what affects it has. The cache line size is swept with the values of, 8, 16, 64 and 128. The theorized trend is that with the increase of the line size, the number of hits will also increase. The total number of memory requests stays constant throughout the runs but with a bigger line size, more integers can be fit, which allows more hits in a line and less misses, since the first bit of the cache line is the only one to miss. Both the simulation and paper analysis proved this theory but the values seem to be a bit different.

The experimental and the analytical results are almost identical, detailed in Figures 3a and 3b respectively show the results for the experiment and analysis.



line_size	num_requests	num_hits	num_misses	hit_rate	execution_time
8	1025	512	513	49.95121951	137483
16	1025	768	257	74.92682927	36107
64	1025	959	66	93.56097561	19474
128	1025	991	34	96.68292683	11026

Figure 3a: Experimental results for cache line size



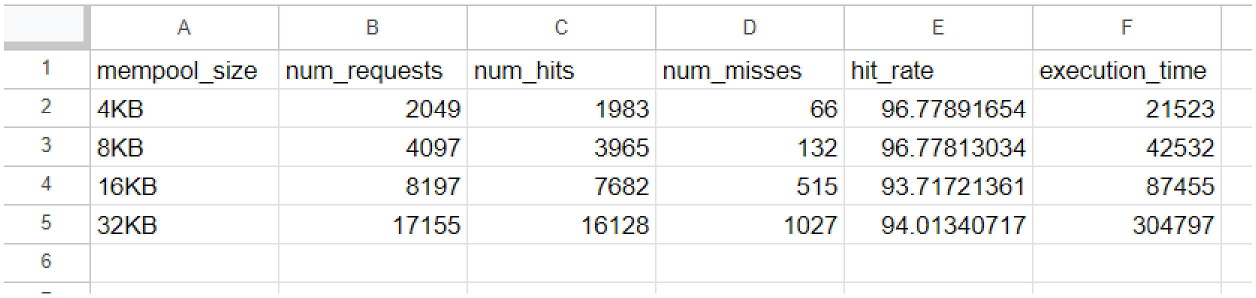
	A	B	C	D	E
1	line_size	num_requests	num_hits	num_misses	hit_rate
2	8	1024	512	512	0.5
3	16	1024	768	256	0.75
4	64	1024	960	64	0.9375
5	128	1024	992	32	0.96875
6					

Figure 3b: Analytical result for cache line size

Part 7:

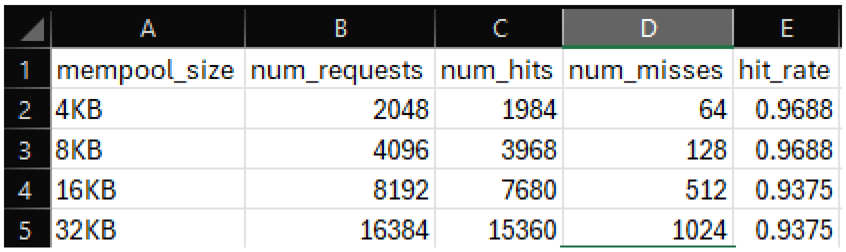
For this part, the noticeable effect of the memory pool is examined by sweeping its value with, 4KB, 8KB, 16KB and 32KB. The size of the memory pool at 4 KB and 8KB fit within the cache memory. The results of these were obtained similar to the process from part 4 but taking the second iteration into consideration. For the memory pool size at 16 KB and 32 KB the calculated number of misses would be found the same but way but multiplied by 2 since the pool size is bigger than the cache so misses calculated rewrite themselves on top of the previous misses.

Both the simulation and analytical results prove the observations with 4KB and 8KB having a hit rate of 96% and decreasing to ~93% for the memories larger than the cache size.



mempool_size	num_requests	num_hits	num_misses	hit_rate	execution_time
4KB	2049	1983	66	96.77891654	21523
8KB	4097	3965	132	96.77813034	42532
16KB	8197	7682	515	93.71721361	87455
32KB	17155	16128	1027	94.01340717	304797

Figure 4a: Experimental results for cache capacity

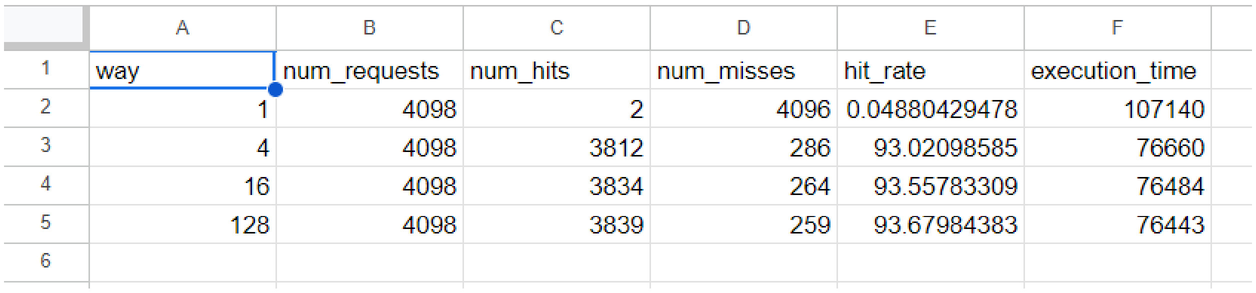


mempool_size	num_requests	num_hits	num_misses	hit_rate
4KB	2048	1984	64	0.9688
8KB	4096	3968	128	0.9688
16KB	8192	7680	512	0.9375
32KB	16384	15360	1024	0.9375

Figure 4b: Analytical result for cache capacity

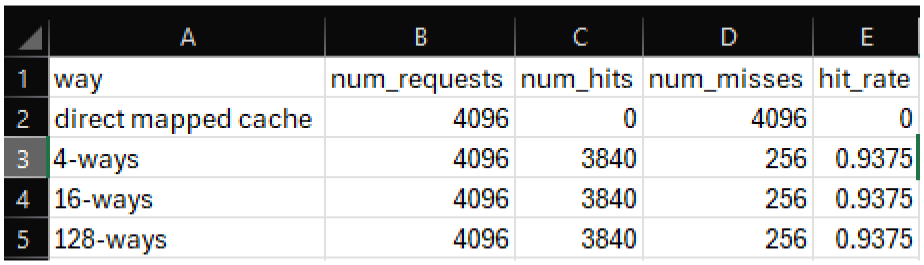
Part 8:

As the ways increase, theoretically the number of misses and requests do not change. For any associativity more than 2, allows for separate addresses to be used for mempool[i] and mempool[i + stride/4]. This allows for the first bit of every line to miss, then allowing the stride/4 address to also miss. There would be 2 misses at every line with 128 cache lines available. If direct mapping/one way associativity were to occur, the mempool[i + stride/4] call would always cause a conflict at every value of i. Both the experiment and analysis agree with these assumptions and direct mapping gives a hit rate of 0, and the rest are the same at 93.75%, shown in Figure 5a and 5b respectively.



way	num_requests	num_hits	num_misses	hit_rate	execution_time
1	4098	2	4096	0.04880429478	107140
4	4098	3812	286	93.02098585	76660
16	4098	3834	264	93.55783309	76484
128	4098	3839	259	93.67984383	76443

Figure 5a: Experimental result for associativity



way	num_requests	num_hits	num_misses	hit_rate
direct mapped cache	4096	0	4096	0
4-ways	4096	3840	256	0.9375
16-ways	4096	3840	256	0.9375
128-ways	4096	3840	256	0.9375

Figure 5b: Analytical result for associativity