CS3210 Lab 2 Report

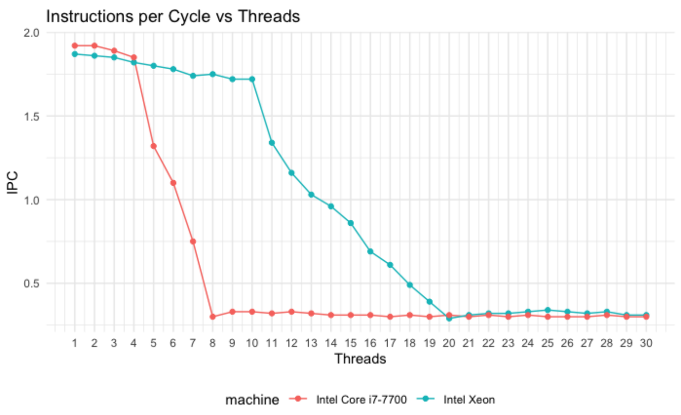
Wang Xiyu (A0282500R)

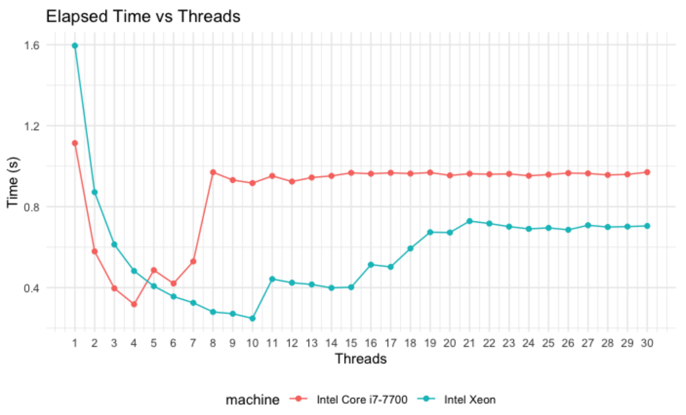
September 16, 2025

Ex11 Performance Metrics

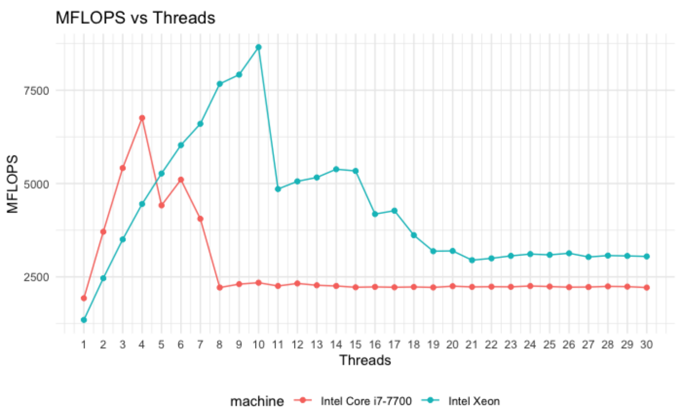
This sections shows the comparision of i7-7700(4 cores 8 threads) and Xeon Silver 4114 (10 cores 20 threads) in performance measured by instruction per cycle(IPC), wall clock time(elapsed time) and MFLOPS(Million floating point operations). Both processors have the same core architecture.

Instruction per Cycle & Elapsed time & MFLOPS





We can observe that for i7-7700 IPC moderately decreases from 1 to 4 threads, and IPC on Xeon 4114 also moderately decreases at similar rate from 1 to 10 threads, indicating that there exist overhead of thread scheduling, therefore the number of instructions for the actual task counted in floating point operations decreases. But the overall performance measured in wall clock time improves rapidly as there are more processing units being utilized. When the number of threads = the number of physical cores the wall clock time reaches minimum, as all cores are utilized and the overhead due to thread scheduling and memory contention is still low.



When the number of cores < number of threads < number of processing units(PUs), instruction per cycle for both machines decreases more quickly, as other than the task allocation overhead, there is memory overhead incurred as well, since both PUs in the same physical core share the same L1 and L2 caches, and when both PUs are being utilized, there can be memory contention. For both processors we can observe that MFLOPS peaks when the number of threads = the number of cores, this is intuitive as there are more processing units are being utilized and the number of floating point operation increases. MFLOPS starts to decrease when number of cores $\leq$ the number of threads $\leq$ the number of PUs due to memory contention. When number of PUs < number of threads IPC and MFLOPS stablize at the lowest, as the CPU cannot physically run more threads than the number of PUs in parallel, and all threads are not I/O-bound, more threads over the number of PUs will not be scheduled, thus explains the stablization.

Ex12&13 Optimization & Performance Analysis

As we can observe from the perf statistics, there are only scalar floating point operations being performed, which is a wasted opportunity for SIMD parallelism as we are accessing array elements and we can exploit the spatial locality if we can access contiguous cache in consecutive instructions. As 2D arrays are stored row-wise, wwe can access the first matrix by rows, but the second matrix is accessed by columns first, ie jumping by `B.size()`. If we can access matrix B row-wise we can utilize AVX instructions to perform packed floating point operations.

$$AB[i][j] = \sum_{k=0}^{\texttt{A[0].size()}-1} A[i][k] \times B[k][j]$$

We can observe that the order of the accessing order does not matter, as it does not change the way each cell in the result matrix is updated, therefore algebraically the result is the same; numerically, floating-point non-associativity may cause small rounding differences, but that's a trade-off for better performance.

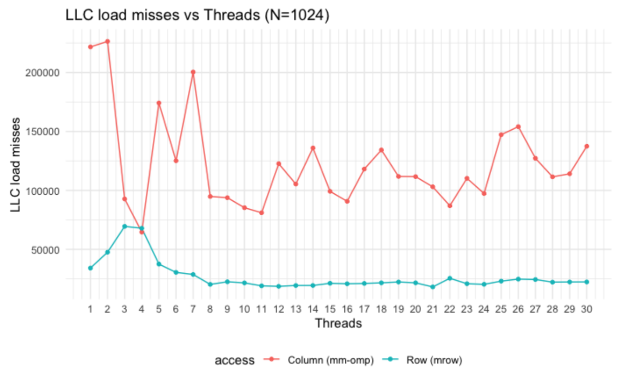
Original

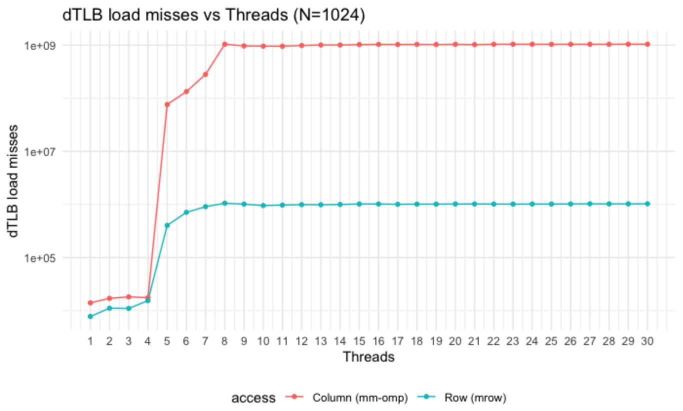
```
for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
        for (k = 0; k < size; k++)
            AB[i][j] += A[i][k] * B[k][j];
```

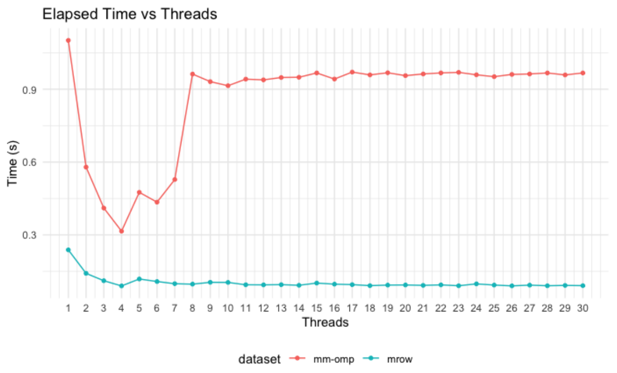
Optimized

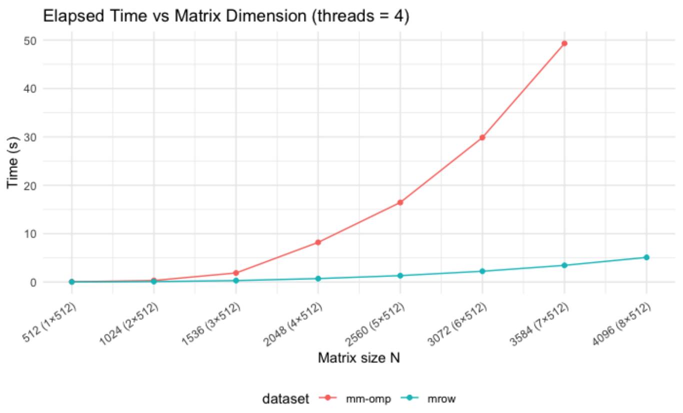
```
for (i = 0; i < size; i++)
    for (k = 0; k < size; k++)
        for (j = 0; j < size; j++)
            AB[i][j] += A[i][k] * B[k][j];
```

As we can see the improvement of cache misses and data TLB misses is massive, At N=1024, row vs column shows $\sim 20\times$ fewer dTLB misses and $\sim 19-25\times$ fewer L1D misses, but only $\sim 5\times$ fewer LLC misses, The cache line size has 64 bytes which can bring 16 single precision floating point numbers a time, row wise access can make full use of the 16 elements brought in while column wise access only 1 among the 16 is used. This is consistent with the observation. L1 data cache miss rate is reduced by a factor of 10, and the value does not change significantly across number of threads, thus the figure is moved to the appendix. This better utilization of cache leads to better performance measured by wall clock time. This improvement amplifies as the size of matrices multiplied increases.









Reproduce result

compile & perf sampling options:

```
g++ mmomp.cpp -o mmomp -fopenmp -O3
srun perf stat -r 3 -e fp_arith_inst_retired.scalar_single, fp_arith_inst_retired.128b_packed_single,
     fp_arith_inst_retired.256b_packed_single, cycles,instructions ./mmomp
g++ rowbase.cpp -o rowbase -fopenmp -O3
srun perf stat -r 3 -e fp_arith_inst_retired.scalar_single, fp_arith_inst_retired.128b_packed_single,
     fp_arith_inst_retired.256b_packed_single, cycles, instructions, ./rowbase
```

Appendix

All resources used in preparing the diagrams, e.g. scripts, modified code can be found here: https://github.com/Wxy2003-xy/CS3210-Lab-References