Tiffany Wong
CS351
Lab 4 Report

A separate (typed) design document (named lab5-report.pdf) describing the results in a table format. You must evaluate the performance of the various parameters outlined and fill in the 2 tables specified to showcase the results. You must summarize your findings and explain why you achieve the performance you achieve, and how the results compare between the various approaches.

Finding Theoretical Throughput
FLOPS = (sockets) x (cores per socket) x (cycles per second) x (FLOPS per cycle)

sockets	1
cores per socket	2
cycles per second	2.3
flops per cycle	4

FLOPS per cycle for various processors [edit]

Microarchitecture	ISA	FP64
Intel Atom (Bonnell, Saltwell, Silvermont and Goldmont)	SSE3 (64-bit)	2
Intel Core (Merom, Penryn) Intel Nehalem[7] (Nehalem, Westmere)	SSE4 (128-bit)	4
Intel Sandy Bridge (Sandy Bridge, Ivy Bridge)	AVX (256-bit)	8
Intel Haswell[7] (Haswell, Devil's Canyon, Broadwell) Intel Skylake (Skylake, Kaby Lake, Coffee Lake, Whiskey lake, Amber lake)	AVX2 & FMA (256-bit)	16

```
tiffany@tiffwong-VirtualBox:/media/sf_student-05/lab5$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          2
On-line CPU(s) list:             0,1
Thread(s) per core:              1
Core(s) per socket:              2
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i5-7360U CPU @ 2.30GHz
```

Sockets and cores per socket are found by using the lscpu command.
Cycles per second is given from processor info, 2.3 GHz.
Flops per cycle is given from the processor I have and a Wikipedia table.

FLOPS = 1*2*2.3*4

Flops

Mode	Type	Size	Threads	Measured Time	Measured Throughput	Theoretical Throughput	Efficiency (%)
flops	single	small	1	2.434696	4.10999	18.4	22.33690217
flops	single	small	2	0.9319855	11.950403	18.4	64.94784239
flops	single	small	4	0.3871305	42.285369	18.4	229.811788
flops	single	medium	1	24.702621	4.0487725	18.4	22.00419837
flops	single	medium	2	9.3163715	11.96054	18.4	65.00293478
flops	single	medium	4	4.438728	37.017525	18.4	201.1822011
flops	single	large	1	246.3699715	4.061104	18.4	22.07121739
flops	single	large	2	93.536862	11.8875005	18.4	64.60598098
flops	single	large	4	40.9886385	38.603928	18.4	209.8039565
flops	double	small	1	1.8549705	6.1897675	18.4	33.64004076
flops	double	small	2	0.6161325	16.2535705	18.4	88.33462228
flops	double	small	4	0.2146585	50.4092505	18.4	273.9633179
flops	double	medium	1	18.4176425	6.231171	18.4	33.86505978
flops	double	medium	2	6.2004015	16.1351345	18.4	87.69094837
flops	double	medium	4	2.560994	45.114531	18.4	245.1876685
flops	double	large	1	184.7887625	6.188859	18.4	33.63510326
flops	double	large	2	62.0578625	16.1243955	18.4	87.63258424
flops	double	large	4	24.8010695	45.658162	18.4	248.1421848

My measured time and measured throughput show that with flops, it runs pretty average and maintains the average-ness of the timing. There were multiple outliers, but it was can be explained. For example, for mode=flops, type=single, size=large, and threads=1, the measured time is 246.3699715 seconds. It's a high number because it's the largest size it can go up to, 1000 billion operations and it's working in a single thread, which explains why it takes so long. Flops work best in small sizes and with 4 threads, which also makes sense because it's the smallest size I could run through the flops function (10 billion operations) and span it across 4 thread, working in 4 parallel threads, therefore making the process faster/fastest. The efficiency of all the different combinations of things is admittedly all over the place, but since my laptop's theoretical throughput (flops/sec) is 18.4 and the measured throughput isn't consistent, the efficiency ranges from 22%-273%.

Matrix

Mode	Type	Size	Threads	Measured Time	Measured Throughput	Theoretical Throughput	Efficiency (%)
matrix	single	small	1	0.4165185	2.7074805	18.4	14.71456793
matrix	single	small	2	0.2112305	5.290949	18.4	28.75515761
matrix	single	small	4	0.119077	9.3173595	18.4	50.63782337
matrix	single	medium	1	30.4129855	2.2458695	18.4	12.2058125
matrix	single	medium	2	15.872795	4.270431	18.4	23.20886413
matrix	single	medium	4	8.7760425	8.0007155	18.4	43.48214946
matrix	single	large	1	1915.624657	2.269127	18.4	12.33221196
matrix	single	large	2	1023.422236	4.202333	18.4	22.8387663
matrix	single	large	4	686.1575515	6.055843	18.4	32.91219022
matrix	double	small	1	0.7652915	1.4445145	18.4	7.850622283
matrix	double	small	2	0.4018455	2.810734	18.4	15.27572826
matrix	double	small	4	0.2202405	5.3196265	18.4	28.91101359
matrix	double	medium	1	50.06541	1.405034	18.4	7.636054348
matrix	double	medium	2	25.8813985	2.748557	18.4	14.93780978
matrix	double	medium	4	12.7207	5.5405615	18.4	30.11174728
matrix	double	large	1	2909.050807	1.493486	18.4	8.116771739
matrix	double	large	2	1523.919567	2.890514	18.4	15.70931522
matrix	double	large	4	819.8557145	5.545157	18.4	30.13672283

My measured time and measured throughput show that with matrix, the measured time is all over the place while the measured throughput has a pretty average range. There were several outliers in the measured time, but it was can be explained. For example, for mode=matrix, type=double, size=large, and threads=1, the measured time is 2909.050807 seconds. It's an extremely high number because it's the largest size it can go up to, a 16384x16384 matrix and it's working in a single thread, which explains why it takes so long. I can deduce it's because I could further optimize the matrix function in the code for more efficiency. Matrix works best in small sizes and with 4 threads, which also makes sense for the same reason as it does for flops. It's because it's the smallest size I could run through the matrix function (a 1024x1024 matrix) and span it across 4 thread, working in 4 parallel threads, therefore making the process faster/fastest. The efficiency of all the different combinations of things is better than how flops held up, but since my laptop's theoretical throughput (flops/sec) is 18.4 and the measured throughput isn't consistent, the efficiency ranges from 7%-50% (a smaller range than flops).