CS3210: Lab 3

Syed Abdullah

September 20, 2017

1 Overview

In this experiment, `mm-omp.c` was executed with varying number of threads and matrix sizes. The matrix sizes chosen are: 1, 256, 512, 1024, 1536, 2048. Each configuration is run **five** times and the relevant measurements recorded. The raw data from both the program (reported wall-clock time) and `perf` (cycles, instructions and runtime of program) are tabulated and presented in **Appendices A.1 & A.2**.

For calculation purposes, we will average out the five measurements and use the average values for time (both program & perf), cycle and instruction counts.

1.1 Machines Used

In this report, we shall refer to the **Dell Inspiron One 2320** machine as the **i5 machine** and the **Dell Optiplex 990** machine as the **i7 machine**.

The relevant specifications of the machines are presented in the table below.

	i5 machine	i7 machine
CPU	i5-2400	i7-2600
Clock Frequency	2.5GHz	3.4GHz
# cores	4	8

Table 1: Specifications of both test machines

2 Exercise 3: IPC & MFLOPS

2.1 Instructions per Cycle (IPC)

Average IPC can be calculated as follows:

$$IPC(A) = \frac{N_{instr}(A)}{N_{cycle}(A)}$$

Note that this refers to the *average* instructions executed per cycle and not the actual IPC, as that is dependent on the processor and instruction (e.g. different processors may use different number of cycles for a type of instruction).

The results of the IPC calculations are tabulated below. In order to identify the trends in the data, the data is also plotted onto a graph, which is also reproduced below.

2.1.1 Tabulations & Graphs

	1	256	512	1024	1536	2048
1	0.7798	2.5474	2.2861	1.8639	1.6239	1.5589
2	0.6981	2.1121	1.7198	2.0475	1.4472	1.3978
4	0.6386	1.6488	1.5456	1.4403	1.2729	1.3978
8	0.6888	2.1046	1.6905	1.9095	1.5679	1.4698
16	0.7186	2.3433	2.1446	1.7563	1.8588	1.5570
32	0.7525	2.5205	2.0443	1.9085	1.8146	1.6203

Table 2: Tabulation of IPC against # threads (row) & matrix size (column) for the i5 machine

	1	256	512	1024	1536	2048
1	0.7798	2.5474	2.2861	1.8639	1.6239	1.5589
2	0.6981	2.1121	1.7198	2.0475	1.4472	1.3978
4	0.6386	1.6488	1.5456	1.4403	1.2729	1.3978
8	0.6888	2.1046	1.6905	1.9095	1.5679	1.4698
16	0.7186	2.3433	2.1446	1.7563	1.8588	1.5570
32	0.7525	2.5205	2.0443	1.9085	1.8146	1.6203

Table 3: Tabulation of IPC against # threads (row) & matrix size (column) for the i7 machine

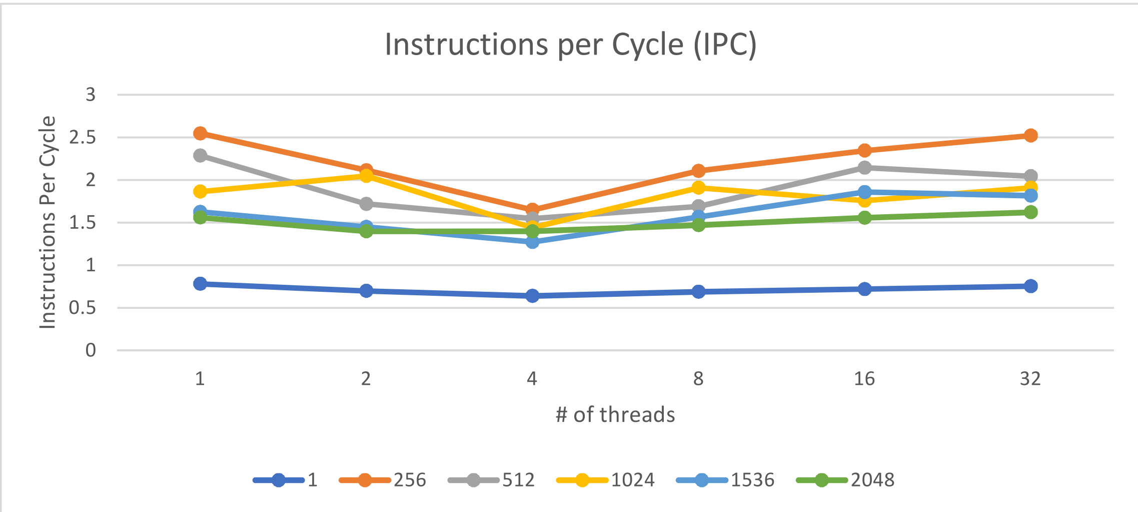

Figure 1: Plot of IPC with respect to # of threads on the **i5 machine**

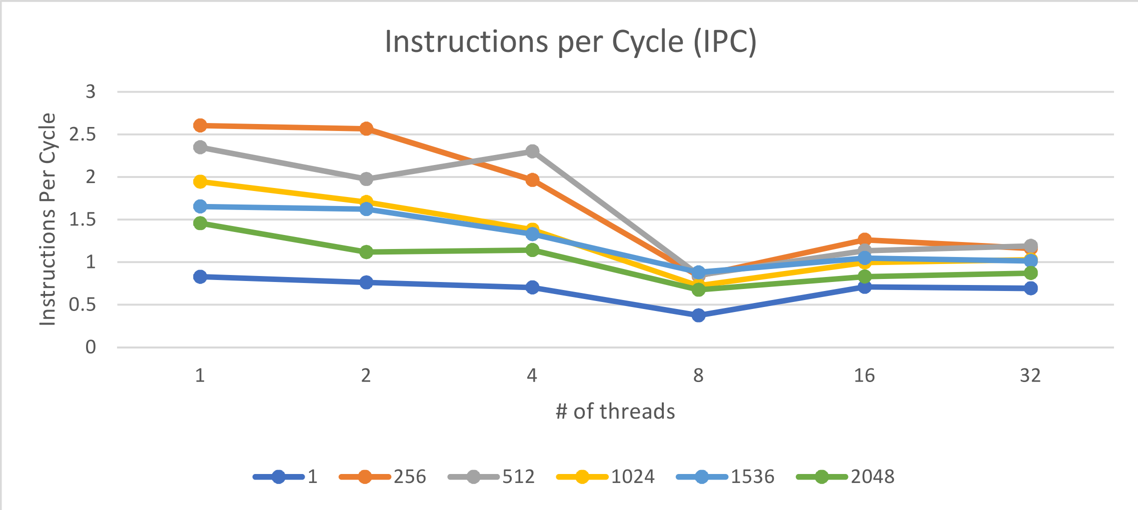

Figure 2: Plot of IPC with respect to # of threads on the **i7 machine**

2.1.2 Analysis of Results

From the graph, we can see two general trends when the number of threads increases: an initial decrease in the IPC up till a specific number of threads (common # of threads, albeit different between the two machines) and afterwards, a slight increase in the IPC.

This point where the IPC stops decreasing is significant. For the i5 machine, it seems that the IPC stops decreasing after 4 threads and on the i7 machine, after 8 threads. If we take a look at the number of cores that the two machines have, we can see that this specific # threads corresponds to the # of cores the CPU on the machine has – 4 cores for the i5 machine and 8 cores for the i7 one.

Thus, utilising this observation, we can make a conclusion on why the IPC value exhibits this trend with increasing number of threads. As we increase the # of threads performing the matrix multiplication, we are able to split the work up between multiple threads, with each thread taking up a fraction of the job.

For number of threads less than the number of cores, the decrease in IPC may be attributed to the contention between threads for the shared memory, which are the matrices `result`, `a` and `b`. If we assume that all threads are running on different cores (assuming the OS/OpenMP spreads it out), then all cores would simultaneously require access to the memory and thus, leading to contention, which results in threads have to wait for the resource (stall) to be free. This would mean that there's less instructions executed per cycle, as there would be more cycles where the thread is waiting to access the memory.

However, as we get past the number of cores, the IPC increases. Given the fact that there are now more threads than physical cores, multiple threads will now run on a shared core. If the matrices `a` and `b` are stored in the core or CPU level cache (based on the cache policies) there would be less contention, especially when reading values. This is because when the core switches to another thread in the same program, it does not have to refetch the matrices from memory, given that `a` and `b` are never written to. With that, it means that there's less waiting in any given clock cycle and higher probability that the actual CPU instructions are being executed.

This also explains why 1 thread has the highest IPC, as there's almost no contention for access to the memory and as such, there's virtually no stalls and waiting for memory in any given cycle.

2.2 Million Floating Point Operations Per Second (MFLOPS)

In order to calculate the MFLOPS, we have to first establish the definition of a **floating point operation**. In this report, we shall only be concerned about **floating point operations** that are explicitly stated in the code. From this definition, we can then determine the number of floating point operations that are performed in each execution of `mm-omp`.

In `mm-omp.c`, the main matrix multiplication code is:

```
#pragma omp parallel for shared(a, b, result) private (i, j, k)
for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
        for(k = 0; k < size; k++)
            result.element[i][j] += a.element[i][k] * b.element[k][j];
```

Observe the line where the actual calculation is performed. There are two calculations being performed, one is the multiplication between the two elements and another, the addition with the resultant element. The three loops repeats the operation `size` number of times, and since they are nested, there are a total of `size` × `size` × `size` operations of the statement performing the actual calculation. Thus, there are $2 \times sz \times sz \times sz = 2sz^3$ floating point operations performed (where sz is the size of the matrix).

To obtain the MFLOPS, we will divide the number of floating point operations by the time that is reported by the program (not perf). As the program starts timing right before the actual multiplication is done and ends it right after it is completed, the time reported would be the time taken to perform all the floating point operations that we are interested.

2.2.1 Tabulations & Graphs

	1	256	512	1024	1536	2048
1	0.0034	279.6203	253.2410	206.9664	180.4541	173.2400
2	0.0032	441.5057	345.0327	441.5057	302.5193	290.3280
4	0.0029	559.2405	524.2880	483.4497	430.7987	483.4769
8	0.0022	645.2775	571.1393	668.1654	542.2533	510.8191
16	0.0017	798.9150	741.5344	607.3200	649.4406	541.6442
32	0.0011	838.8608	710.1467	667.3349	632.7708	566.2822

Table 4: Tabulation of MFLOPS against # threads (row) & matrix size (column) for the i5 machine

	1	256	512	1024	1536	2048
1	0.0055	335.5443	306.4332	256.0796	217.6242	191.8895
2	0.0052	671.0886	491.6400	428.1267	427.0924	270.2342
4	0.0049	883.0114	1,167.1107	657.1247	631.9984	558.8040
8	0.0012	883.0114	808.5405	671.9286	865.0940	652.3340
16	0.0028	986.8951	1,056.8325	970.8335	1,033.3272	823.7375
32	0.0019	986.8951	1,118.4811	1,012.9640	1,004.4010	867.9332

Table 5: Tabulation of # threads, matrix size and MFLOPS for the i7 machine

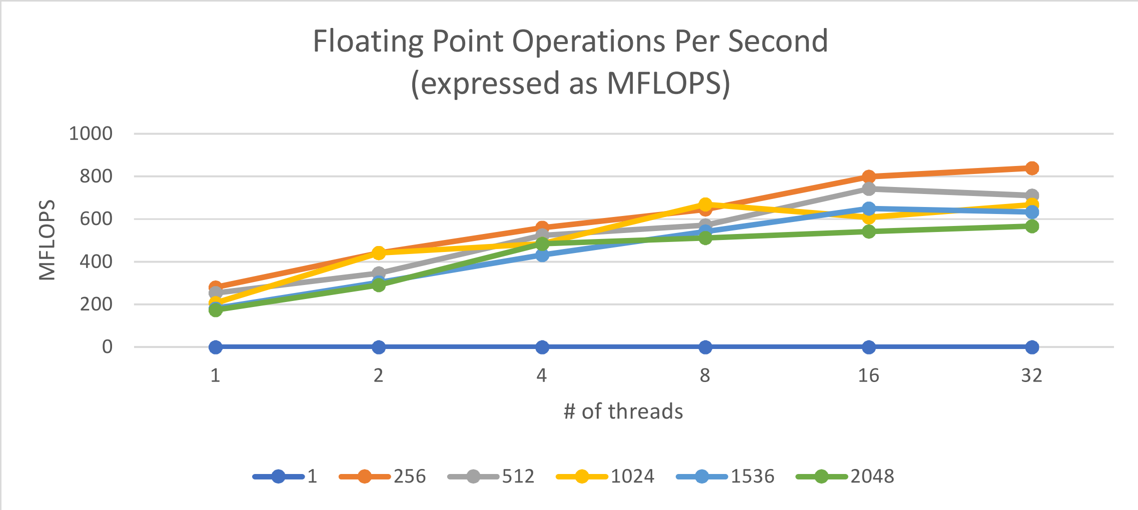

Figure 3: Plot of MFLOPS with respect to number of threads on the **i5 machine**

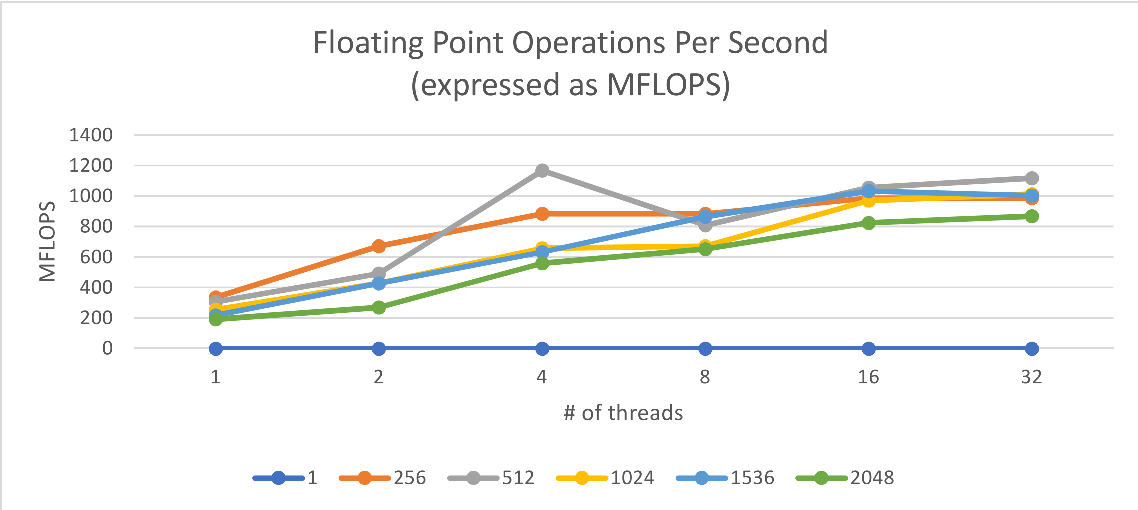

Figure 4: Plot of MFLOPS with respect to number of threads on the **i7 machine**

2.2.2 Analysis of Results

From the two graphs above, we can see an increase in the MFLOPS as the number of threads utilised increases.

Observing the code, we can see that the OpenMP directive splits the work in the outermost loop equally between threads. When we increase the number of threads, there are "more portions", each with less amount of floating point operations, as we divide the $2sz^3$ operations by the number of threads. Each thread will only work on its portion of the loop (let's ignore the data/memory dependency). Since the threads are executed in parallel, this would result in an increase in floating point operations performed per unit time, as threads perform floating point operations concurrently.

3 Exercise 4: Execution Time & Speedup

3.1 Execution Time

Execution time in this context refers to the **parallel execution time**. **Parallel Execution Time** consists of both the actual computations being performed (in this case, matrix multiplication) and the overhead time to allow for parallelism to occur (e.g. exchange of data between threads and spawning/destroying threads).

There are two "total runtime" reported when executing `mm-omp` with `perf`, coming from both the program and perf itself. However, as we have established earlier, the total time reported by the program is only for the matrix multiplication portion of the program and does not include the "setup" time (although it is trivial, as there's no overlapping problems).

	1	256	512	1024	1536	2048
1	0.0006	0.1219	1.0647	10.4068	40.2229	99.2708
2	0.0006	0.0802	0.7916	4.9093	24.0600	59.3504
4	0.0007	0.0624	0.5278	4.4886	16.9240	35.7142
8	0.0009	0.0567	0.4834	3.2639	13.4653	33.8079
16	0.0012	0.0482	0.3769	3.5831	11.2643	31.8961
32	0.0018	0.0440	0.3924	3.2655	11.5552	30.5184

Table 6: Execution time (s) on different matrix sizes and # of threads for the i5 machine

	1	256	512	1024	1536	2048
1	0.0004	0.1036	0.8802	8.4089	33.3494	89.6148
2	0.0004	0.0532	0.5571	5.0539	17.0529	63.7211
4	0.0004	0.0367	0.2418	3.3079	11.5527	30.8935
8	0.0016	0.0461	0.3448	3.2391	8.4633	26.4863
16	0.0007	0.0349	0.2648	2.2482	7.0952	21.0023
32	0.0010	0.0364	0.2510	2.1591	7.2996	19.9404

Table 7: Execution time (s) on different matrix sizes and # of threads for the i7 machine

3.2 Speedup

The equation for calculating speedup is as follows

$$S_p(n) = \frac{T_{seq}(n)}{T_p(n)}$$

where T_{seq} is the sequential execution time and T_p is the parallel execution time (with k threads).

The speedup for `mm-omp` is calculated and tabulated below

	1	256	512	1,024	1,530	2,048
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9241	1.5193	1.3449	2.1198	1.6718	1.6726
4	0.8427	1.9543	2.0173	2.3185	2.3767	2.7796
8	0.6513	2.1492	2.2025	3.1885	2.9872	2.9363
16	0.5010	2.5280	2.8246	2.9044	3.5708	3.1123
32	0.3287	2.7709	2.7135	3.1869	3.4809	3.2528

Table 8: Speedup achieved for different sizes & # of threads on the i5 machine

	1	256	512	1,024	1,530	2,048
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.9500	1.9455	1.5801	1.6639	1.9556	1.4064
4	0.8930	2.8210	3.6407	2.5421	2.8867	2.9008
8	0.2268	2.2478	2.5529	2.5961	3.9405	3.3834
16	0.5173	2.9668	3.3237	3.7402	4.7003	4.2669
32	0.3560	2.8474	3.5075	3.8947	4.5687	4.4941

Table 9: Speedup achieved for different sizes & # of threads on the i7 machine

4 Exercise 5: Determining the Sequential Fraction

4.1 Method 1: Utilising Amdahl's Law

The formula for calculating speedup of a parallel program utilising Amdahl's Law is

$$\text{Speedup} = \frac{1}{f + \frac{1-f}{p}}$$

where f is the sequential fraction of the program and p is the speedup of the parallel portion of the program.

Using the speedup that was calculated in 3.2, we can manipulate the speedup equation to ensure that f_p is on one side of the equation, like so:

$$\begin{aligned}
\text{Speedup} &= \frac{1}{f + \frac{1-f}{p}} \\
f + \frac{1-f}{p} &= \frac{1}{\text{Speedup}} \\
\frac{1 - f + p(f)}{p} &= \frac{1}{\text{Speedup}} \\
1 - f + p(f) &= \frac{p}{\text{Speedup}} \\
f(p-1) &= \frac{p}{\text{Speedup}} - 1 \\
f &= \frac{\frac{p}{\text{Speedup}} - 1}{p-1}
\end{aligned}$$

We can then solve for f, as we have the speedup and p, which is the number of threads.

4.1.1 Tabulation

	256	512	1,024	1,530	2,048
2	0.3164	0.4871	-0.0565	0.1963	0.1957
4	0.3489	0.3276	0.2417	0.2277	0.1464
8	0.3889	0.3760	0.2156	0.2397	0.2464
16	0.3553	0.3110	0.3006	0.2320	0.2761
32	0.3403	0.3482	0.2916	0.2643	0.2851

Table 10: Sequential fraction, f, for different sizes & # of threads on the i5 machine

	256	512	1,024	1,530	2,048
2	0.0280	0.2658	0.2020	0.0227	0.4221
4	0.1393	0.0329	0.1912	0.1285	0.1263
8	0.3656	0.3048	0.2974	0.1472	0.1949
16	0.2929	0.2543	0.2185	0.1603	0.1833
32	0.3303	0.2620	0.2328	0.1937	0.1974

Table 11: Sequential fraction, f, for different sizes & # of threads on the i7 machine

From these results, we can take the average of the values to obtain an approximation to the sequential fraction, f.

Type	f
i5 machine	0.276094819
i7 machine	0.207765723

Table 12: Average value for the sequential fraction f

4.2 Method 2: Measuring Sequential Part

In the previous section, we mentioned that there are two readings related to "execution time", one from `perf` and the other reported by the program. There is also a difference in the time reported from both sources, with `perf` having a larger reported time. This is due to the fact that the time reported by the program is for the matrix multiplication portion of the program and does not include the sequential portion of the program (e.g. matrix setup).

The sequential time can be measured by taking the difference between the `perf` time and time reported by the program.

	256	512	1,024	1,530	2,048
1	0.0154	0.0044	0.0030	0.0015	0.0010
2	0.0526	0.0172	0.0092	0.0042	0.0030
4	0.0379	0.0299	0.0104	0.0059	0.0050
8	0.0830	0.0277	0.0153	0.0074	0.0052
16	0.1288	0.0396	0.0132	0.0093	0.0056
32	0.0906	0.0366	0.0145	0.0088	0.0059

Table 13: Sequential fraction, f, for different sizes & # of threads on the i5 machine

	256	512	1,024	1,530	2,048
1	0.0345	0.0048	0.0027	0.0014	0.0009
2	0.0608	0.0199	0.0075	0.0049	0.0023
4	-0.0350	0.0487	0.0121	0.0073	0.0048
8	0.1753	0.0371	0.0133	0.0101	0.0057
16	0.0261	0.0409	0.0161	0.0114	0.0070
32	0.0653	0.0436	0.0181	0.0114	0.0073

Table 14: Sequential fraction, f, for different sizes & # of threads on the i7 machine

From these results, we can take the average of the values to obtain an approximation to the sequential fraction, f.

Type	f
i5 machine	0.023066546
i7 machine	0.022206108

Table 15: Average value for the sequential fraction f

4.3 Measuring Maximum Speedup Achievable

The maximum speedup that is theoretically achievable is when only the sequential fraction of the program execution remains (i.e. parallel portion of the program has insignificant runtime compared to the sequential portion). This means:

$$\text{Max Speedup} = \frac{1}{f}$$

Using the above equation, we get the following results:

Type	Method 1	Method 2
i5 machine	3.62	43.35
i7 machine	4.81	45.03

Table 16: Maximum achievable speedups

As the sequential fraction for Method 1 was derived from comparing between experiments with different # of threads, the speedup takes into account factors that would not be accounted for if we calculate the theoretical fraction (for instance, memory contention, method of parallelism used & machine-based constraints). Thus, the speedup that is calculated can be seen as the “maximum practical speedup” (although it may still not take into account other constraints, for instance, thread size larger than matrix size).

Method 2’s speedup is higher as the sequential fraction was derived based on measurements of the sequential portion, using data from the same # of threads. As such, it does not take into account the factors when varying the # of threads used.

A Measurements from program and `perf`

A.1 On the i5 machine

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000642984	1,074,022	771,538
1	0	0.0005339	954,882	784,390
1	0	0.000541842	959,040	749,928
1	0	0.000581824	969,513	746,239
1	0	0.000604752	979,251	797,601
256	0.12	0.120757886	391,547,344	1,002,475,503
256	0.12	0.124717826	395,072,326	1,002,482,678
256	0.12	0.121166539	392,910,163	1,002,422,396
256	0.12	0.122021302	395,013,804	1,002,470,735
256	0.12	0.120716531	393,089,664	1,002,455,114
512	1.06	1.065100607	3,485,741,926	7,967,176,303
512	1.06	1.063790669	3,485,832,109	7,967,187,085
512	1.06	1.06415438	3,480,189,135	7,967,223,350
512	1.06	1.06611828	3,487,094,768	7,967,253,941
512	1.06	1.06417568	3,486,577,692	7,967,205,116
1024	10.43	10.459986831	34,238,323,428	63,547,369,822
1024	10.31	10.342052608	33,869,364,915	63,546,995,863
1024	10.44	10.469909181	34,345,291,947	63,547,035,160
1024	10.39	10.41934678	34,138,499,453	63,547,401,630
1024	10.31	10.342909777	33,876,513,157	63,546,887,761
1536	42.25	42.309290825	138,848,835,651	214,261,264,310
1536	40.74	40.799152881	133,785,304,912	214,264,432,924
1536	39.17	39.228899816	128,640,621,862	214,262,808,379
1536	39.12	39.180193411	128,563,563,859	214,259,186,412
1536	39.54	39.596931601	129,880,861,251	214,258,219,439
2048	99.07	99.170937495	325,285,028,815	507,631,518,907
2048	99.08	99.184673672	325,461,064,820	507,629,500,939
2048	99.11	99.212297012	325,356,211,885	507,630,493,264
2048	99.23	99.329121207	325,858,589,006	507,628,991,699
2048	99.35	99.457078587	326,199,143,679	507,635,640,860

Table 17: Total execution time, # cycles and # instructions for **1 thread**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000680756	1346823	887410
1	0	0.000645308	1290700	897874
1	0	0.000638319	1307086	905972
1	0	0.000619214	1288054	904860
1	0	0.00056029	1185944	884434
256	0.06	0.065371187	402931838	1005032173
256	0.06	0.065326156	404628748	1005295877
256	0.06	0.065502713	403034013	1005248539
256	0.06	0.065244452	402610718	1006090758
256	0.14	0.139660793	767812092	1007243304
512	0.55	0.559115948	3506594788	7970838933
512	1.13	1.143252956	6426335748	7973578449
512	0.54	0.557781673	3499796046	7970514887
512	0.55	0.561050537	3508612717	7970817101
512	1.12	1.1369127	6235911169	7973486905
1024	4.48	4.528232002	28638140556	63552770266
1024	5.43	5.477339872	34683092430	63554456920
1024	4.52	4.562051653	28822199053	63551783825
1024	5.42	5.46521931	34535622104	63555444587
1024	4.47	4.513658627	28521412687	63553228941
1536	21.03	21.128132489	133907686842	214272923538
1536	20.6	20.703688923	131477696870	214271839242
1536	23.09	23.194582483	147289913121	214281097511
1536	33.94	34.046138732	192894692951	214292976852
1536	21.13	21.227405746	134751317227	214273120555
2048	52.06	52.239369612	331576119108	507652665620
2048	51.58	51.756742986	328498894280	507650868236
2048	51.93	52.10681386	330794462130	507654208981
2048	89.21	89.385172543	500170929620	507699984789
2048	51.09	51.26397622	324853062721	507651159669

Table 18: Total execution time, # cycles and # instructions for **2 threads**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000719741	1782740	1105663
1	0	0.000678562	1709953	1106759
1	0	0.000690291	1700676	1077694
1	0	0.000671275	1659610	1088267
1	0	0.000687734	1798808	1146251
256	0.04	0.041530667	412013769	1009834293
256	0.09	0.094974158	917330229	1015941774
256	0.04	0.042078268	417982105	1010275450
256	0.04	0.041713703	413607314	1009858613
256	0.09	0.091515271	907239973	1012863642
512	0.83	0.846252453	8020918228	7983159207
512	0.34	0.350557092	3530734462	7981202560
512	0.33	0.351279748	3543267080	7984175722
512	0.73	0.740587334	7190796877	7982810634
512	0.33	0.350229263	3537337784	7981861118
1024	3.26	3.309970241	33876753486	63560119519
1024	5.93	5.976485386	58730196279	63572770208
1024	6.61	6.655236968	64833329460	63577661413
1024	2.79	2.835922544	28910334462	63561173821
1024	3.62	3.665378817	34317280596	63567103955
1536	12.51	12.614236401	126542628057	214293680547
1536	22.4	22.500734297	219572987164	214329310940
1536	25.11	25.210036894	246169963109	214333272521
1536	12.27	12.36624724	127029935047	214285924556
1536	11.83	11.928598336	122475772176	214284975994
2048	29.95	30.131323096	309917395028	507686263378
2048	30.12	30.301652301	311661523954	507681965329
2048	31.69	31.870417587	327918596095	507683880398
2048	31.17	31.35040989	322487630628	507681150975
2048	54.74	54.917050418	544069402158	507756811357

Table 19: Total execution time, # cycles and # instructions for **4 threads**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000963353	2052038	1328565
1	0	0.000906363	1981786	1350636
1	0	0.000889746	1944239	1329048
1	0	0.00087875	1854369	1322244
1	0	0.000822731	1868834	1351898
256	0.05	0.051737947	396075617	1003408453
256	0.04	0.044465886	399668779	1003397157
256	0.06	0.063877872	542368993	1003476510
256	0.05	0.053988123	425306491	1003408012
256	0.06	0.069469784	620486609	1003504894
512	0.56	0.568777154	5543266763	7969873148
512	0.57	0.587841366	5770092354	7969991586
512	0.35	0.358418712	3495452491	7969154032
512	0.52	0.537706272	5263946913	7969947360
512	0.35	0.364204914	3499263061	7969108441
1024	3.23	3.277576497	33444332364	63555344906
1024	3.23	3.280204071	33436237072	63556013535
1024	3.17	3.221251997	32703300497	63553414716
1024	3.22	3.270665519	33437888175	63555836472
1024	3.22	3.269669756	33396257926	63555804685
1536	10.35	10.446450564	106824866071	214280658275
1536	17.5	17.600570259	175160797837	214308678673
1536	16.35	16.452475128	167738681653	214303543077
1536	11.47	11.566610257	118373955667	214285355872
1536	11.16	11.260314956	115285784383	214284213307
2048	26.72	26.898958635	276076752243	507681064016
2048	27.54	27.717029142	284025173236	507685591950
2048	42.95	43.123139049	437744351146	507740591167
2048	43.34	43.513419214	443984219334	507748475278
2048	27.61	27.786752822	285312933805	507685199472

Table 20: Total execution time, # cycles and # instructions for **8 threads**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.001131656	2849663	1971697
1	0	0.001267098	2836095	1927678
1	0	0.001142931	2522917	1879182
1	0	0.001145428	2538315	1895026
1	0	0.001112096	2541575	1875328
256	0.04	0.04955201	418486854	1003995556
256	0.04	0.048243028	403264917	1004046870
256	0.04	0.043504429	402251561	1004058608
256	0.05	0.054607192	519364198	1004106171
256	0.04	0.045147949	399062869	1004071843
512	0.34	0.358438071	3515317956	7969960397
512	0.34	0.355417265	3494906014	7969989971
512	0.34	0.355492137	3511560624	7970116878
512	0.44	0.453445404	4549387983	7970292304
512	0.35	0.361851273	3510211683	7970059413
1024	3.25	3.297488602	33521487107	63557455116
1024	3.2	3.249358157	33169623284	63555727023
1024	4.21	4.253209327	42464020016	63561129594
1024	4.14	4.184438524	41960163272	63560789425
1024	2.88	2.931108696	29828942513	63553807098
1536	11.08	11.186272916	114496558286	214287545667
1536	11.45	11.552449541	118305804821	214290802723
1536	11.03	11.134249185	114121581642	214287786820
1536	10.83	10.933605286	111890348598	214286371796
1536	11.41	11.514897901	117598987704	214288995629
2048	35.7	35.875149542	365192618118	507732129276
2048	35.42	35.596118854	361556565992	507723538642
2048	29.33	29.504275645	302909452557	507704981471
2048	28.85	29.034725642	298234573680	507704341572
2048	29.29	29.470175208	302529991606	507707941004

Table 21: Total execution time, # cycles and # instructions for **16 threads**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.001792011	4229211	3124820
1	0	0.001757909	4116870	3066052
1	0	0.00173902	4089864	3047404
1	0	0.00177065	4006786	3083014
1	0	0.001779729	4086782	3128195
256	0.04	0.044611669	399943336	1005338757
256	0.04	0.043647581	398201745	1005335896
256	0.04	0.044967576	400854248	1005350017
256	0.04	0.043256603	396972637	1005401189
256	0.04	0.043434333	398364105	1005330843
512	0.4	0.410499034	4078067770	7971831963
512	0.37	0.387326368	3811592612	7971694274
512	0.34	0.354409554	3502484686	7971478336
512	0.39	0.404239878	4061325793	7971637836
512	0.39	0.405305698	4043688450	7971798573
1024	3.24	3.290785529	33489551493	63561723684
1024	3.14	3.184948777	32465024964	63559941264
1024	3.24	3.28515788	33563077738	63561452812
1024	3.24	3.286836692	33595946399	63561164224
1024	3.23	3.279773378	33404771414	63560798062
1536	10.92	11.023037168	112923769150	214297548407
1536	12.74	12.839657631	130279779690	214305964593
1536	12.44	12.541728179	128316467294	214304988137
1536	10.61	10.709631111	109769822039	214296796365
1536	10.56	10.661844526	109208774541	214295535716
2048	32.25	32.435634305	331519997979	507740574697
2048	30.4	30.581800588	314332509867	507733937229
2048	30.68	30.858612332	317132138773	507734736392
2048	29.26	29.442126053	302405402513	507727092857
2048	29.1	29.273579013	301408027020	507724685251

Table 22: Total execution time, # cycles and # instructions for **32 threads**

A.2 On the i7 machine

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000424744	1,024,233	778,900
1	0	0.000370313	931,471	749,014
1	0	0.000355153	906,943	773,679
1	0	0.00034258	879,804	764,581
1	0	0.000339932	871,772	752,885
256	0.1	0.102226581	385,306,727	1,002,495,423
256	0.1	0.104263118	384,292,319	1,002,469,164
256	0.1	0.104481264	385,156,235	1,002,454,529
256	0.1	0.104523909	385,291,199	1,002,437,645
256	0.1	0.102365598	384,471,194	1,002,429,325
512	0.88	0.88922532	3,422,538,210	7,966,964,155
512	0.88	0.8818351	3,422,706,087	7,966,992,681
512	0.84	0.844681214	3,276,551,407	7,966,578,894
512	0.9	0.902376936	3,421,701,687	7,967,011,705
512	0.88	0.882877928	3,422,734,699	7,966,983,773
1024	8.19	8.215805589	31,924,498,876	63,543,301,044
1024	8.42	8.443678577	32,811,698,840	63,545,096,277
1024	8.5	8.521777691	33,077,407,773	63,544,517,938
1024	8.41	8.431286992	32,779,063,283	63,544,944,006
1024	8.41	8.431950697	32,768,599,504	63,544,871,134
1536	33.52	33.56392171	130,480,349,508	214,253,581,893
1536	32.85	32.89255356	127,865,827,828	214,252,270,619
1536	33.91	33.9561459	132,002,053,325	214,253,615,142
1536	33.24	33.28418472	129,320,981,777	214,253,329,921
1536	33	33.05040107	128,432,635,163	214,252,210,419
2048	89.29	89.376564	347,467,163,877	507,620,365,610
2048	89.64	89.72279453	348,792,962,106	507,619,574,361
2048	89.98	90.06871818	350,089,640,473	507,621,726,143
2048	89.34	89.4233078	347,619,092,978	507,622,980,523
2048	89.4	89.48244031	347,799,944,920	507,621,036,409

Table 23: Total execution time, # cycles and # instructions for **1 thread**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000416634	1,229,474	875,105
1	0	0.000374093	1,103,674	861,570
1	0	0.000378176	1,109,067	827,487
1	0	0.000384464	1,133,635	875,130
1	0	0.000375776	1,082,281	865,552
256	0.05	0.052835142	389,744,564	1,005,008,655
256	0.05	0.052973858	391,075,103	1,005,214,349
256	0.05	0.053868568	391,365,718	1,005,085,877
256	0.05	0.053239623	393,851,420	1,005,099,423
256	0.05	0.053268112	393,414,304	1,005,082,870
512	0.45	0.459274687	3,445,432,068	7,970,462,482
512	0.44	0.454455754	3,445,544,135	7,970,928,724
512	0.45	0.459015093	3,444,533,852	7,971,086,664
512	0.45	0.459295178	3,445,708,734	7,971,265,299
512	0.94	0.953255134	6,385,877,457	7,973,008,911
1024	4.21	4.252030994	32,875,694,208	63,553,098,264
1024	4.05	4.088457424	31,576,574,956	63,551,712,467
1024	4.23	4.264535569	32,886,132,419	63,552,088,173
1024	4.21	4.246478017	32,862,768,171	63,552,938,303
1024	8.38	8.417872627	56,090,206,448	63,558,459,510
1536	17.53	17.61674335	136,428,938,275	214,266,798,862
1536	16.5	16.58477846	128,096,815,198	214,264,607,349
1536	17.54	17.62026786	136,486,760,129	214,267,938,154
1536	16.49	16.57359909	128,226,141,933	214,264,575,577
1536	16.79	16.86929705	130,635,468,789	214,265,129,129
2048	45.47	45.61554324	353,734,009,205	507,640,395,590
2048	74.49	74.6413477	507,344,980,261	507,677,556,088
2048	76.68	76.82448088	533,311,750,828	507,683,529,603
2048	76.26	76.40603164	529,481,308,528	507,678,547,682
2048	44.97	45.11807097	349,779,450,809	507,642,340,860

Table 24: Total execution time, # cycles and # instructions for **2 thread**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000428932	1,575,669	1,032,042
1	0	0.000404011	1,523,815	1,077,984
1	0	0.000414477	1,494,972	1,078,006
1	0	0.00040646	1,503,427	1,079,315
1	0	0.000398464	1,461,497	1,047,553
256	0.07	0.06846761	948,962,095	1,015,029,043
256	0.03	0.028620501	404,817,143	1,009,707,564
256	0.03	0.028621361	404,976,488	1,009,716,823
256	0.03	0.029117237	407,858,428	1,009,915,797
256	0.03	0.028745082	406,903,271	1,009,809,276
512	0.23	0.241904056	3,473,080,905	7,977,247,102
512	0.23	0.244358947	3,503,392,927	7,979,361,669
512	0.23	0.240941257	3,458,542,967	7,975,501,813
512	0.23	0.240821019	3,457,234,944	7,975,936,325
512	0.23	0.24082298	3,457,246,941	7,975,609,062
1024	2.24	2.281318614	33,029,701,402	63,562,103,211
1024	2.22	2.263042506	32,921,233,510	63,562,103,512
1024	4.91	4.952117967	67,089,073,639	63,568,360,969
1024	2.23	2.265847998	32,989,622,721	63,562,052,280
1024	4.74	4.777106351	64,573,960,138	63,569,830,223
1536	12.3	12.38016971	154,006,027,548	214,281,772,411
1536	9.62	9.70632043	142,376,008,013	214,275,900,443
1536	9.59	9.677701759	141,731,349,170	214,281,875,672
1536	9.57	9.650902567	141,563,217,788	214,274,287,883
1536	16.26	16.34816795	226,635,197,535	214,298,776,425
2048	23.99	24.13561198	354,292,582,615	507,656,698,384
2048	39.66	39.81326458	562,835,253,579	507,704,183,837
2048	41.22	41.37244618	588,499,403,126	507,710,495,894
2048	24.82	24.96752661	367,031,087,800	507,655,267,196
2048	24.03	24.17865093	355,374,600,790	507,650,626,707

Table 25: Total execution time, # cycles and # instructions for **4 thread**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.001763397	35,725,624	13,185,451
1	0	0.002806405	61,714,894	22,606,203
1	0	0.001542404	30,680,045	11,520,616
1	0	0.001499213	29,591,837	11,105,830
1	0	0.000469328	3,106,108	1,672,238
256	0.06	0.06754866	1,847,538,650	1,026,834,789
256	0.03	0.035659421	835,024,567	1,057,802,667
256	0.02	0.029522992	822,737,558	1,032,009,488
256	0.02	0.027991976	783,506,604	1,019,460,227
256	0.06	0.069659456	1,907,450,625	1,054,339,229
512	0.48	0.488645277	13,220,238,185	8,025,114,300
512	0.21	0.2256939	6,412,142,216	8,020,036,527
512	0.22	0.229149437	6,511,373,722	8,017,853,254
512	0.53	0.544260345	14,568,219,049	8,023,007,036
512	0.22	0.236187866	6,599,708,752	8,028,342,868
1024	2.01	2.056299222	58,345,388,211	63,603,360,104
1024	1.98	2.023735487	57,748,657,703	63,608,565,808
1024	4.12	4.160948624	111,549,048,073	63,613,202,573
1024	4.14	4.181679017	111,913,270,256	63,615,579,321
1024	3.73	3.772716738	101,069,364,867	63,618,558,333
1536	6.94	7.029179666	205,131,356,301	214,308,013,356
1536	7.28	7.363051567	214,384,272,116	214,334,854,855
1536	7.53	7.609355155	222,280,563,602	214,304,001,152
1536	12.58	12.66803307	354,279,596,635	214,367,554,584
1536	7.56	7.646724389	223,131,716,810	214,332,473,664
2048	19.94	20.09316731	587,306,222,945	507,754,541,919
2048	30.48	30.62712518	858,120,377,264	507,826,063,155
2048	19.88	20.03052586	586,476,104,624	507,739,716,032
2048	29.68	29.8241189	844,824,289,387	507,798,497,971
2048	31.7	31.85664564	893,117,075,005	507,835,334,968

Table 26: Total execution time, # cycles and # instructions for **8 thread**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.000737355	2,965,416	2,010,506
1	0	0.000719752	2,753,149	1,963,400
1	0	0.000697947	2,697,020	1,907,796
1	0	0.000689441	2,685,287	1,937,866
1	0	0.000698126	2,707,523	1,949,604
256	0.03	0.030918661	688,732,273	1,004,068,698
256	0.03	0.030632494	677,979,593	1,004,184,308
256	0.03	0.032149812	684,083,515	1,004,174,620
256	0.05	0.049023718	1,258,396,868	1,004,383,923
256	0.03	0.031826422	672,481,520	1,004,118,527
512	0.23	0.239511827	6,220,898,851	7,970,188,018
512	0.36	0.375359124	10,250,448,053	7,971,196,290
512	0.23	0.237086008	6,266,184,591	7,970,215,647
512	0.22	0.233796385	6,217,217,453	7,970,159,018
512	0.23	0.23835438	6,225,156,487	7,970,229,523
1024	2.05	2.087727974	59,343,753,505	63,558,786,193
1024	2.03	2.063452523	59,251,121,088	63,557,824,216
1024	2.08	2.116882425	59,828,101,568	63,559,019,189
1024	2.85	2.888005921	81,715,163,718	63,564,702,837
1024	2.05	2.085156208	59,510,684,709	63,558,858,367
1536	6.81	6.892257753	198,988,455,423	214,294,341,157
1536	7.28	7.359160509	211,816,466,134	214,298,837,975
1536	7.59	7.674344115	222,184,529,409	214,303,833,592
1536	6.33	6.410055333	184,726,595,035	214,290,440,963
1536	7.06	7.139949018	206,688,876,845	214,296,336,905
2048	25.17	25.3171692	739,949,720,333	507,781,015,984
2048	17.98	18.12356226	528,740,990,032	507,720,880,285
2048	18.51	18.6588099	544,008,011,534	507,724,624,537
2048	24.58	24.72945445	720,828,855,532	507,773,283,595
2048	18.04	18.18240515	529,767,587,818	507,720,543,042

Table 27: Total execution time, # cycles and # instructions for **16 thread**

Matrix Size	Time (s)	Time on perf (s)	# cycles	# instructions
1	0	0.001168762	4,862,873	3,368,807
1	0	0.001015802	4,744,802	3,185,650
1	0	0.000981183	4,517,772	3,154,755
1	0	0.001016825	4,518,104	3,115,089
1	0	0.000965997	4,519,393	3,144,259
256	0.04	0.041812372	910,341,938	1,005,356,936
256	0.03	0.037904675	948,363,866	1,005,601,400
256	0.04	0.039926343	1,023,578,885	1,005,731,258
256	0.03	0.030573189	719,994,164	1,005,416,893
256	0.03	0.031653127	720,478,977	1,005,593,871
512	0.28	0.291193218	8,068,941,139	7,972,325,290
512	0.23	0.239112861	6,372,175,289	7,972,028,856
512	0.24	0.246597505	6,366,457,642	7,971,964,570
512	0.23	0.242216824	6,391,896,639	7,971,851,268
512	0.22	0.235635453	6,329,745,583	7,971,979,053
1024	2.03	2.068976024	58,920,821,399	63,561,997,481
1024	2.04	2.075052429	59,503,731,930	63,562,049,077
1024	2.04	2.084412402	59,377,603,058	63,562,312,030
1024	2.03	2.067588278	59,465,848,142	63,562,176,622
1024	2.46	2.499383157	72,392,991,977	63,566,452,270
1536	6.46	6.545398093	190,195,628,772	214,299,549,007
1536	7.29	7.373374291	214,547,112,546	214,307,487,044
1536	7.62	7.709814192	224,290,129,109	214,312,972,877
1536	6.15	6.228861857	180,642,158,801	214,296,870,153
1536	8.56	8.640440971	248,595,632,444	214,316,937,875
2048	19.53	19.67317093	573,666,383,133	507,753,813,146
2048	19.62	19.76810862	576,625,796,508	507,750,676,280
2048	19.88	20.0272224	584,923,596,200	507,750,655,178
2048	19.85	19.99866544	583,486,665,210	507,750,934,749
2048	20.09	20.23486964	590,974,411,393	507,752,749,933

Table 28: Total execution time, # cycles and # instructions for **32 thread**