EECE5640 Homework 5

Question 1

a.)

Please see the details in the code part -> q1/q1a.cu

The file "q1a.cu" contains the code for this assignment section. The CUDA version used is 10.2, and the gcc version used is 6.4.0. The code's results when N is changed from 2^10 to 2^25 are shown in the section below.

```
[li.junce@c2189 q1]$ ./q1a
Total computing time: 0.736432
Elements of each class of the 100 bins:
54086, 161775, 294763, 363274, 491625, 525293, 626740, 700902, 841847, 988375, 1097327, 1146679, 1212588, 1360746, 1439417, 1
572722, 1696495, 1782075, 1882011, 1980206, 2078436, 2115655, 2254909, 2309960, 2443270, 2574689, 2654613, 2748966, 2882695,
2936781, 3081566, 3156580, 3260135, 3382429, 3442510, 3571931, 3604426, 3742009, 3857482, 3943884, 4011877, 4107879, 4261994,
 4340086, 4491051, 4526498, 4619491, 4759474, 4874269, 4960744, 5024013, 5177147, 5270750, 5394134, 5480420, 5575606, 5608718
, 5788125, 5835703, 5997065, 6088828, 6124420, 6256653, 6386541, 6472074, 6587466, 6665566, 6746052, 6846768, 6924634, 707909
1, 7194618, 7293600, 7307293, 7472771, 7559455, 7617778, 7743862, 7832820, 7936568, 8079592, 8149086, 8299390, 8361037, 84927
16, 8524111, 8655348, 8782777, 8830366, 8975602, 9084846, 9189303, 9237278, 9344946, 9432220, 9586369, 9615651, 9714657, 9864
671, 9922284,
Total number of elements counted: 1024
[li.junce@c2189 q1]$
```

```
[li.junce@c2189 q1]$ ./q1a
Total computing time: 0.785164
Elements of each class of the 100 bins:
93264, 182011, 284526, 313139, 471827, 545048, 626740, 797687, 864724, 962732, 1086240, 1118528, 1233369, 1347276, 1421186, 1
536471, 1655579, 1714006, 1840253, 1997239, 2012425, 2108917, 2246362, 2314021, 2400164, 2518057, 2678848, 2796691, 2877364,
2948918, 3037316, 3149001, 3238517, 3368496, 3459039, 3543311, 3667054, 3716849, 3879961, 3939405, 4068222, 4190525, 4211787,
 4388753, 4449070, 4526498, 4683850, 4770911, 4884844, 4952238, 5069147, 5149360, 5290373, 5345664, 5409778, 5570753, 5610073
, 5756617, 5858808, 5904071, 6065040, 6175822, 6297922, 6334409, 6493115, 6576430, 6629449, 6739960, 6877994, 6953268, 701969
5, 7119060, 7228347, 7353113, 7431383, 7585271, 7682056, 7733009, 7898068, 7925267, 8097970, 8129968, 8247867, 8325546, 84472
50, 8562269, 8617287, 8734138, 8884328, 8923238, 9094019, 9140517, 9278657, 9363492, 9470009, 9513640, 9681849, 9760562, 9880
995, 9991542,
Total number of elements counted: 32768
[li.junce@c2189 q1]$
```

```
[li.junce@c2189 q1]$ ./q1a
Total computing time: 0.867764
Elements of each class of the 100 bins:
37534, 190051, 248099, 385223, 461398, 516072, 656348, 782872, 811534, 985540, 1000448, 1130826, 1232782, 1387002, 1485366, 1
581394, 1674204, 1759429, 1872975, 1901757, 2016739, 2182889, 2287808, 2352358, 2466119, 2534721, 2629262, 2744140, 2877897,
2904354, 3063308, 3139938, 3289799, 3369259, 3485545, 3553821, 3631659, 3731027, 3828891, 3933806, 4081492, 4112573, 4214457,
 4337549, 4469986, 4546421, 4666593, 4797990, 4872036, 4968193, 5072097, 5147505, 5270860, 5344747, 5479565, 5580087, 5698384
, 5777697, 5836171, 5961922, 6093521, 6112667, 6200660, 6364963, 6426745, 6553266, 6663844, 6760015, 6864327, 6942018, 708625
7, 7155170, 7208104, 7394955, 7490578, 7519802, 7623032, 7785730, 7874072, 7910178, 8034267, 8190290, 8252157, 8365322, 84446
99, 8592863, 8694872, 8701521, 8870031, 8927346, 9074519, 9168967, 9278894, 9307936, 9467829, 9512670, 9651373, 9749332, 9858
519, 9910106,
Total number of elements counted: 1048576
```

```
[li.junce@c2189 q1]$ ./q1a
Total computing time: 1.530185
Elements of each class of the 100 bins:
44842, 144159, 222188, 334178, 404742, 586708, 640063, 728402, 826830, 906543, 1031712, 1101773, 1260754, 1321602, 1476600, 1
598432, 1696072, 1729778, 1875331, 1984562, 2042986, 2179823, 2261067, 2359311, 2423528, 2554146, 2648654, 2762347, 2864865,
2978957, 3015967, 3156480, 3239543, 3346078, 3431417, 3564425, 3607634, 3751032, 3889964, 3984367, 4083588, 4108280, 4280788,
 4366892, 4470336, 4596633, 4659239, 4798585, 4840824, 4961964, 5079973, 5118015, 5230593, 5303516, 5498309, 5526134, 5617685
, 5720060, 5872375, 5994924, 6088867, 6161217, 6210734, 6354350, 6411174, 6541831, 6636453, 6759465, 6844450, 6949297, 706880
6, 7118855, 7240383, 7310044, 7424073, 7539011, 7657974, 7788358, 7809963, 7980955, 8049782, 8143462, 8281183, 8311358, 84402
59, 8558606, 8616833, 8762627, 8884837, 8926775, 9003821, 9109720, 9256989, 9350039, 9484458, 9540383, 9666635, 9718281, 9847
516, 9980925,
Total number of elements counted: 33554432
```

Table: Performance of the histogramming kernel in CUDA

N	RunnigTime(s)
2^10	0.736432

N	RunnigTime(s)
2^15	0.785164
2^20	0.867764
2^25	1.530185

The runtime grows as the input does, with a 107.7% increase in computational time between N = 2^10 and 2^25.

b.)

Please see the details in the code part -> q1/q1b.c

The file "q1b.c" contains the code that was used to test the histogramming issue with OpenMP. The outputs from this implementation, which used the same inputs as the previous section from N = 2^10 to 2^25, are shown below.

```
[li.junce@login-00 q1]$ ./q1b
Total computing time: 1.343144
Elements of each class of the 100 bins:
32315, 161775, 284526, 347164, 433948, 519389, 699064, 700902, 876437, 988375, 1078483, 1199894, 1223259, 1390582, 1421186, 157807
2, 1659742, 1762388, 1840663, 1907092, 2046551, 2197879, 2226024, 2338917, 2440687, 2589061, 2604970, 2721893, 2882695, 2931774, 3
017631, 3129896, 3245414, 3318458, 3449430, 3540487, 3635983, 3787105, 3838320, 3943829, 4023788, 4139837, 4273277, 4365849, 44010
45, 4577156, 4614205, 4797771, 4817334, 4959774, 5074096, 5129324, 5235484, 5324410, 5480420, 5544482, 5636174, 5733186, 5891186,
5972798, 6088828, 6175822, 6231944, 6399788, 6407663, 6533049, 6667071, 6748495, 6878614, 6990754, 7082624, 7186663, 7275503, 7379
385, 7494431, 7555808, 7637638, 7733009, 7830992, 7984400, 8093706, 8196772, 8201462, 8391122, 8401877, 8505864, 8656790, 8705398,
 8899556, 8990861, 9047815, 9123905, 9237278, 9358517, 9469246, 9586369, 9635047, 9727750, 9843632, 9999935,
Total number of elements counted: 803
[li.junce@login-00 q1]$
```

```
[li.junce@login-00 q1]$ ./q1b
Total computing time: 1.732407
Elements of each class of the 100 bins:
22236, 138455, 278328, 360239, 443661, 562433, 672935, 700032, 812651, 994442, 1048158, 1148450, 1299760, 1369561, 1489925, 156626
1, 1676604, 1763783, 1847546, 1930579, 2044945, 2101130, 2215137, 2311483, 2402063, 2551781, 2676073, 2711775, 2876379, 2911765, 3
077990, 3110439, 3239833, 3352228, 3440862, 3502629, 3675906, 3716900, 3891941, 3943829, 4036146, 4144931, 4232093, 4307649, 44295
51, 4550194, 4687992, 4702983, 4853374, 4965696, 5017960, 5173728, 5236421, 5355102, 5454665, 5540684, 5638833, 5762352, 5803301,
5904305, 6004921, 6106012, 6226133, 6382394, 6427007, 6540286, 6682748, 6774483, 6855421, 6988453, 7044138, 7100889, 7232527, 7335
651, 7425637, 7562658, 7616086, 7786424, 7830992, 7972599, 8031232, 8138805, 8201921, 8371559, 8408077, 8532020, 8690256, 8722736,
 8878789, 8990531, 9084271, 9195524, 9259183, 9337925, 9423879, 9547219, 9646114, 9730037, 9807662, 9903263,
Total number of elements counted: 21371
[li.junce@login-00 q1]$
```

```
[li.junce@login-00 q1]$ ./q1b
Total computing time: 5.842574
Elements of each class of the 100 bins:
55170, 158918, 214874, 300241, 430357, 531352, 666441, 725116, 823549, 914554, 1038380, 1190528, 1288884, 1393598, 1457613, 159884
9, 1607823, 1708455, 1879457, 1957929, 2012806, 2148555, 2286560, 2331602, 2482341, 2515558, 2667301, 2791376, 2808371, 2977758, 3
097855, 3124652, 3255694, 3368827, 3469818, 3508385, 3611460, 3795689, 3819084, 3943564, 4082500, 4140136, 4200454, 4325118, 44340
65, 4521746, 4695985, 4729625, 4866088, 4982226, 5070670, 5143602, 5224400, 5333344, 5495245, 5583677, 5629684, 5751801, 5874500,
5958576, 6047310, 6177766, 6219462, 6337805, 6492575, 6565042, 6660594, 6794266, 6878117, 6954898, 7089518, 7115585, 7254026, 7304
296, 7444281, 7596082, 7655507, 7755393, 7874939, 7944688, 8001469, 8104799, 8210856, 8317275, 8465492, 8554281, 8694908, 8769226,
 8831654, 8971569, 9020783, 9131299, 9256426, 9304714, 9449184, 9503617, 9607894, 9776801, 9846982, 9953951,
Total number of elements counted: 731636
[li.junce@login-00 q1]$
```

```
[li.junce@login-00 q1]$ ./q1b
Total computing time: 25.615100
Elements of each class of the 100 bins:
79285, 141566, 217982, 378687, 480029, 589630, 648221, 782549, 899111, 997538, 1033130, 1149506, 1235363, 1388749, 1491533, 152075
7, 1661583, 1732515, 1855562, 1963552, 2043692, 2111889, 2273844, 2394940, 2417721, 2570224, 2628352, 2782793, 2826884, 2960686, 3
085019, 3141944, 3201086, 3396174, 3405428, 3556615, 3698004, 3790253, 3802280, 3943829, 4007028, 4140695, 4202430, 4355928, 44988
28, 4525865, 4695573, 4750755, 4850420, 4925482, 5053573, 5166396, 5267802, 5331484, 5431503, 5529351, 5667948, 5711152, 5866037,
5903189, 6015681, 6195402, 6212123, 6393060, 6450315, 6541523, 6677475, 6710567, 6885457, 6980537, 7017342, 7168188, 7204159, 7327
818, 7408586, 7521135, 7610272, 7789158, 7824760, 7984400, 8001115, 8148405, 8252124, 8383118, 8401877, 8563422, 8607318, 8765892,
 8821116, 8960735, 9005637, 9138039, 9250702, 9308398, 9416390, 9509915, 9699119, 9707189, 9891415, 9989891,
Total number of elements counted: 27208368
[li.junce@login-00 q1]$
```

Table: Performance of the histogramming kernel in OpenMP

N	RunnigTime(s)
2^10	1.343144
2^15	1.732407
2^20	5.842574
2^25	25.615100

One the one hand, the CPU and GPU no longer communicate with one another, which results in significantly lower absolute computing time values, CPU running time is more than GPU at the same condition. On the other hand, I notice CPU poor scalability in comparison to the GPU version increase between N=2^10 and 2^25, clearly demonstrating the scalability advantage of using GPUs.

Question 2

a.)

Please see the details in the code part -> q2/q2a.cu

The file "q2a.cu" contains the code used for this section of the assignment. Once more, the versions of CUDA and gcc used are 6.4.0 and 10.2, respectively.

The 6-element 3D stencil computation has been converted to a GPU implementation using the method of assigning one read of "b" and the corresponding constant multiplication and addition in "a" to each thread.

Nthreads = (n-2)^3 * 6 represents the total number of threads as being equal to the sum of the non-zero elements of "a" ((n-2)^3) and the number of reads of "b" per element of "a." We set the initial values of all the elements in "b" to 1, performed the stencil computation for low values of n (4), and printed the results to verify the code's output. All non-boundary elements of "a" should be equal to 4.8 (6*0.8) and 0 for the boundary values in the output for the code given above. The code output for the case of n=4 is shown in below, where each 4 by 4 matrix represents a different value of the k-index. The correct behavior of the code is then verified from there.

```
[[li.junce@c2195 q2]$ module load gcc/6.4.0
[[li.junce@c2195 q2]$ module load cuda/10.2
[[li.junce@c2195 q2]$ nvcc q2a.cu -o q2a
[[li.junce@c2195 q2]$ ./q2a
Total computing time for n=4: 0.808641
Total number of threads = 48
RESULTS a:
0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0

0.0  0.0  0.0  0.0
0.0  4.8  4.8  0.0
0.0  4.8  4.8  0.0
0.0  0.0  0.0  0.0

0.0  0.0  0.0  0.0
0.0  4.8  4.8  0.0
0.0  4.8  4.8  0.0
0.0  0.0  0.0  0.0

0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0

[li.junce@c2195 q2]$
```

Below are the computation time results for various "n" values. Here, we can see how well the GPU scales, with almost no change in computation time from n=4 to n=512. Beyond those points, there are more threads than the GPU can handle, and some operations are serialized, which lengthens the computation time.

```
Total computing time for n=4: 0.713142
Total number of threads = 48

Total computing time for n=8: 0.770687
Total number of threads = 1296

Total computing time for n=16: 0.786224
Total number of threads = 16464

Total computing time for n=32: 0.845023
Total number of threads = 162000

Total computing time for n=64: 0.834579
Total number of threads = 1429968

Total computing time for n=128: 0.834081
Total number of threads = 12002256

Total computing time for n=256: 0.996141
Total number of threads = 98322384

Total computing time for n=512: 1.753866
Total number of threads = 795906000
```

Table: Computing time performance for the proposed GPU implementation of the 6-element 3D stencil computaion

N	ComputingTime(s)

N	ComputingTime(s)
4	0.713142
8	0.770687
16	0.786224
32	0.845023
64	0.834579
128	0.834081
256	0.996141
512	1.753866

b.)

We can use tensorflow to accelerate my code on the GPU. TensorFlow supports running computations on a variety of types of devices, including CPU and GPU. They are represented with string identifiers for example:

- "/device:CPU:0": The CPU of your machine.
- "/GPU:0": Short-hand notation for the first GPU of your machine that is visible to TensorFlow.
- "/job:localhost/replica:0/task:0/device:GPU:1": Fully qualified name of the second GPU of your machine that is visible to TensorFlow.

If a TensorFlow operation has both CPU and GPU implementations, by default, the GPU device is prioritized when the operation is assigned. For example, tf.matmul has both CPU and GPU kernels and on a system with devices CPU:0 and GPU:0, the GPU:0 device is selected to run tf.matmul unless you explicitly request to run it on another device.

If a TensorFlow operation has no corresponding GPU implementation, then the operation falls back to the CPU device. For example, since tf.cast only has a CPU kernel, on a system with devices CPU:0 and GPU:0, the CPU:0 device is selected to run tf.cast, even if requested to run on the GPU:0 device.

Question 3

The'most advanced data center accelerator' architecture was unveiled in 2016 as the Pascal P100 architecture by NVIDIA. As already mentioned, this architecture was designed to enhance the efficiency of the computing equipment in data centers.

On the other hand, the Ampere A100 architecture, which was just unveiled (in May 2020), is NVIDIA's first 7nm architecture geared toward computations related to AI and neural networks. The A100 architecture uses third-generation Tensor Cores, which are processing units that speed up matrix multiplication, one of the most frequent operations in the field of AI and Deep Learning, among other advancements.

The technical features of NVIDIA's most recent architectures, the Pascal, Volta, and Ampere architectures, are summarized in Table: Specifications of the three most recent architectures from NVIDIA.

In addition to having tensor cores, which the P100 architecture lacks, we note that the Ampere architecture offers higher specifications in almost every category. It is important to note that the Ampere architecture has fewer tensor cores per GPU, which could give the impression that it performs worse. The A100 architecture's main differentiating feature, the use of fine-grained structured sparsity, allows it to achieve 20x the overall mixed-precision compute capabilities compared to P100. The main secret of the A100 performance improvements is the use of sparse matrix representations, which can significantly increase the throughput of general matrix multiplication as was studied in earlier homework assignments.

Table: Specifications of the three most recent architectures from NVIDIA

DATA CENTER GPU	NVIDIA TESLA P100	NVIDIA TESLA V100	NVIDIA A100
GPU Codename	GP100	GV100	GA100
GPU Architecture	NVIDIA Pascal	NVIDIA Volta	NVIDIA Ampere
GPU Board Form Factor	SXM	SXM2	SXM4
SMs	56	80	108
TPCs	28	40	54
FP32 Cores/SM	64	64	64
GPU Boost Clock	1480 MHz	1530 MHz	1410 MHz

References:

https://www.hardwaretimes.com/nvidia-ampere-architectural-analysis-a-look-at-the-a100-tensor-core-gpu/

https://technical.city/en/video/Tesla-P100-PCIe-16-GB-vs-Tesla-A100

https://en.wikipedia.org/wiki/Ampere_(microarchitecture)

Question 4

Please see the details in the code part -> q4/

Testing in P100 node

```
[[li.junce@c2194 q4]$ nvcc VecAddCUDA.cu -o VecAddCUDA
[[li.junce@c2194 q4]$ ./VecAddCUDA
 GPU calculation time 0.029408 msec
 final result: 1.000000
 [li.junce@c2194 q4]$
```

```
[li.junce@d1005 q4]$ pgcc -acc -Minfo=accel VecAddOpenACC.c -o VecAddOpenACC
pgcc-Warning-CUDA_HOME has been deprecated. Please, use NVHPC_CUDA_HOME instead.
vecaddgpu:
      6, Generating copyin(a[:n]) [if not already present]
         Generating copyout(c[:n]) [if not already present]
         Generating copyin(b[:n]) [if not already present]
      8, Loop is parallelizable
         Generating NVIDIA GPU code
          8, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
[li.junce@d1005 q4]$ ./VecAddOpenACC
GPU calculation time 2.754387 msec
final result: 1.000000
[li.junce@d1005 q4]$
```

Table: Performance of computing vector addition in P100

Method	RunnigTime(msec)
CUDA	0.029408
OpenACC	0.785164

Testing in A100 node

```
[li.junce@d1005 q4]$ nvcc VecAddCUDA.cu -o VecAddCUDA
[li.junce@d1005 q4]$ ./VecAddCUDA
 GPU calculation time 0.025152 msec
 final result: 1.000000
[li.junce@d1005 q4]$
```

```
[li.junce@d1005 q4]$ pgcc -acc -Minfo=accel VecAddOpenACC.c -o VecAddOpenACC
pgcc-Warning-CUDA_HOME has been deprecated. Please, use NVHPC_CUDA_HOME instead.
vecaddgpu:
      6, Generating copyin(a[:n]) [if not already present]
         Generating copyout(c[:n]) [if not already present]
         Generating copyin(b[:n]) [if not already present]
      8, Loop is parallelizable
         Generating NVIDIA GPU code
          8, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
[li.junce@d1005 q4]$ ./VecAddOpenACC
GPU calculation time 2.754387 msec
final result: 1.000000
```

Table: Performance of computing vector addition in A100

Method	RunnigTime(msec)
CUDA	0.025152
OpenACC	2.754387

The performance of employing CUDA is more effective in two separate nodes, despite the fact that OpenACC can speed up the run speed.

Question 5

Please see the details in the code part -> q5/

For the single precision, the files "q5_single.cu" contain the code used for this section of the assignment. Once more, the versions of CUDA and GCC used are 6.4.0 and 10.2, respectively. 11 different iteration values have been tested and measured for both versions (N = 2^10, 2^11,..., 2^20). Below presents the findings. From the result table, we can once more see how the GPU's scalability properties produce constant compute times for a range of input iteration values that rise exponentially in number.

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 1024

Please wait. Running...

Aproximated value of PI = 3.1406140327453613
SINGLE precision total computing time for n=1024: 0.730869
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 2048

Please wait. Running...

Aproximated value of PI = 3.1411006450653076
SINGLE precision total computing time for n=2048: 0.826870
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 4096

Please wait. Running...

Aproximated value of PI = 3.1413505077362061
SINGLE precision total computing time for n=4096: 0.718903
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 8192

Please wait. Running...

Aproximated value of PI = 3.1414725780487061
SINGLE precision total computing time for n=8192: 0.733090
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 16384

Please wait. Running...

Aproximated value of PI = 3.1415269374847412
SINGLE precision total computing time for n=16384: 0.720390
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 32768

Please wait. Running...

Aproximated value of PI = 3.1415696144104004
SINGLE precision total computing time for n=32768: 0.790834
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 65536

Please wait. Running...

Aproximated value of PI = 3.1415817737579346
SINGLE precision total computing time for n=65536: 0.771916
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 131072

Please wait. Running...

Aproximated value of PI = 3.1415886878967285
SINGLE precision total computing time for n=131072: 0.760510
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 262144

Please wait. Running...

Aproximated value of PI = 3.1415917873382568
SINGLE precision total computing time for n=262144: 0.925196
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 524288

Please wait. Running...

Aproximated value of PI = 3.1415936946868896
SINGLE precision total computing time for n=524288: 0.732396
[li.junce@c2195 q5]$
```

```
[li.junce@c2195 q5]$ ./q5_single
Leibniz's series is used to approximate the number PI in a GPU implementation...
Enter the number of iterations: 1048576

Please wait. Running...

Aproximated value of PI = 3.1415960788726807
SINGLE precision total computing time for n=1048576: 0.729117
[li.junce@c2195 q5]$
```

Performance of the single precision Leibniz's series Pi computation using a GPU

N	ComputingTime(s)	Pi
2^10	0.730869	3.1406140327453613
2^11	0.826870	3.1411006450653076
2^12	0.718903	3.1413505077362061
2^13	0.733090	3.1414725780487061
2^14	0.720390	3.1415269374847412
2^15	0.790834	3.1415696144104004
2^16	0.771916	3.1415817737579346
2^17	0.760510	3.1415886878967285
2^18	0.925196	3.1415917873382568
2^19	0.732396	3.1415936946868896
2^20	0.729117	3.1415960788726807

The results for double precision have not been included because the proposed code uses Cuda's "atomicAdd" function, which is incompatible for the GPU in question with double precision input variables. The suggested double atomic implementation has been attempted, but it has also failed.