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 *cudamemcopy*. I tried to remove this by creating a ¡¡¡1,1¿¿¿ 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

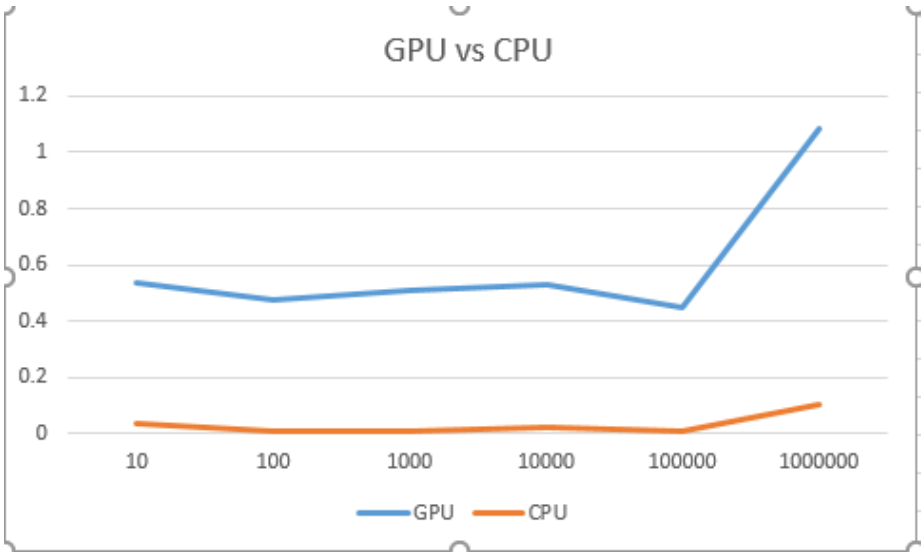

Figure 1: GPU and CPU performance.

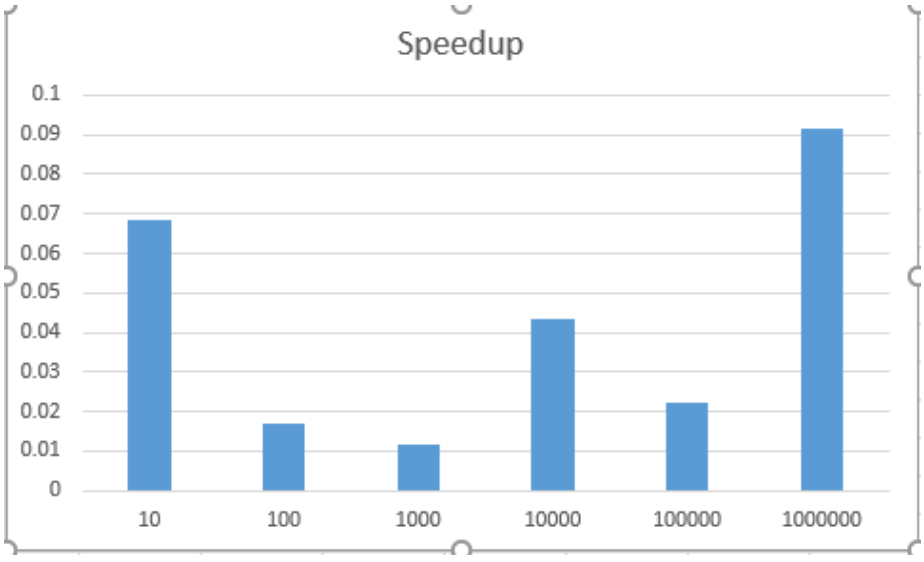

Figure 2: Speedup.

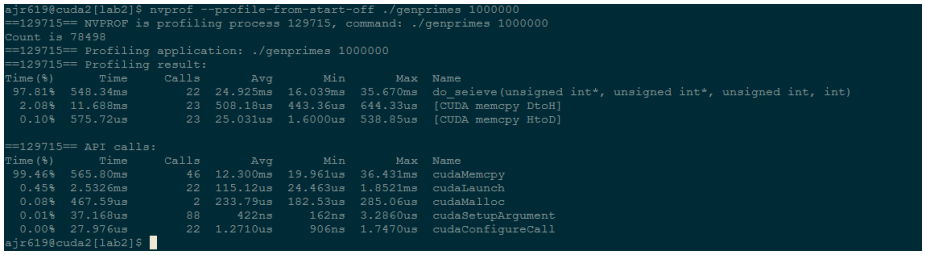

Figure 3: Basic nvprof profiling.

Figure 4: Branch Divergence.