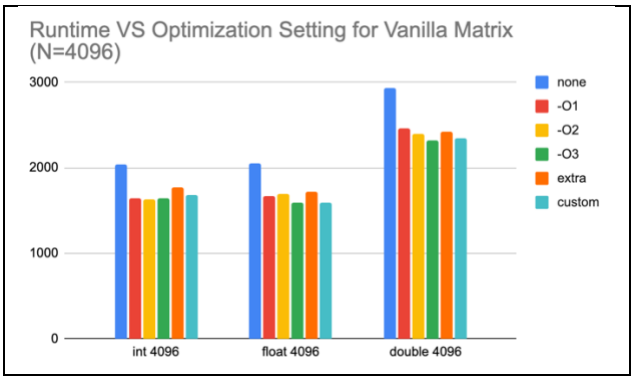
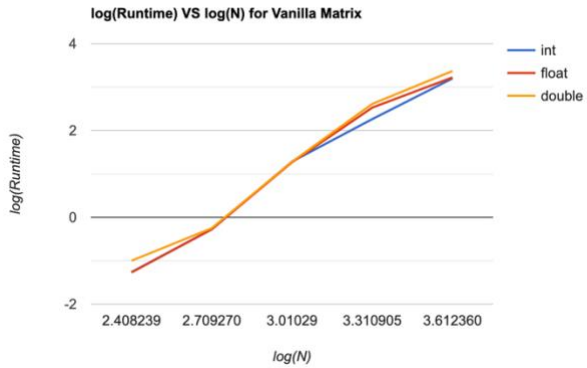
CPEN512 Assignment 0

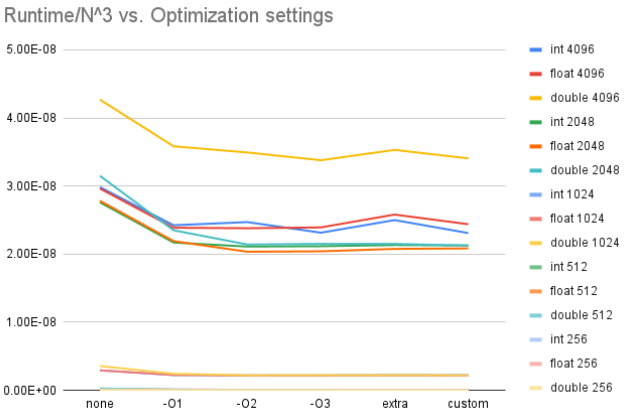
Jiaqi Zhang (**63174551**)

Question 3

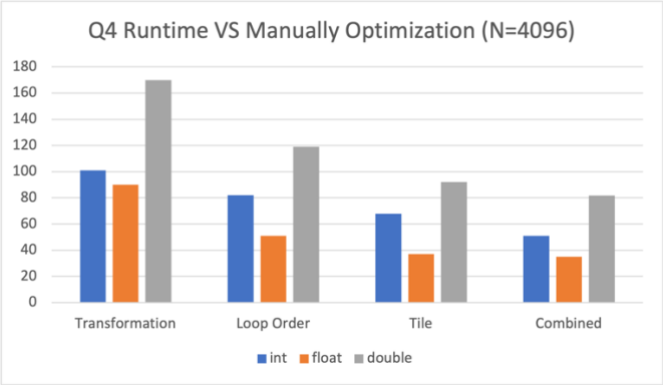
The custom flag used in question three is *-O3 -march=native -mtune=generics*.







Question 4

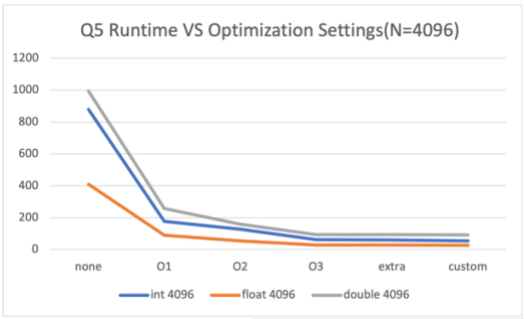


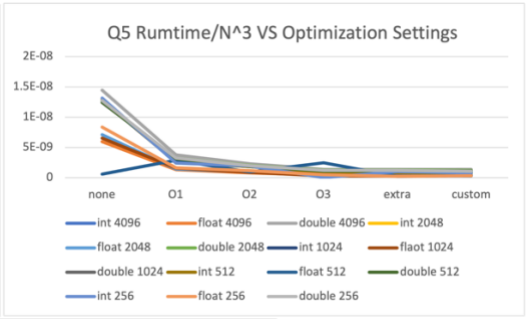
The best Optimization flag I found is *-Ofast -m64 -funroll-loops*.

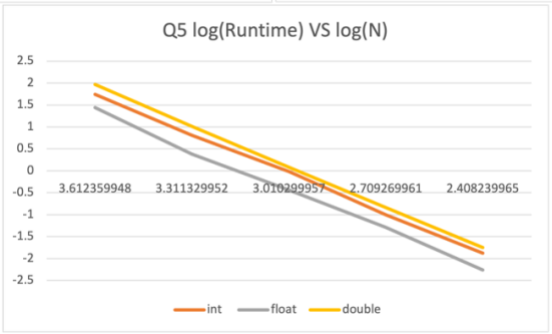
I implemented three Optimization strategies which are **Transformation Optimization, Loop Optimization, Tile Optimization and Combined Optimization(Strassen and Tile).** I see obvious

improvements for these four Optimization. The reason for the success of first three optimization is that they are all benefited from cache. They reduced the number of intervening iterations and thus data fetched between data reuses. The combined optimization performs best because it combined two optimization: Tile and Strassen. The key of Strassen Algorithm is that it substitutes one multiplication matrix by seven addition.

Question 5







Question 6: Float Table (in seconds)

	N=64	N=256	N=1024	N=4096
Vanilla -O2	0.000363	0.024269	2.822935	913.678227
Transformation -O2	0.000297	0.014939	1.624265	102.032924
Loop Order -O2	0.000257	0.015229	1.066267	78.621803
Tile -O2	0.000294	0.021816	1.184017	77.655803
Combined -O2	0.000364	0.015132	0.983029	67.686206
Vanilla -custom	0.000284	0.021959	2.353906	821.377503
Transformation -custom	0.000246	0.019864	1.343041	89.761551
Loop Order -custom	0.000052	0.003374	0.214317	51.57944
Tile -custom	0.000089	0.003487	0.307382	44.639761
Combined -custom	0.000209	0.005113	0.286522	37.71366

Question 7

1) Among the three data types, double performs the worst with the highest runtime. Integer and float runtimes are similar.
2) Among six Optimization Settings, none optimization performs the worst. -O1 is faster with a big difference from none. -O2, -O3, extra and custom flags are slightly faster than -O1. Optimization Settings make much bigger difference for Manually Optimized Matrix than Vanilla Matrix.
3) Since the time complexity of matrix multiplication is in order 3, we can observe that time cost grows quadratically as matrix size goes up.

Question 8

For float, size 4096, with custom flag: -Ofast -m64 -funroll-loops

Compile: gcc -Ofast -m64 -funroll-loops -D SET_FLOAT assign0_JiaqiZhang.c -o arr

Run: ./arr 4096

Results are as follows:

```
miley01@cpen512:~$ gcc -Ofast -m64 -funroll-loops -D SET_FLOAT assign0_JiaqiZhang.c -o arr
miley01@cpen512:~$ ./arr 4096
Vanilla Matrix time is 1550.521034 seconds
Transformation optimization Square Matrix time is 151.235641 seconds
Loop optimization Square Matrix time is 59.969424 seconds
Tile optimization Square Matrix time is 50.043450 seconds
Combined optimization Square Matrix time is 44.018705 seconds
```