CS3210 Lab 2

Richard Willie (A0219710L)

September 8, 2022

Exercise 6

As observed from the figures, there is no linear relationship between increasing the number of threads, and other performance metrics such as IPC (Instructions Per Cycle), MFLOPS (Million Floating Point Operations Per Second), and execution time (or time elapsed).

It is also interesting to note that IPC and MFLOPS decreased significantly when the number of threads exceed the number of physical cores. In addition, the execution time decreased (as expected) as the number of threads increased, however, it eventually plateau beyond the number of physical cores on both i7-7700 and xs-4114. This is probably an indication that the number of physical cores could be the bottleneck in executing parallel programs.

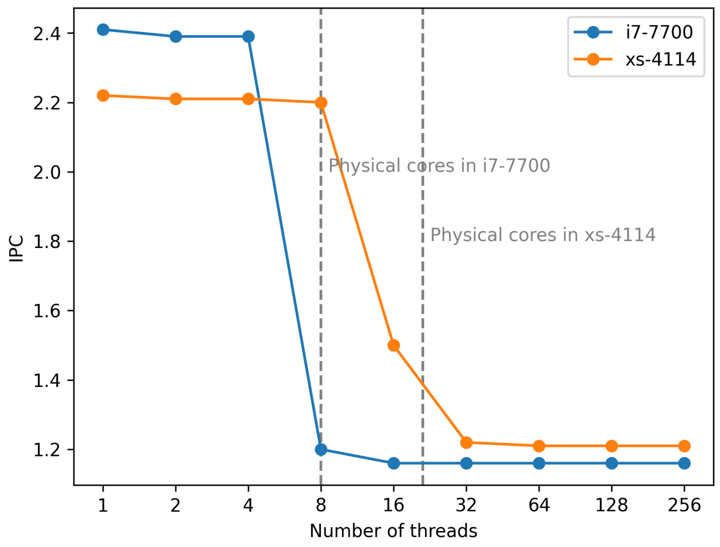

Figure 1: Comparison of IPC w.r.t. number of threads on different machines.

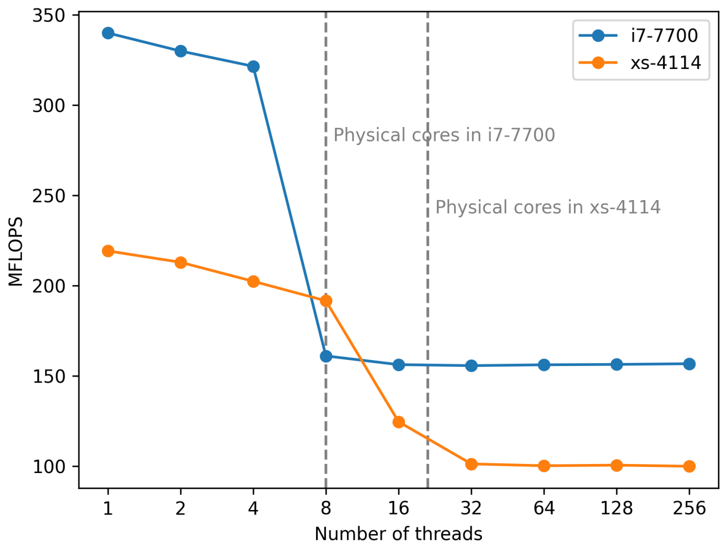

Figure 2: Comparison of MFLOPS w.r.t. number of threads on different machines.

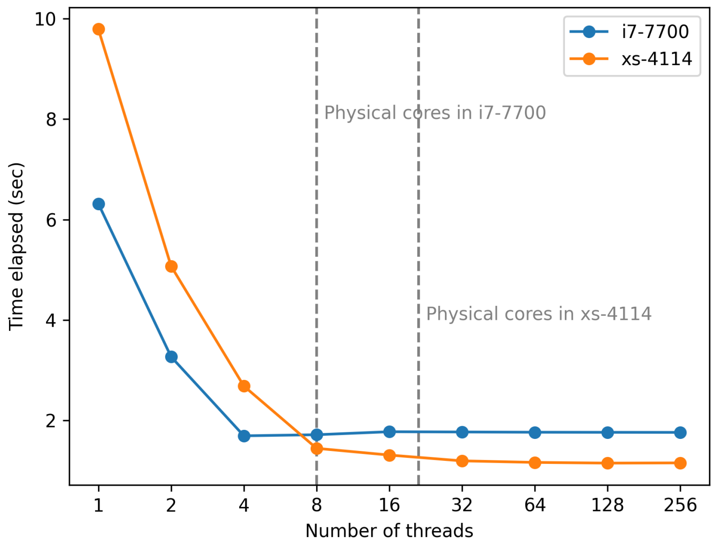

Figure 3: Comparison of execution time w.r.t. number of threads on different machines.

Exercise 7

The implementation of row-major access can be done by altering only a single line of the program. For example, by changing the matrix multiplication as follows:

result.element[i][k] += a.element[i][j] * b.element[j][k]

Alternatively, since the the elements are randomly generated, we can assume that one of the matrix is already transposed. For example, by swapping the indices to access the elements in B, thus the multiplication can be done row-wised. Therefore, the following implementation is also valid:

result.element[i][j] += a.element[i][k] * b.element[j][k]

Exercise 8

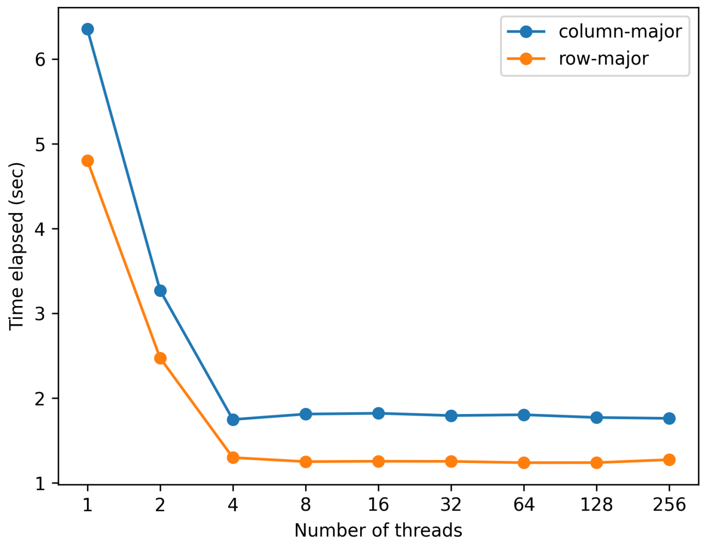

Figure 4: Comparison of execution time for row-major vs. column-major implementation (tested on i7-7700).

Two key observations:

1. Row-major is faster than column-major implementation.
2. Row-major has less L1D cache misses as compared to column-major implementation.

The observations should not be a surprise. Caches are organized into cache lines, and they are usually 64 bytes in most modern processors. The size of cache line is important with respect to spatial cache locality (e.g., how close the data are stored in memory). For instance, with row-major implementation, access is usually fast because elements reside in the same cache line.

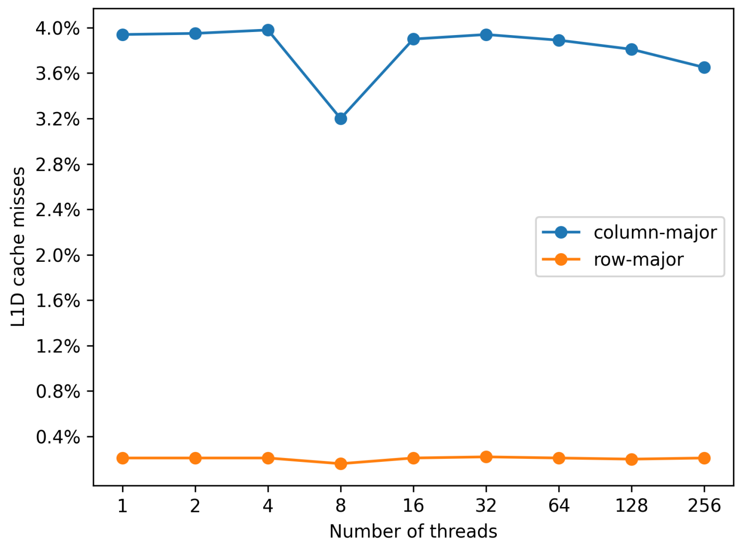

Figure 5: Comparison of L1D cache misses for row-major vs. column-major implementation (tested on i7-7700).

For column-major access, consecutive elements might not be in the same cache line (due to spatial locality). Even when the CPU is able to prefetch more than one cache line at a time, access could still be slow. For example, for large matrix, we might end up dealing with cache eviction. When accessing column-wised, by the time we get back to the first row, the entries might have already been evicted from the cache. Therefore, the CPU have to load them again from memory which increases execution time.