HW2

Problem1

Output

```
Output from main_q1
----------------
Running main() from ./googletest-main/googletest/src/gtest_main.cc
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from RecurrenceTestFixture
[ RUN      ] RecurrenceTestFixture.GPUAllocationTest_1
[       OK ] RecurrenceTestFixture.GPUAllocationTest_1 (3553 ms)
[ RUN      ] RecurrenceTestFixture.InitalizeArrayTest_2
[       OK ] RecurrenceTestFixture.InitalizeArrayTest_2 (16827 ms)
[ RUN      ] RecurrenceTestFixture.RecurrenceKernelTest_3
Largest error found at pos: 3 error 9.73416e-08 expected 1.22465 and got 1.22465
Largest error found at pos: 32540 error 1.19207e-07 expected 1.00002 and got 1.00002
Largest error found at pos: 15176 error 2.38363e-07 expected 2.00046 and got 2.00046
Largest error found at pos: 621787 error 5.605e-07 expected 17.0147 and got 17.0147
Largest error found at pos: 621787 error 1.1559e-06 expected 290.417 and got 290.418
Largest error found at pos: 621787 error 2.40832e-06 expected 84343.2 and got 84343.4
Largest error found at pos: 621787 error 4.8222e-06 expected 7.11377e+09 and got 7.1138e+09
Largest error found at pos: 621787 error 9.6468e-06 expected 5.06057e+19 and got 5.06062e+19
Largest error found at pos: 822131 error 1.68278e-05 expected 6.36337e+33 and got 6.36348e+33
Largest error found at pos: 160732 error 2.63022e-05 expected 7.92214e+35 and got 7.92194e+35

Questions 1.1-1.3: your code passed all the tests!

[       OK ] RecurrenceTestFixture.RecurrenceKernelTest_3 (17360 ms)
[ RUN      ] RecurrenceTestFixture.RecurrenceThreadsTest_4
                Q1.4
-------------------------------------------
Number of Threads   Performance TFlops/sec
               32                1.47674
               64                2.95588
               96                4.41782
              128                 5.87104
              160                 8.88747
              192                 10.8214
              224                 12.6685
              256                 14.2902
              288                 12.1986
              320                 13.4498
              352                  14.694
              384                 15.8742
              416                 12.9832
              448                 13.9844
```

480	15.0394
512	15.5814
544	13.3867
576	13.7399
608	14.9094
640	15.5752
672	13.644
704	14.3129
736	15.053
768	15.0605
800	13.6576
832	14.4427
864	14.6467
896	15.3487
928	14.3418
960	14.3377
992	15.3028
1024	15.3571

[OK] RecurrenceTestFixture.RecurrenceThreadsTest_4 (88395 ms)
[RUN] RecurrenceTestFixture.RecurrenceItersTest_6
Q1.6

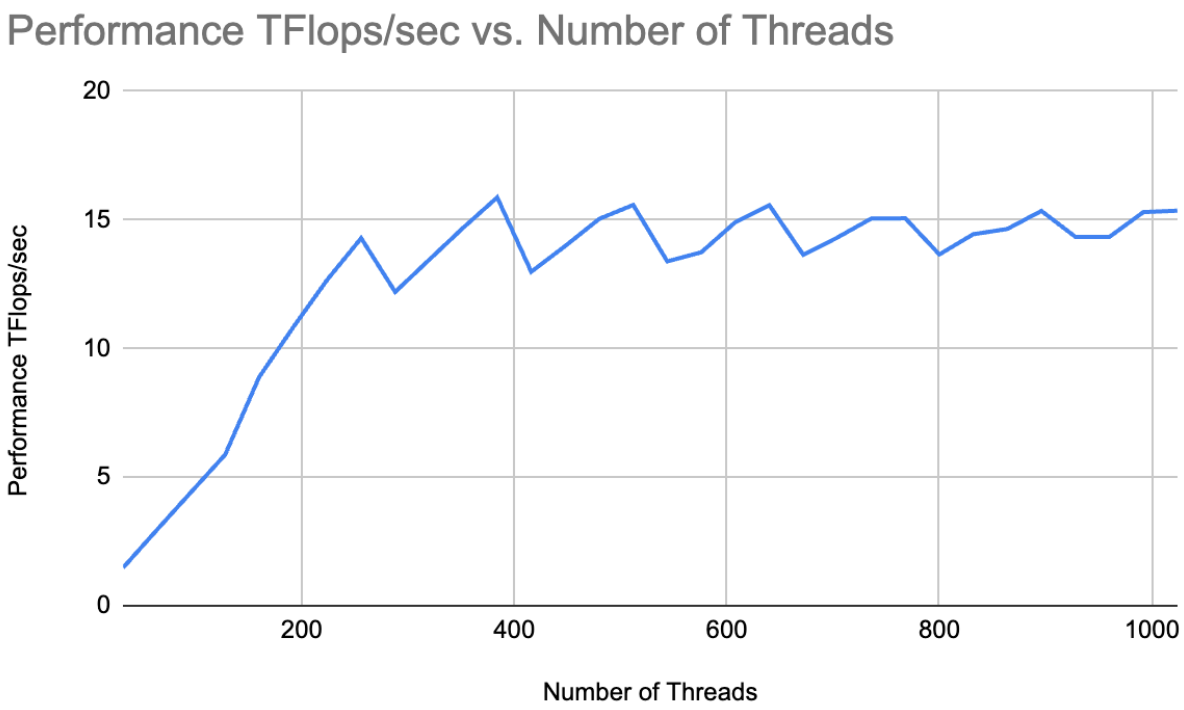
Number of Iters	Performance TFlops/sec
20	0.483372
40	3.25521
60	4.67581
80	6.13497
100	6.98324
120	7.8125
140	8.89228
160	9.5057
180	9.85114
200	10.1051
300	9.13743
400	9.38791
500	10.1891
600	10.2796
700	10.8025
800	11.0473
900	11.268
1000	11.3843
1200	11.6604
1400	11.794
1600	11.9933
1800	12.0812
2000	11.8304
2200	12.207
2400	12.288
2600	12.3255
2800	12.3824
3000	12.3412

[OK] RecurrenceTestFixture.RecurrenceItersTest_6 (64874 ms)

[----------] 5 tests from RecurrenceTestFixture (191021 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (191021 ms total)
[PASSED] 5 tests.

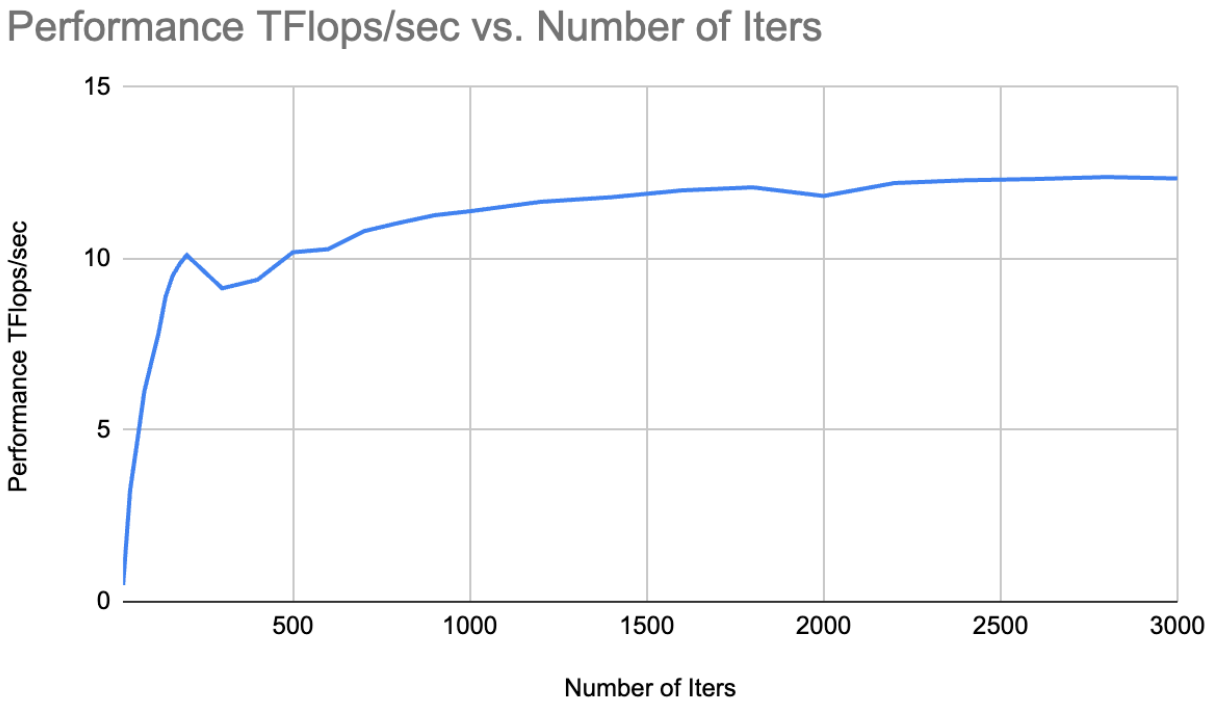
Question 1.4



The performance in TFlops per second goes up when the number of threads increases from 32 to 384. Then from 384 to 1024, it just fluctuates around the peak. For 32 threads to 384 threads, each thread block might not be using the full computational potential of each SM, leading to underutilization of the GPU. The performance here is likely limited by the inability to fully utilize each SM's computational power and by overheads associated with managing more blocks, and the low occupancy.

For threads greater than 384, the hardware limits (number of active warps, memory bandwidth, etc.) are reached, so no more improvement can be observed besides fluctuations. The concept of occupancy (the ratio of active warps to the maximum number of warps per SM) suggests that there is an optimal point where adding more threads does not contribute to better performance.

Question 1.5



In general, the more iterations you run, the better average performance in TFlops/sec you get. The reason is that the GPU starts up takes some time, and the performance in the initial iterations is very low. After warming up, the performance will be closer to its peak, and on average, the more number of iterations, the better the performance we will get.

Problem2

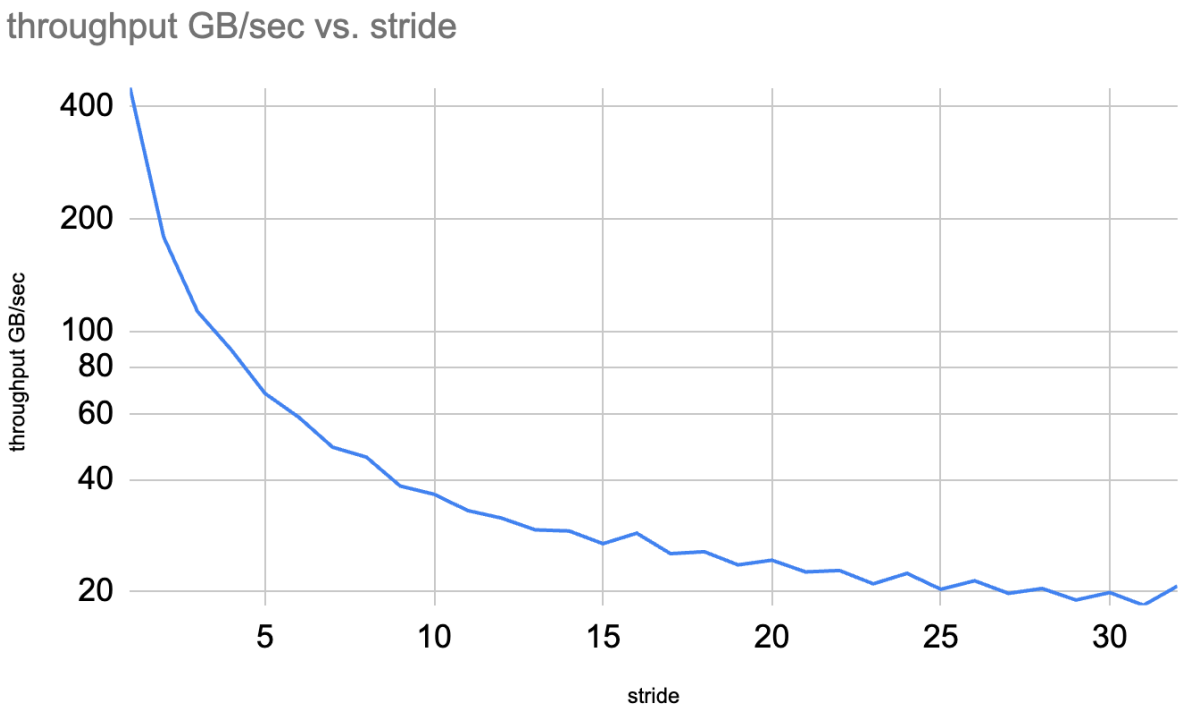
Output from main_q2

```
----------------
Running main() from ./googletest-main/googletest/src/gtest_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from testQ3
[ RUN      ] testQ3.StrideTest
# Using device: Quadro RTX 6000
# stride     time [ms]   GB/sec
      1      1.3347     449.5
      2      3.3595     178.6
      3      5.3188     112.8
      4      6.7389      89.0
      5      8.8195      68.0
```

```
 6     10.2181      58.7
 7     12.2991      48.8
 8     13.0804      45.9
 9     15.6256      38.4
10     16.4570      36.5
11     18.2037      33.0
12     19.0642      31.5
13     20.5049      29.3
14     20.6496      29.1
15     22.2786      26.9
16     20.9394      28.7
17     23.7187      25.3
18     23.4675      25.6
19     25.3890      23.6
20     24.6517      24.3
21     26.5042      22.6
22     26.2694      22.8
23     28.5838      21.0
24     26.7305      22.4
25     29.4844      20.3
26     28.0721      21.4
27     30.3696      19.8
28     29.3875      20.4
29     31.6615      19.0
30     30.2001      19.9
31     32.5383      18.4
32     28.9418      20.7
[       OK ] testQ3.StrideTest (85703 ms)
[----------] 1 test from testQ3 (85707 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (85708 ms total)
[  PASSED  ] 1 test.
```

Question 2.1



Question 2.2

The trend is that as the stride increases, the throughput goes down. The reason is that the bottleneck is the memory access traffic from/to global memory. For stride = 1, the memory access is coalesced in the best way. When threads within a warp access memory in a coalesced manner, the memory controller can efficiently fetch data with fewer memory transactions. Memory accesses that are cached in L2 only are serviced with 32-byte memory transactions. Each memory request from a warp is broken down into cache line requests that are issued independently. When memory accesses are aligned and coalesced, we get the maximum bandwidth. Bandwidth goes down if memory access is unaligned or memory access has a stride. The bigger the stride is, the more memory transactions there have to be, causing the throughput to be lower.

Problem 3

Output from main_q3

```
----------
Running main() from ./googletest-main/googletest/src/gtest_main.cc
[==========] Running 9 tests from 1 test suite.
```

```
[----------] Global test environment set-up.
[----------] 9 tests from testMatrix
[ RUN      ] testMatrix.deviceWarmup
[       OK ] testMatrix.deviceWarmup (2207 ms)
[ RUN      ] testMatrix.deviceSigmoid
[       OK ] testMatrix.deviceSigmoid (0 ms)
[ RUN      ] testMatrix.deviceRepeatColVec
[       OK ] testMatrix.deviceRepeatColVec (0 ms)
[ RUN      ] testMatrix.deviceSum
[       OK ] testMatrix.deviceSum (0 ms)
[ RUN      ] testMatrix.deviceSoftmax
[       OK ] testMatrix.deviceSoftmax (0 ms)
[ RUN      ] testMatrix.deviceCELoss
[       OK ] testMatrix.deviceCELoss (0 ms)
[ RUN      ] testMatrix.deviceElemArith
[       OK ] testMatrix.deviceElemArith (0 ms)
[ RUN      ] testMatrix.deviceSquare
[       OK ] testMatrix.deviceSquare (0 ms)
[ RUN      ] testMatrix.deviceSigmoidBackprop
[       OK ] testMatrix.deviceSigmoidBackprop (0 ms)
[----------] 9 tests from testMatrix (2209 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 1 test suite ran. (2209 ms total)
[  PASSED  ] 9 tests.
```