Rohan Tiwari
AMATH 583
PS7

1. Hello_omp

1.1 How many omp threads are reported as being available? Try increasing the number of cpus-per-task. Do you always get a corresponding number of omp threads? Is there a limit to how many omp threads you can request?

2 omp threads are reported as available.

It does not allow allocating more than 40 cpu tasks and gives the following error

"srun: error: Unable to allocate resources: Job violates accounting/QOS policy (job submit limit, user's size and/or time limits)"

The number of omp threads is bounded by the hardware concurrency which is 40.

```
[rtiwari6@klone1 hello_omp]$ srun --time 5:00 -A amath -p gpu-rtx6k --cpus-per-task 40 ./ompi_info.exe
srun: job 4574408 queued and waiting for resources
srun: job 4574408 has been allocated resources
OMP_NUM_THREADS      =
hardware_concurrency() = 40
omp_get_max_threads()  = 40
omp_get_num_threads()  = 1

[rtiwari6@klone1 hello_omp]$ srun --time 5:00 -A amath -p gpu-rtx6k --cpus-per-task 41 ./ompi_info.exe
srun: error: QOSGrpCpuLimit
srun: error: Unable to allocate resources: Job violates accounting/QOS policy (job submit limit, user's size and/or time limits)
```

1.2 What is the reported hardware concurrency and available omp threads if you execute ompi_info.exe on the login node?

We get 96 hardware concurrency and 96 max omp threads

```
./ompi_info.exe
OMP_NUM_THREADS      =
hardware_concurrency() = 96
omp_get_max_threads()  = 96
omp_get_num_threads()  = 1
```

2. norm

2.1 What are the max Gflop/s reported when you run norm_parfor.exe with 8 cores? How much speedup is that over 1 core? How does that compare to what you had achieved with your laptop?

Rohan Tiwari
AMATH 583
PS7

Max Gflops/s reported for 8 cores is 11.2425 GFlops/s. Based on problem size, the speed up can be anywhere between 3x-7x for 8 cores compared to 1 core. For 8 cores, speed up is between 3x-7x for 8 threads versus 1 thread. On my laptop, the speed up was around 2x-3x for 8 threads versus 1 thread which is much lower.

The max GFlops/s reported on laptop for parfor was 6.20731 GFlops/s which is much lower than was seen for 8 cores here.

```
[rtiwari6@klone1 norm]$ srun --time 5:00 -A amath  -p gpu-rtx6k norm_parfor.exe

      N  Sequential   1 thread  2 threads  4 threads  8 threads    1 thread    2 threads    4 threads    8 threads

1048576    1.85875    1.86047     1.8468    1.83501    1.79729           0  3.54027e-14  2.52052e-14  2.53976e-14

2097152   0.232615    1.86499    1.86156    1.84964    1.82625    0.646447    0.646447    0.646447    0.646447

4194304   0.215887    1.75995    1.75392    1.74643    1.73754    0.646447    0.646447    0.646447    0.646447

8388608   0.215358     1.6526    1.66309    1.66045    1.66971    0.646447    0.646447    0.646447    0.646447

16777216  0.214585      1.716    1.71331    1.71331    1.69863    0.646447    0.646447    0.646447    0.646447

33554432  0.214444    1.71822    1.71321    1.70947    1.71071    0.646447    0.646447    0.646447    0.646447

[rtiwari6@klone1 norm]$ srun --time 5:00 -A amath  -p gpu-rtx6k --cpus-per-task 8  ./norm_parfor.exe

      N  Sequential   1 thread  2 threads  4 threads  8 threads    1 thread    2 threads    4 threads    8 threads

1048576    1.43963    1.80536    3.54703    6.98322    4.59171    0.646447    0.646447    0.646447    0.646447

2097152    1.64764    1.79066    3.59086    6.87638    5.61571    0.646447    0.646447    0.646447    0.646447

4194304     1.6414    1.62704    3.16356    6.22134    3.95139    0.646447    0.646447    0.646447    0.646447

8388608    1.32563     1.6333    3.23635      6.223    10.6454    0.646447    0.646447    0.646447    0.646447

16777216   1.29285    1.56123    3.14271    6.02497    11.2425    0.646447    0.646447    0.646447    0.646447

33554432   1.32701    1.56796    3.10689    5.36258    6.29708    0.646447    0.646447    0.646447    0.646447
```

3. matvec

3.1 What are the max Gflop/s reported when you run pmatvec.exe with 16 cores? How does that compare to what you had achieved with your laptop?

We see considerable speed up for CSR and CSC^T when going from 1 thread to 16 threads as these were the versions optimized using openmp. The speed up is anywhere between 2x-12x depending on problem sizes for CSR and 3x-12x for CSC^T. On the laptop this speed up was between 2x-3x for both CSR and CSC^T.

Max GFlops/s reported with 16 cores is 25.3158 GFlops/s which is for CSR. Max GFlops/s achieved on laptop was 5.71819 GFlops/s. The hyak version is about 5x faster.

Oversubscribing does not help with the performance. It actually made the performance slower.

```
[rtiwari6@klone1 matvec]$ srun -A amath -p gpu-rtx6k --time 5:00 --cpus-per-task 16 ./pmatvec.exe 2048 16

1 threads

N(Grid) N(Matrix)       NNZ       COO     COO^T       CSR     CSR^T       CSC     CSC^T
```

64	4096	20224	1.81942	2.08476	2.1993	1.96212	1.98155	2.1993
128	16384	81408	1.65776	1.75956	2.11147	1.94747	1.96656	2.13393
256	65536	326656	1.66006	1.731	2.17771	1.98556	1.89277	2.20138
512	262144	1308672	1.23169	1.29252	1.82076	1.64872	1.63584	1.88637
1024	1048576	5238784	0.97171	0.983608	1.2617	1.21098	1.21709	1.27505
2048	4194304	20963328	0.913933	0.91643	1.1566	1.089	1.08198	1.14868

2 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.02158	2.41129	4.54856	2.22374	2.35455	4.76516
128	16384	81408	1.67158	1.79098	4.26786	1.92874	1.91037	4.17894
256	65536	326656	1.56997	1.68772	4.05053	1.84115	1.87525	4.1332
512	262144	1308672	1.24635	1.32524	3.22135	1.52838	1.55102	3.48979
1024	1048576	5238784	0.956286	0.97171	2.36259	1.20492	1.20492	2.36259
2048	4194304	20963328	0.913933	0.91894	2.23609	1.09612	1.08548	2.23609

4 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.00137	2.41129	7.69757	2.27428	2.38258	7.69757
128	16384	81408	1.67158	1.79098	8.02357	1.94747	1.91037	8.02357
256	65536	326656	1.63328	1.74592	7.78949	1.89277	1.87525	7.78949
512	262144	1308672	1.22449	1.29252	6.54336	1.57434	1.57434	6.75444
1024	1048576	5238784	0.960096	0.97961	4.46267	1.20492	1.21709	4.46267
2048	4194304	20963328	0.91894	0.921465	4.30017	1.09612	1.08548	4.30017

8 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.02158	2.44069	10.0068	2.24873	2.32717	10.0068
128	16384	81408	1.65776	1.77513	13.3726	1.91037	1.89235	13.3726
256	65536	326656	1.64656	1.74592	15.579	1.91063	1.87525	15.579
512	262144	1308672	1.22449	1.27675	11.6326	1.49563	1.47456	12.3169
1024	1048576	5238784	0.948756	0.963936	8.60657	1.20492	1.21098	8.3098
2048	4194304	20963328	0.91643	0.921465	8.18081	1.09612	1.089	8.18081

16 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.02158	2.41129	3.57387	2.27428	2.41129	6.45602
128	16384	81408	1.63081	1.77513	15.4299	1.91037	1.91037	15.4299
256	65536	326656	1.62021	1.731	25.3158	1.89277	1.85804	25.3158

512 262144 1308672 1.1965 1.26137 23.2653 1.44405 1.43416 26.1734

1024 1048576 5238784 0.956286 0.967807 14.1755 1.09043 1.19893 14.1755

2048 4194304 20963328 0.91643 0.91894 9.31703 1.06143 1.05809 9.06522

[rtiwari6@klone1 matvec]$ srun -A amath -p gpu-rtx6k --time 5:00 --cpus-per-task 16 ./pmatvec.exe 2048 32

1 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	1.78693	2.1067	2.1993	1.92439	1.92439	2.1993
128	16384	81408	1.64417	1.72922	2.11147	1.94747	1.94747	2.13393
256	65536	326656	1.64656	1.71633	2.15454	1.96628	1.96628	2.17771
512	262144	1308672	1.16326	1.21033	1.73048	1.57434	1.64872	1.86953
1024	1048576	5238784	0.975644	0.991704	1.26834	1.21709	1.19893	1.25513
2048	4194304	20963328	0.91643	0.921465	1.1566	1.089	1.08198	1.15262

2 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.02158	2.41129	4.65434	2.27428	2.32717	4.65434
128	16384	81408	1.65776	1.77513	4.17894	1.91037	1.91037	4.17894
256	65536	326656	1.63328	1.71633	4.05053	1.85804	1.85804	4.1332
512	262144	1308672	1.22449	1.30054	3.48979	1.57434	1.47456	3.32361
1024	1048576	5238784	0.952506	0.97171	2.33965	1.21098	1.20492	2.38598
2048	4194304	20963328	0.91894	0.924003	2.2511	1.09612	1.08198	2.22128

4 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.00137	2.41129	7.41247	2.27428	2.35455	8.00547
128	16384	81408	1.65776	1.77513	8.02357	1.87467	1.89235	8.02357
256	65536	326656	1.62021	1.70191	7.78949	1.87525	1.87525	7.78949
512	262144	1308672	1.1965	1.26902	5.81632	1.50639	1.47456	5.9825
1024	1048576	5238784	0.956286	0.975644	4.38153	1.19893	1.20492	4.38153
2048	4194304	20963328	0.91894	0.921465	4.30017	1.09612	1.08198	4.24574

8 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.00137	2.41129	10.0068	2.27428	2.30042	10.5335
128	16384	81408	1.65776	1.77513	14.3278	1.89235	1.89235	14.3278
256	65536	326656	1.63328	1.71633	15.579	1.87525	1.89277	15.579
512	262144	1308672	1.21033	1.26137	13.0867	1.57434	1.49563	13.0867
1024	1048576	5238784	0.956286	0.975644	8.3098	1.20492	1.20492	8.3098

2048	4194304	20963328	0.913933	0.91894	8.18081	1.089	1.08198	7.98603

16 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	2.02158	2.44069	1.74032	2.24873	2.35455	5.55935
128	16384	81408	1.64417	1.75956	14.3278	1.89235	1.91037	14.3278
256	65536	326656	1.5947	1.68772	25.3158	1.84115	1.84115	25.3158
512	262144	1308672	1.16326	1.23169	23.2653	1.52838	1.53961	26.1734
1024	1048576	5238784	0.948756	0.963936	10.9538	1.12086	1.19299	10.9538
2048	4194304	20963328	0.906522	0.913933	7.98603	1.04166	1.0785	7.98603

32 threads

N(Grid)	N(Matrix)	NNZ	COO	COO^T	CSR	CSR^T	CSC	CSC^T
64	4096	20224	1.94307	2.41129	0.491736	2.1993	2.32717	0.485769
128	16384	81408	1.60471	1.71444	1.74425	1.92874	1.94747	1.71444
256	65536	326656	1.5947	1.71633	4.82206	1.85804	1.84115	5.06317
512	262144	1308672	1.21033	1.27675	9.51761	1.39592	1.48502	9.10381
1024	1048576	5238784	0.937681	0.956286	7.30255	1.09538	1.18711	6.0246
2048	4194304	20963328	0.906522	0.91643	6.84517	1.0648	1.09612	7.29159

4. Pagerank

4.1 How much speedup (ratio of elapsed time for pagerank comparing 1 core with 8 cores) do you get when running on 8 cores?

If we compare the performance of 1 thread / 1 core to 8 threads /8 cores we see a speed up of around 2.54x

[rtiwari6@klone1 pagerank]$ srun -A amath -p gpu-rtx6k --time 5:00 --cpus-per-task 1 ./pagerank.exe -n 1 /gscratch/amath/amath583/data/as-Skitter.mtx

elapsed time [read]: 8695 ms

Converged in 46 iterations

elapsed time [pagerank]: 3558 ms

elapsed time [rank]: 134 ms

[rtiwari6@klone1 pagerank]$ srun -A amath -p gpu-rtx6k --time 5:00 --cpus-per-task 8 ./pagerank.exe -n 8 /gscratch/amath/amath583/data/as-Skitter.mtx

srun: job 4654112 queued and waiting for resources

srun: job 4654112 has been allocated resources

elapsed time [read]: 6821 ms

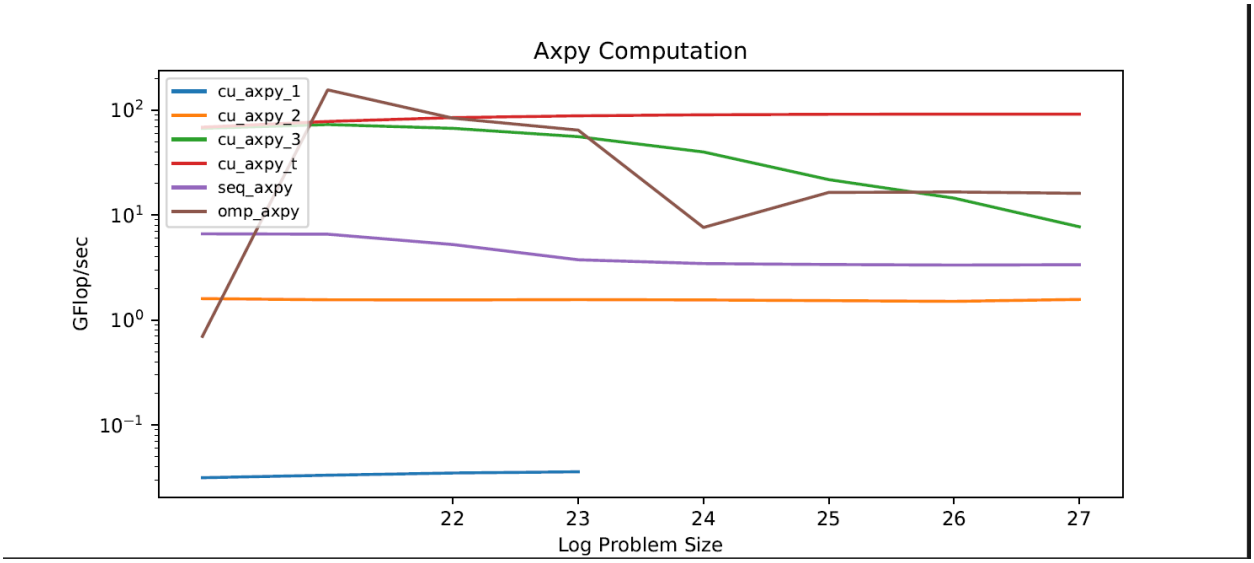
Rohan Tiwari
AMATH 583
PS7

Converged in 46 iterations

elapsed time [pagerank]: 1399 ms

elapsed time [rank]: 130 ms

5. AXPY CUDA



5.1 How many more threads are run in version 2 compared to version 1? How much speedup might you expect as a result? How much speedup do you see in your plot?

Version 1 uses 1 thread and Version 2 uses 256 threads, so additional threads is 256-1=255 in Version 2. This was figured out from the following lines of code:

Version 1:

```
    madd<<<1, 1>>>(N, 3.14159, x, y);
```

Version 2:

```
    madd<<<1, 256>>>(N, 3.14159, x, y);
```

We would expect a speedup of 256x in version 2 but we see much less from the plot. The plot shows roughly about 50x speedup.

5.2 How many more threads are run in version 3 compared to version 2? How much speedup might you expect as a result? How much speedup do you see in your plot? (Hint: Is the speedup a function of the number of threads launched or the number of available cores, or both?)

Version 3 uses blocks. Block_size is 256 which means each block has 256 threads

num_blocks = (N + block_size - 1) / block_size.

Default N=2^16. We see num_blocks = (2^16 + 256 -1) / 256 which is roughly around 2^8. We have 2^8 threads per block and 256 blocks which means we would expect a speed up of 2^16.

In the plot, we see about 2200x for small problem sizes e.g. N=20 and it reduces for larger ones e.g 1864x for N=23. I think the number of cores available does limit the final speed up obtained even if we are able to launch a lot of threads.

5.3 (AMATH 583) The cu_axpy_3 also accepts as a second command line argument the size of the blocks to be used. Experiment with different block sizes with, a few different problem sizes (around 2^24 plus or minus). What block size seems to give the best performance? Are there any aspects of the GPU as reported in deviceQuery that might point to why this would make sense?

For best performance, the occupancy should be high. For this the number of active warps must be close to the max warps. This ensures that any instruction fetch latency is hidden away. In the table below, the datapoints for various runs are captured. The highlighted ones are the max GFlops/s for that problem size.

Block Size	Problem Size (N)	GFlops/s
32	24	32.89
64	24	37.28
256	24	39.01
128	24	39.01
512	24	39.01
1024	24	39.94
32	20	52.48
64	20	67.10
256	20	69.90
128	20	69.90
512	20	69.90
1024	20	67.10
32	28	6.27
64	28	6.22
256	28	6.61
128	28	6.57
512	28	6.71
1024	28	7.10
32	16	16.28
64	16	17.84
256	16	19.06
128	16	18.85
512	16	18.85
1024	16	18.64

Looking at the numbers above block size of 512 generally does well. It comes out to be the 1st or 2nd best performing for different problem sizes that were tried. From the DeviceQuery results below, max

number of threads per block and per multiprocessor is 1024. Max warp size possible is 32. It is a good idea to keep block size a multiple of 32 as threads in a warp are executed together. Since each multiprocessor supports 1024 threads, with a block size of 512, we can end up with max 2 active blocks on the processor.

Output of devicequery:

```
[rtiwari6@klone1 deviceQuery]$ srun -p gpu-rtx6k  -A amath --gres=gpu:rtx6k ./deviceQuery
/mmfs1/home/rtiwari6/ps7/samples/1_Utilities/deviceQuery/./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Quadro RTX 6000"
  CUDA Driver Version / Runtime Version          11.6 / 11.6
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 24220 MBytes (25396969472 bytes)
  (72) Multiprocessors, ( 64) CUDA Cores/MP:     4608 CUDA Cores
  GPU Max Clock rate:                            1770 MHz (1.77 GHz)
  Memory Clock rate:                             7001 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 6291456 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
```

```
 Texture alignment:                  512 bytes
 Concurrent copy and kernel execution:        Yes with 3 copy engine(s)
 Run time limit on kernels:                   No
 Integrated GPU sharing Host Memory:          No
 Support host page-locked memory mapping:     Yes
 Alignment requirement for Surfaces:          Yes
 Device has ECC support:                      Disabled
 Device supports Unified Addressing (UVA):    Yes
 Device supports Managed Memory:              Yes
 Device supports Compute Preemption:          Yes
 Supports Cooperative Kernel Launch:          Yes
 Supports MultiDevice Co-op Kernel Launch:    Yes
 Device PCI Domain ID / Bus ID / location ID:   0 / 35 / 0
 Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >


deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.6, CUDA Runtime Version = 11.6, NumDevs = 1
Result = PASS
```

6. Nvprof

6.1 Looking at some of the metrics reported by nvprof, how do metrics such as occupancy and efficiency compare to the ratio of threads launched between versions 1, 2, and 3?

Dr. Liu said on piazza “You can ignore question 9 in this assignment. nvprof has been disabled on RTX 6000 for event/metric collection.”

7. Striding

7.1 Think about how we do strided partitioning for task-based parallelism (e.g., OpenMP or C++ tasks) with strided partitioning for GPU. Why is it bad in the former case but good (if it is) in the latter case?

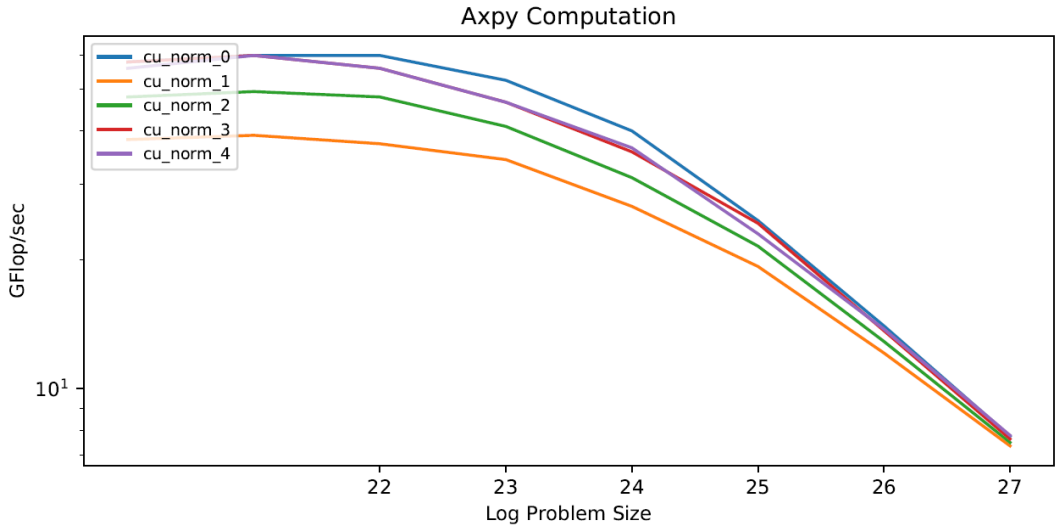
It is bad in OpenMP or C++ tasks case because threads were not accessing contiguous memory. We used strides to assign threads to data but the data is contiguous and this made the number of cache misses increase.

In case of GPU, strided partitioning works differently. This a way to assign work to threads due to the 3d structure they are stored in. From the code, strides are equal to the length of block dimensions. Threads are organized into blocks and each block is executed on a processing unit (I think each unit can execute multiple blocks). Inside each block, threads are organized into warps which are scheduled for execution

together. Since strides are equal to block dimensions, within the warp the data access becomes contiguous. Warps essentially work as SIMD – threads inside the warp access contiguous data in parallel and the thread Ids are consecutive inside the wrap. So, within each warp we can get contiguous access which leads to better performance than the former case.

8. Norm_cuda

8.1 What is the max number of Gflop/s that you were able to achieve from the GPU? Overall?



Highest GFlops/s observed is 59.9186 for the GPU. This is much faster than what is observed for CPU implementations.

59.9186 is observed for cu_norm_0 for N=21,22 and for cu_norm_3 for N=21.

9. Retrospective

9.1 The most important thing I learned from this assignment was..

Programming in Cuda and it's programming model.

9.2 One thing I am still not clear on is..

What it takes to determine the correct block size