Question 1: (I’m using my own laptop with an nvidia gpu)

a How many GPUs are attached to mcu?

Two GPUs are attached to mcu

b What is the complete product name of each GPU?

GPU 0: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-cca6f7c7-a959-b848-3875-e5eddaabd222)

GPU 1: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-b3c99acd-d0c0-a736-b04e-1be9c8801078)

c What are the minimum, maximum, and default power limits?

nvidia-smi -q -d power

GPU 00000000:03:00.0

Power Readings

Power Management : Supported

Power Draw : 39.61 W

Power Limit : 260.00 W

Default Power Limit : 260.00 W

Enforced Power Limit : 260.00 W

Min Power Limit : 100.00 W

Max Power Limit : 320.00 W

Power Samples

Duration : 0.60 sec

Number of Samples : 31

Max : 40.43 W

Min : 17.26 W

Avg : 36.06 W

GPU 00000000:04:00.0

Power Readings

Power Management : Supported

Power Draw : 24.29 W

Power Limit : 260.00 W

Default Power Limit : 260.00 W

Enforced Power Limit : 260.00 W

Min Power Limit : 100.00 W

Max Power Limit : 320.00 W

Power Samples

Duration : 0.07 sec

Number of Samples : 4

Max : 33.52 W

Min : 22.45 W

Avg : 27.41 W

d What power limit is currently set?

Power Limit : 260.00 W

e What is the GPU shutdown temperature?

nvidia-smi -q -d temperature

GPU 00000000:03:00.0

Temperature

GPU Current Temp : 36 C

GPU Shutdown Temp : 94 C

GPU Slowdown Temp : 91 C

GPU Max Operating Temp : 89 C

GPU Target Temperature : 84 C

Memory Current Temp : N/A

Memory Max Operating Temp : N/A

GPU 00000000:04:00.0

Temperature

GPU Current Temp : 35 C

GPU Shutdown Temp : 94 C

GPU Slowdown Temp : 91 C

GPU Max Operating Temp : 89 C

GPU Target Temperature : 84 C

Memory Current Temp : N/A

Memory Max Operating Temp : N/A

f What is the maximum SM clock?

nvidia-smi -q -d CLOCK

GPU 00000000:03:00.0

Temperature

GPU Current Temp : 36 C

GPU Shutdown Temp : 94 C

GPU Slowdown Temp : 91 C

GPU Max Operating Temp : 89 C

GPU Target Temperature : 84 C

Memory Current Temp : N/A

Memory Max Operating Temp : N/A

GPU 00000000:04:00.0

Temperature

GPU Current Temp : 35 C

GPU Shutdown Temp : 94 C

GPU Slowdown Temp : 91 C

GPU Max Operating Temp : 89 C

GPU Target Temperature : 84 C

Memory Current Temp : N/A

Memory Max Operating Temp : N/A

[tyao0625@mcu ~]$ clear

[tyao0625@mcu ~]$ nvidia-smi -q -d CLOCK

==============NVSMI LOG==============

Timestamp : Thu Mar 10 22:45:23 2022

Driver Version : 470.63.01

CUDA Version : 11.4

Attached GPUs : 2

GPU 00000000:03:00.0

Clocks

Graphics : 1350 MHz

SM : 1350 MHz

Memory : 7000 MHz

Video : 1245 MHz

Applications Clocks

Graphics : N/A

Memory : N/A

Default Applications Clocks

Graphics : N/A

Memory : N/A

Max Clocks

Graphics : 2160 MHz

SM : 2160 MHz

Memory : 7000 MHz

Video : 1950 MHz

Max Customer Boost Clocks

Graphics : N/A

SM Clock Samples

Duration : Not Found

Number of Samples : Not Found

Max : Not Found

Min : Not Found

Avg : Not Found

Memory Clock Samples

Duration : Not Found

Number of Samples : Not Found

Max : Not Found

Min : Not Found

Avg : Not Found

Clock Policy

Auto Boost : N/A

Auto Boost Default : N/A

GPU 00000000:04:00.0

Clocks

Graphics : 1350 MHz

SM : 1350 MHz

Memory : 7000 MHz

Video : 1245 MHz

Applications Clocks

Graphics : N/A

Memory : N/A

Default Applications Clocks

Graphics : N/A

Memory : N/A

Max Clocks

Graphics : 2160 MHz

SM : 2160 MHz

Memory : 7000 MHz

Video : 1950 MHz

Max Customer Boost Clocks

Graphics : N/A

SM Clock Samples

Duration : Not Found

Number of Samples : Not Found

Max : Not Found

Min : Not Found

Avg : Not Found

Memory Clock Samples

Duration : Not Found

Number of Samples : Not Found

Max : Not Found

Min : Not Found

Avg : Not Found

Clock Policy

Auto Boost : N/A

Auto Boost Default : N/A

Question 2:

a How many total cores are available on each GPU?

(068) Multiprocessors, (064) CUDA Cores/MP: 4352 CUDA Cores

Each GPU has 4352 CUDA Cores

b What is the maximum dimension size of a thread block?

Max dimension size of a thread block (x,y,z): (1024, 1024, 64)

c What is the maximum dimension size of a grid size?

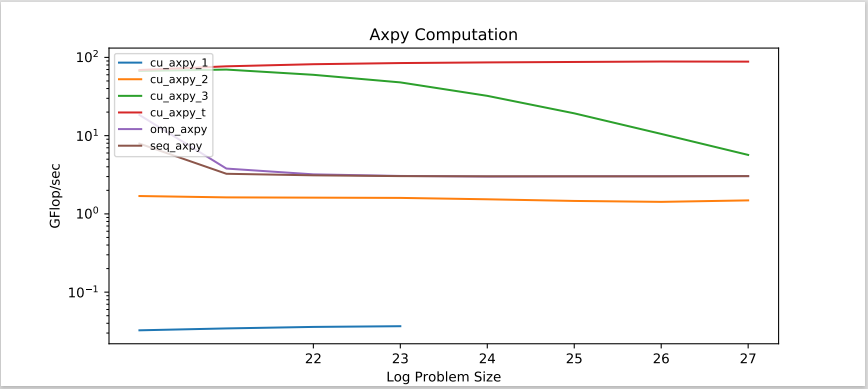
Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)

d What is the CUDA capability level of the GPUs?

CUDA Capability Major/Minor version number: 7.5

Question 3:

Cu\_axpy\_1 is only invoking 1 block per grid and 1 thread per block, whereas Cu\_axpy\_2 invokes 1 block per grid and 256 threads per block. So there’re 255 more threads running in version 2 comparing to version 1. Under the most optimum assumption, based on the output reading of version 1, we assume it’s the performance of exactly 1 thread per second, that is, 0.0375 glops / sec per thread. Because version 2 has 256 more threads, we times the per thread performance with 256 and the result is 9.6 gflps. But in reality we only got 2.5 gflps on version 2. There’re two floating operations per madd function and N is 2^16 by default, which means there’re 2\*2^16=float operation, the expected speedup = expected axpy\_1 time / expected axpy\_2 time = (2\*2^16 / 0.0375 gflops) / (2\*2^16 / 2.5 gflops) = 26.6 / 0.4 = 66.5. But in reality speedup = 116ms/169ms = 0.69. This is also what I saw from my plot shown as follow:



Question 4:

Cu\_axpy\_3 has 256 blocks per grid and 256 threads per block, so that’s 255\*256 = 65280 more threads than cu\_axpy\_2. The version 2 has 2.5 gflps as we’ve already know. Version 3 on average has close to 23.5 gflops. This is not as close to what we’d expect. The expected speedup = expected axpy\_2 time / expected axpy\_3 time = (2\*2^16 / 2.5 gflops) / (2\*2^16 / 23.5 gflops) = 4 / 0.0425 = 94. But in reality speedup = 169ms / 71 ms = 2.38. The following is the axpy\_cuda plot:

Chart, line chart

Description automatically generated

Question 5:

Based on my observations the block size give the best performance when block size = 128, output shown as follow:

[tyao0625@mcu axpy\_cuda]$ ./cu\_axpy\_3.exe 24 128

# elapsed time [cuda\_malloc]: 1240 ms

# elapsed time [cuda\_call]: 54 ms

# gflops / sec [ madd ]: 31.0689

# elapsed time [cuda\_free]: 6 ms

Which makes a lot of sense, because based on deviceQuery, the warp size of the gpu is 32, and 128 is 4 times more than the warp size. Which mean each core has to deal with 4 threads, 2 float operation, store 3 float values at a time. And from the device query we also know that the memory bus width is 352-bit. To reduce memory traffic, we need to make the 4 threads operations data stored as close to that number as possible. We need to store 3 float values, each float is 32 bits and we have 4 threads per core, which is 3\*4\*32 = 384 in total. So this way we maximized the usage of each core while also reduce to the least amount of memory traffic or make use the most amount of memory usage.

Question 6:

Because the latter case GPU has a lot more cores and bigger shared memories than CPU. The data being transferred from host to the device is much bigger than the former CPU case, which reduced the memory traffic memory significantly. Each SM contained multiple cores which they will all retrieve entries from the shared memory on the device. When the memory traffic gets resolved, which was the biggest issue that caused stride solution being slow, the speed is no longer affected by and therefore significantly increased.

Question 7:

The max number of Gflops I was able to achieve from the GPU was around 55 Gflops. The following are the outputs from the seq and omp executable:

[tyao0625@mcu norm\_cuda]$ ./norm\_seq.exe

N Sequential 1 thread 2 threads 4 threads 8 threads

1048576 12.7864 12.7178 12.752 12.752 12.752

2097152 10.6644 11.2178 11.3246 11.2443 11.2709

4194304 8.50802 8.56901 8.44789 8.61533 8.59983

8388608 8.20346 8.13441 8.25955 8.24546 8.23141

16777216 8.11267 8.17994 8.24833 8.16639 8.22084

33554432 8.213 8.09567 8.17352 8.1998 8.09567

[tyao0625@mcu norm\_cuda]$ ./norm\_parfor.exe

N Sequential 1 thread 2 threads 4 threads 8 threads

1048576 12.0706 12.1014 23.838 47.4376 87.8474

2097152 11.0356 11.087 22.4356 39.3086 35.4951

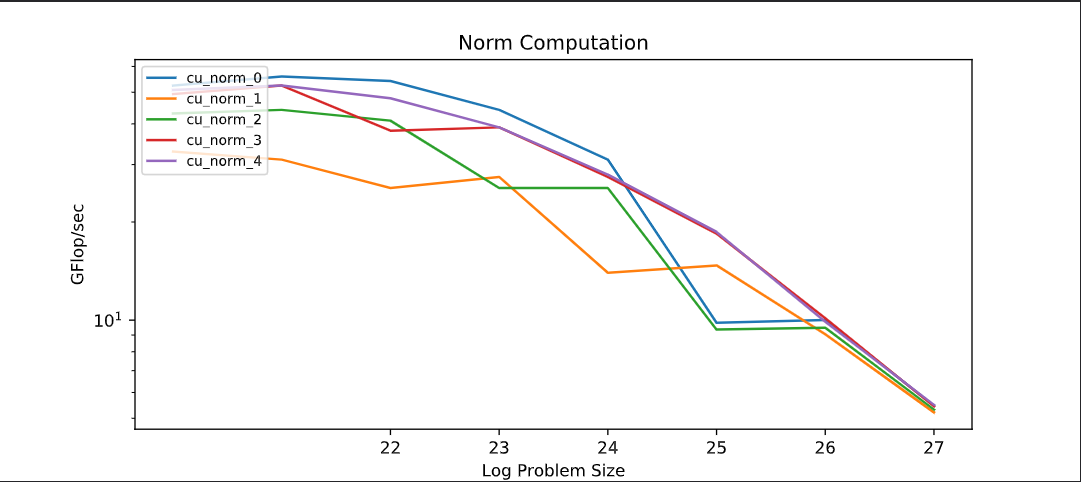
4194304 8.47785 8.67787 10.4628 13.3562 9.48712

8388608 8.21741 8.10711 9.70249 10.4134 8.98111

16777216 8.09935 8.22084 9.44924 9.80617 8.16639

33554432 8.1998 8.1998 9.32408 9.80822 9.18968

The cuda performance is much faster with bigger Gflops across implementations. Especially when the problem sizes are bigger. The following is my norm\_cuda.pdf plot:



Question 8:

The following are the outputs from my norm\_thrust.exe

Float

N Sequential First Second First Second

1048576 2.05736 12.1941 12.2687 1 1

2097152 2.05641 21.3954 21.5098 1 1

4194304 1.98201 33.9774 33.9774 1 1

8388608 1.97474 48.1605 48.1605 1 1

16777216 1.97286 60.787 61.6809 1 1

33554432 1.9729 70.3561 70.3561 1 1

67108864 1.97213 75.7681 77.0102 1 1

134217728 1.97162 80.13 80.13 nan nan

Double

N Sequential First Second First Second

1048576 2.04582 10.5298 10.6134 1 1

2097152 1.89376 16.9719 16.9719 1 1

4194304 1.86499 24.0673 24.0673 1 1

8388608 1.85484 30.3233 30.5496 1 1

16777216 1.85097 35.2463 35.2463 1 1

33554432 1.85148 38.2638 38.2638 1 1

67108864 1.85019 39.8103 39.8103 1 1

134217728 1.84937 40.672 40.9825 1 1

The highest performance I achieved was 80 Gflops for Float and 41 Gflops for double. When problem sizes are small, the performances of float and double are very similar. With float being just a little higher than double. But I started to see a more drastic performance when the problem sizes are bigger. In the biggest problem size, the float’s Gflops almost doubled comparing to double. The following is a listing of my norm\_thrust:

template<typename T>

T norm\_thrust(const thrust::device\_vector<T>& x) {

  T sum = thrust::reduce(x.begin(), x.end(), 0);

  return std::sqrt(sum);

}