CS3021/3421 Tutorial 6

Q1. Compute the number of hits and misses if the following list of hexadecimal addresses is applied to caches with the following organisations.

(i) 128 byte 1-way cache with 16 bytes per line (direct mapped)
(ii) 128 byte 2-way set associative cache with 16 bytes per line
(iii) 128 byte 4-way set associative cache with 16 bytes per line
(iv) 128 byte 8-way associative cache with 16 bytes per line (fully associative)

0000 ⇒ 0004 ⇒ 000c ⇒ 2200 ⇒ 00d0 ⇒ 00e0 ⇒ 1130 ⇒ 0028 ⇒
113c ⇒ 2204 ⇒ 0010 ⇒ 0020 ⇒ 0004 ⇒ 0040 ⇒ 2208 ⇒ 0008 ⇒
00a0 ⇒ 0004 ⇒ 1104 ⇒ 0028 ⇒ 000c ⇒ 0084 ⇒ 000c ⇒ 3390 ⇒
00b0 ⇒ 1100 ⇒ 0028 ⇒ 0064 ⇒ 0070 ⇒ 00d0 ⇒ 0008 ⇒ 3394 ⇒

Assume that the first 4 bits of the address is used as the offset within the cache line, the next $\log_2(N)$ bits select the set and the remaining bits form the tag. Furthermore, assume that the all cache lines are initially invalid and that a LRU replacement policy is used.

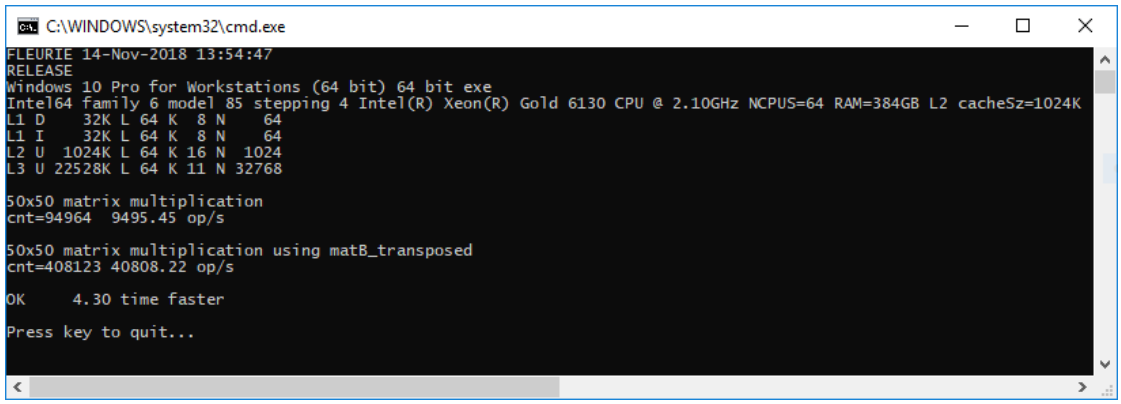
Q2. Write a program (in C, C++, Java, …) to solve Q1 (should be less than 200 lines of code). Make sure you can create a generalised cache object with parameters L, K and N.

Q3. It is possible to write programs to be cache friendly (i.e. have a high cache hit rate). One example is the multiplication of matrices. The following code is a typical loop for computing matC = matA*matB.

```
for (UINT row = 0; row < N; row++) {
    for (UINT col = 0; col < N; col++) {
        UINT sum = 0;
        for (UINT k = 0; k < N; k++)
            sum += matA[row][k]*matB[k][col];
        matC[row][col] = sum;
    }
}
```

The central computation of sum is cache friendly with regard to the accesses made to matA and matC, but unfriendly with regard to the accesses made to matB. This is because elements in a row are stored in consecutive memory locations so going from one column to the next is likely to access data in a previously read cache line whereas going from one row to the next is not. One simple way to make the accesses to matB more cache friendly is to transpose matB and modify the multiply code to take this into account. The CS3021/3421 website has links to the code for a test program which will continuously multiply two random NxN matrices for NSECONDS using method 1 and then method 2. It reports the number of multiplications performed per second for

each method. There are 4 files matrixMultiply.cpp, helper.h, helper.cpp and a makefile. The helper files provide helper functions and supports so that the program can be compiled and run on Windows and Linux. Add code to transpose matB and to multiply matA by matB_transposed (look for *ADD YOUR CODE HERE*), build and run (see the makefile) to determine the speed up, if any.



This Tutorial is worth double marks (marked out of 20). Submit the source code for Q2 and Q3 and one .pdf document containing (i) evidence that your program generates the correct results for Q2 and (ii) a screen shot of the output of Q3. Please submit your answer via Blackboard by 9am Fri 30-Nov-18.

128 byte 1-way cache with 16 bytes per line (L = 16, N = 8, K=1)

tag	4 word (16bytes)			

Address Format

15															0

address	set	hit/miss
0000		
0004		
000c		
2200		
00d0		
00e0		
1130		
0028		
113c		
2204		
0010		
0020		
0004		
0040		
2208		
0008		
00a0		
0004		
1104		
0028		
000c		
0084		
000c		
3390		
00b0		
1100		
0028		
0064		
0070		
00d0		
0008		
3394		

128 byte 2-way cache with 16 bytes per line (L = 16, N = 4, K=2)

tag (K=0)	tag(K=1)	4 word (16bytes)				4 word (16bytes)			

Address Format

15															0

address	set	hit/miss
0000		
0004		
000c		
2200		
00d0		
00e0		
1130		
0028		
113c		
2204		
0010		
0020		
0004		
0040		
2208		
0008		
00a0		
0004		
1104		
0028		
000c		
0084		
000c		
3390		
00b0		
1100		
0028		
0064		
0070		
00d0		
0008		
3394		