CS3210 Lab 2

Aaron Sng (A0242334N)

9 September 2021

Machines Used

- soctf-pdc-014 - Intel i7-7700 with 4 physical cores and 8 hyperthreads
- soctf-pdc-018 - Intel Dual Socket Xeon Silver 4114 with 10 physical cores and 20 hyperthreads

Exercise 4

As observed in figure 1, there is no linear relationship between increasing number of threads, and the other performance metrics which are IPC (Instructions Per Cycle) and MFLOPS (Million Floating Point Operations Per Second). The following phenomenon could be due to architectural limitations. With the number of threads lesser than or equal to the number of physical cores, it is observed that MFLOPS roughly increased proportionally to the number of threads used. In other words, doubling the number of threads would double MFLOPS. However, having more threads than physical cores on different machines, it is observed that the system performance gains were reduced. Specifically, for the i7 CPU, MFLOPS stayed relatively constant after more than 4 threads were used, while for the Xeon, MFLOPS continued to increase, though at a noticeably slower rate. This effect is also depicted in figure 3, where each thread begins to be less efficient in performing floating point operations after surpassing the physical core count. This phenomenon is also reflected in IPC, where IPC was on a downward trend after the number of threads used surpassed the actual physical number of cores on the machine. This phenomenon could be attributed to the various parallelism mechanisms present on both machines and due to their effectiveness at dealing with parallel tasks. For instance, the Xeon CPU has the Intel AVX-512 FMA unit to enhance it's effectiveness at instruction-level parallelism, while the i7 CPU doesn't has it. Another indicator is that the Xeon CPU has a much bigger cache than the i7 CPU (13.75MB vs 8MB). This allows for preserving the state of the matrix multiplication better in the Xeon, requiring overall lesser cache misses. The last indicator is that the Xeon CPU in maintaining multi-thread programs. The number of context switches are kept to a minimal on the Xeon CPU. At 256 threads, an average of 3734 context switches were recorded, while on the i7 an average of 4656 context switches.

Thread Count	1	2	4	8	16	32	64	128	256
IPC	2.30	2.38	2.41	1.19	1.16	1.13	1.13	1.14	1.17
MFLOPS	10022	19579	38998	37884	35949	35885	35952	36540	37038
MFLOPS per Thread	10022	9790	9750	4736	2247	1121	562	285	145

Table 1: 1024x1024 Matrix Multiplication on the Intel 7700

Thread Count	1	2	4	8	16	32	64	128	256
IPC	2.22	2.21	2.21	2.20	1.62	1.31	1.35	1.35	1.22
MFLOPS	6609	13173	25786	47753	55378	81089	101271	109699	112205
MFLOPS per Thread	6609	6587	6447	5969	3461	2534	1582	857	438

Table 2: 1024x1024 Matrix Multiplication on the Dual Socket Intel Xeon Silver 4114

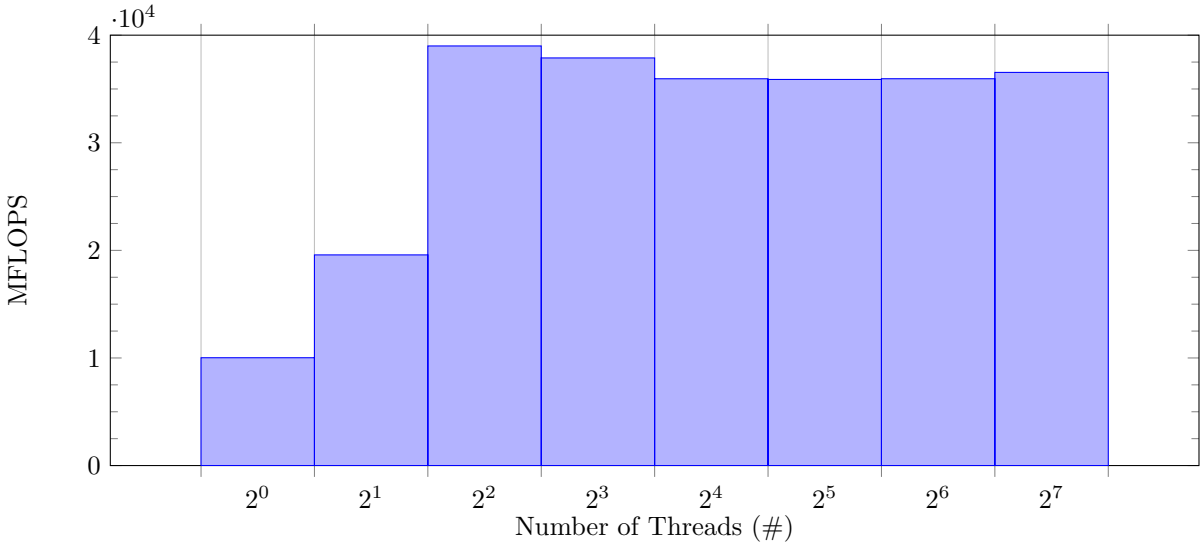

Figure 1: MFLOPS against Threads used in Matrix Multiplication on the Intel 7700

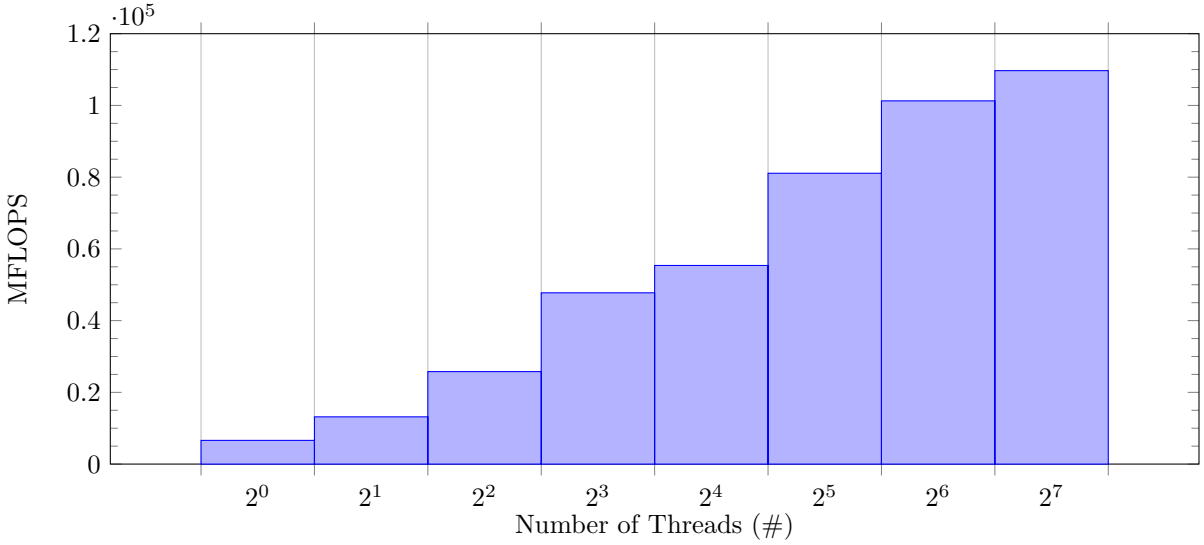

Figure 2: MFLOPS against Threads used in Matrix Multiplication on the Intel Xeon Silver 4114

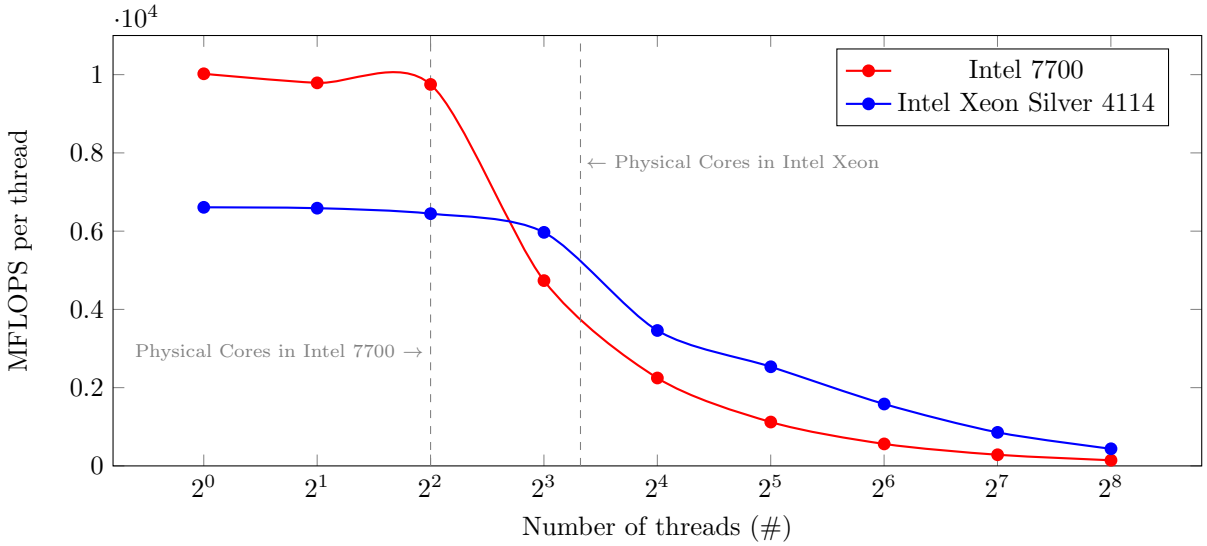

Figure 3: MFLOPS per Thread against Threads Used in Matrix Multiplication

Exercise 5

Currently the matrix multiplication in mm-omp.c assesses the column elements, which disrupts the row-major order of which matrices are used to be stored in memory. Hence, to implement the row-major order as required by the exercise, the transpose of B is taken and the elements are multiplied in the row-wise rather than column-wise. However, since the transpose is assumed to be taken as stated in the exercise, my implementation involved swapping the indices used to assess the elements in B and the multiplication was done row-wise. The following relation here describes the mathematical principle behind the following operation.

$$\begin{pmatrix} a & b \\ - & - \end{pmatrix}\begin{pmatrix} c & - \\ d & - \end{pmatrix} = \begin{pmatrix} a & b \\ - & - \end{pmatrix}\begin{pmatrix} c & d \\ - & - \end{pmatrix}^T = \begin{pmatrix} a\cdot b + c\cdot d & - \\ - & - \end{pmatrix}$$

Exercise 6

The row-wise implementation was much faster than the column-wise implementation provided. Overall, both metrics of IPC and MFLOPS increased. This could be attributed to the row-wise implementation increasing spatial locality of memory in the program, having just needing to assess neighbouring elements. Hence, this reduces the occurrence of cache misses increasing IPC and MFLOPS. In fact, L1 cache miss rate was about 0.20% using the row-wise operation while the it was about 3.80% using the column-wise operation. This eliminates redundancies used in assessing the memory especially during cache-misses, thereby speeding up the program.

Thread Count	1	2	4	8	16	32	64	128	256
IPC	3.15	3.14	3.12	1.66	1.68	1.67	1.67	1.64	1.67
MFLOPS	13258	25947	50822	53645	51644	52617	53033	52145	53154

Table 3: 1024x1024 Matrix Multiplication Row Wise on the Intel 7700

Thread Count	1	2	4	8	16	32	64	128	256
IPC	3.14	3.10	3.09	3.04	2.33	1.73	1.90	1.71	1.68
MFLOPS	9390	18548	35947	66570	80100	123570	130167	142479	147930

Table 4: 1024x1024 Matrix Multiplication Row Wise on the Dual Socket Intel Xeon Silver 4114