Assignment 5 Report

KR Hariharan
CS19B079

March 18, 2021

System Configuration	• AMD Ryzen 7 4800h • Clock Frequency : 2.9GHz • Cores : 8 • Threads : 16 • L1 cache: 256KB (L1I) + 256KB (L1D) - 8 way set associative • L2 cache: 4MB - 8 way set associative • L3 cache: 2x4MB • Main Memory: 2x8GB
Scenario 1	**Row major - row major multiplication**
Perf tool	
duration_time (ms)	3,016,974.853 (+- 2.43%)
cycles	12,86,96,96,652 (+- 2.37%)
instructions	9,53,66,92,338 (+- 0.11%) [0.74 insn per cycle]
branch-instructions	31,16,68,691 (+- 1.42%)
cache-references	37,55,05,247 (+- 2.05%)
cache-misses	26,88,54,345 (+- 0.37%) [71.598 % of all cache refs]
L1-dcache-loads	64,40,28,793 (+- 0.51%)
L1-dcache-load-misses	27,14,68,752 (+- 0.03%) [42.15% of all L1-dcache hits]
dTLB-loads	27,11,56,332 (+- 0.05%)
dTLB-load-misses	15,25,062 (+- 10.91%) [0.56% of all dTLB cache hits]
LLC-loads	<not supported>
LLC-load-misses	<not supported>
Valgrind	
LL references	269,095,247 (268,830,367 rd + 264,880 wr)
LL misses	399,094 (135,307 rd + 263,787 wr)
Scenario 2	**Row major - Column major multiplication**
Perf tool	
duration_time (ms)	565,977.319 (+- 0.28%)

cycles	2,41,42,79,152 (+- 0.28%)
instructions	9,05,20,57,137 (+- 0.05%) [3.75 insn per cycle]
branch-instructions	31,23,79,939 (+- 1.24%)
cache-references	13,65,33,346 (+- 0.07%)
cache-misses	23,23,578 (+- 0.46%) [1.702 % of all cache refs]
L1-dcache-loads	62,96,84,120 (+- 0.39%)
L1-dcache-load-misses	6,93,86,531 (+- 0.14%) [11.02% of all L1-dcache hits]
dTLB-loads	11,84,813 (+- 0.09%)
dTLB-load-misses	28,930 (+- 4.33%) [2.44% of all dTLB cache hits]
LLC-loads	<not supported>
LLC-load-misses	<not supported>
Valgrind	
LL references	67,440,975 (67,178,142 rd + 262,833 wr)
LL misses	397,044

Observation
We can see that matrix multiplication is **5.33 times faster, with 5.06 more instructions per cycle when B is stored in column-major format**. This can be directly correlated to the much lower percentage of cache misses, L1-dcache-load-misses, and LL references.

The sharp difference in cache misses can be explained by **spatial locality.**

When a matrix is stored in row-major format, elements of the same row are stored in nearby addresses, whereas when a matrix is stored in column major format, elements of the same column are stored in nearby addresses.

In matrix multiplication, we access the first matrix (A) row-wise and second matrix (B) column-wise.

Thus, calculating the value of element C[i][j] when:

1. **B is stored in row-major format**: for every k in [1,1024], each element A[i][k] is stored in adjacent addresses, but each B[k][j] is stored 1024 words away from each other. Thus, while only one block of data is accessed for every 16 values of A[i][k] (block size = 64B = 16*(sizeof)int), a different block needs to be accessed for each B[k][j], leading to a large number of capacity and conflict misses.

2. **B is stored in column-major format**: for every k in [1,1024], each element A[i][k], as well as each element of B[j][k] are stored in adjacent addresses. Thus only one block of data is accessed for every 16 values of A, and one block for every 16 values of B (instead of 16 blocks when stored in row-major order). As a much lower number of blocks are accessed, due to the exploitation of spatial locality in matrix B, the probability of capacity and conflict misses is greatly reduced.