

GPU — Architecture & Programming

Assignment 1

Anirudhan Rajagopalan — ajr619@nyu.edu

November 17, 2016

1 List of optimizations

1.1 First GPU version

Implemented with a kernel that takes the array and a prime number as arguments. The kernel calculates its id and checks if its id is a perfect multiple of the prime number and also checks if it is not equal to the prime number. If this condition matches, it sets the array to 1 and exits.

Time for 10M: Too long to wait (more than 30 minutes). Killed the process.

1.2 Second GPU version

This version takes multiple prime numbers as inputs and does the same processing.

1. When 8 primes are considered; Time = 5m, 20s
2. When 128 primes are considered; Time = 1m, 32s
3. When 1024 primes are considered; Time = 41.375 seconds
4. When 2048 primes are considered; Time = 40 seconds

While doing this testing, I also checked the code for branch divergence and other events that happens in the GPU. From the summary output of `nvcc`, we can find that the majority of the time is spent in *cudaMemcpy*. I tried to remove this by creating a `memcpy` kernel that again creates multiple kernels for prime number processing. But this failed as I cannot compile this for the current cuda device I have.

1.3 Reducing Branch Divergence

1. I tried reducing branch divergence by combining all exit cases into a single case. This helped improved the performance by around 4 seconds.
2. I also added conditions to check if a thread is processing a number which is already a non-prime. This didn't improve the performance much.
3. I added change to exit out of the for loop as soon as the number is found to be non-prime. This helped in saving around 3 seconds.

2 Speed of GPU vs CPU

3 Speedup for various values of N

4 nvprof analysis

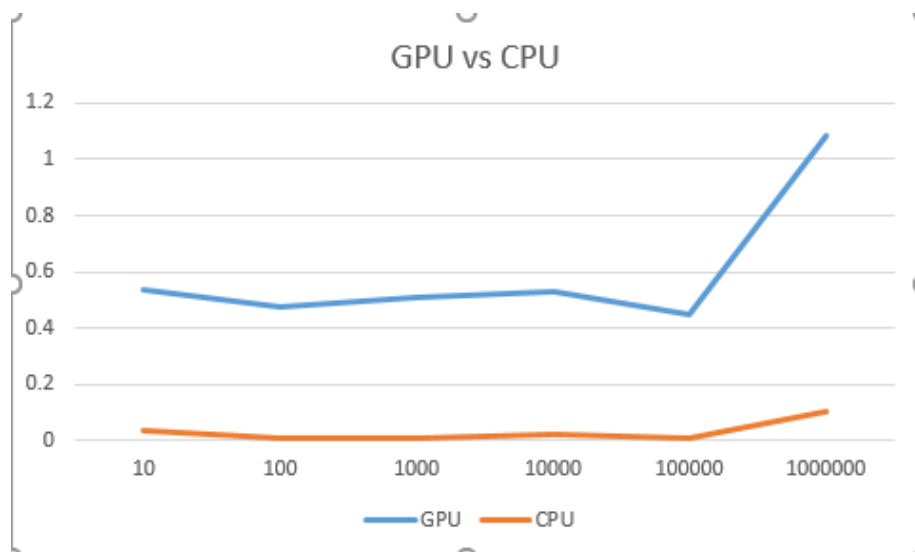


Figure 1: GPU and CPU performance.

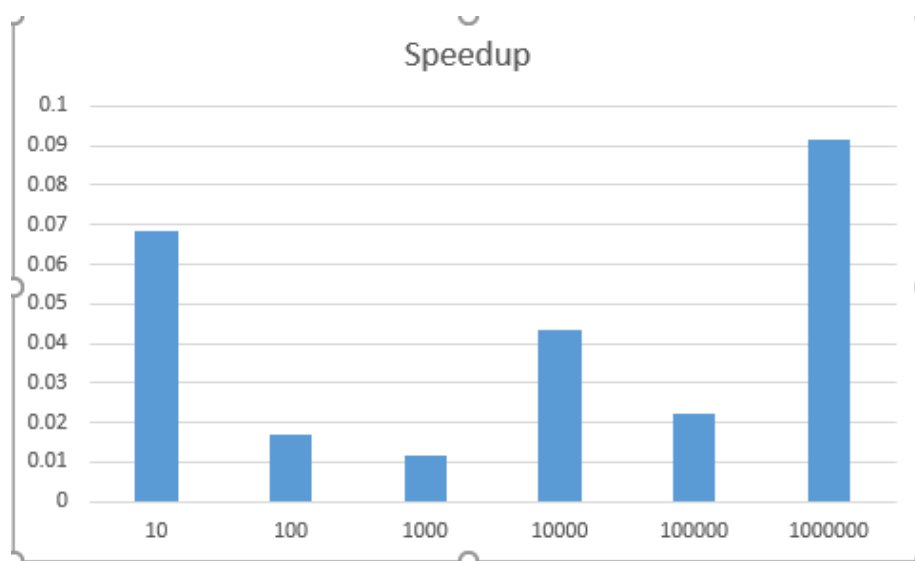


Figure 2: Speedup.

```

ajr619@cuda2[lab2]$ nvprof --profile-from-start-off ./genprimes 1000000
==129715== NVPROF is profiling process 129715, command: ./genprimes 1000000
Count is 78498
==129715== Profiling application: ./genprimes 1000000
==129715== Profiling result:
Time(%)      Time      Calls      Avg      Min      Max      Name
97.81%    548.34ms        22    24.925ms    16.039ms    35.670ms    do_sieve(unsigned int*, unsigned int*, unsigned int, int)
2.08%     11.688ms        23     508.18us    443.36us    644.33us    [CUDA memcpy DtoH]
0.10%      575.72us        23     25.031us    1.6000us    538.85us    [CUDA memcpy HtoD]
==129715== API calls:
Time(%)      Time      Calls      Avg      Min      Max      Name
99.46%    565.80ms        46    12.300ms    19.961us    36.431ms    cudaMemcpy
0.45%     2.5326ms        22     115.12us    24.463us    1.8521ms    cudaLaunch
0.08%      467.59us         2     233.79us    182.53us    285.06us    cudaMalloc
0.01%      37.168us        88         422ns    162ns     3.2860us    cudaSetupArgument
0.00%      27.976us        22     1.2710us     906ns    1.7470us    cudaConfigureCall
ajr619@cuda2[lab2]$

```

Figure 3: Basic nvprof profiling.

```

ajr619@cuda2[lab2]$ nvprof --profile-from-start-off --events branch,divergent_branches ./genprimes 1000000
===== Warning: Event "branch" cannot be found on device 1.
===== Warning: Event "branch" cannot be found on device 2.
===== Warning: Event "divergent_branches" cannot be found on device 0.
===== Warning: Event "divergent_branches" cannot be found on device 1.
===== Warning: Event "divergent_branches" cannot be found on device 2.
===== Warning: Event "divergent_branches" cannot be found on device 3.
==129586== NVPROF is profiling process 129586, command: ./genprimes 1000000
Count is 664579
==129586== Profiling application: ./genprimes 1000000
==129586== Profiling result:
==129586== Event result:
Invocations
Device "GeForce GTX TITAN X (0)"
Kernel: do_sieve(unsigned int*, unsigned int*, unsigned int, int)
172      branch    1460038820    3070073136    2231536279    3.8382e+11
ajr619@cuda2[lab2]$

```

Figure 4: Branch Divergence.