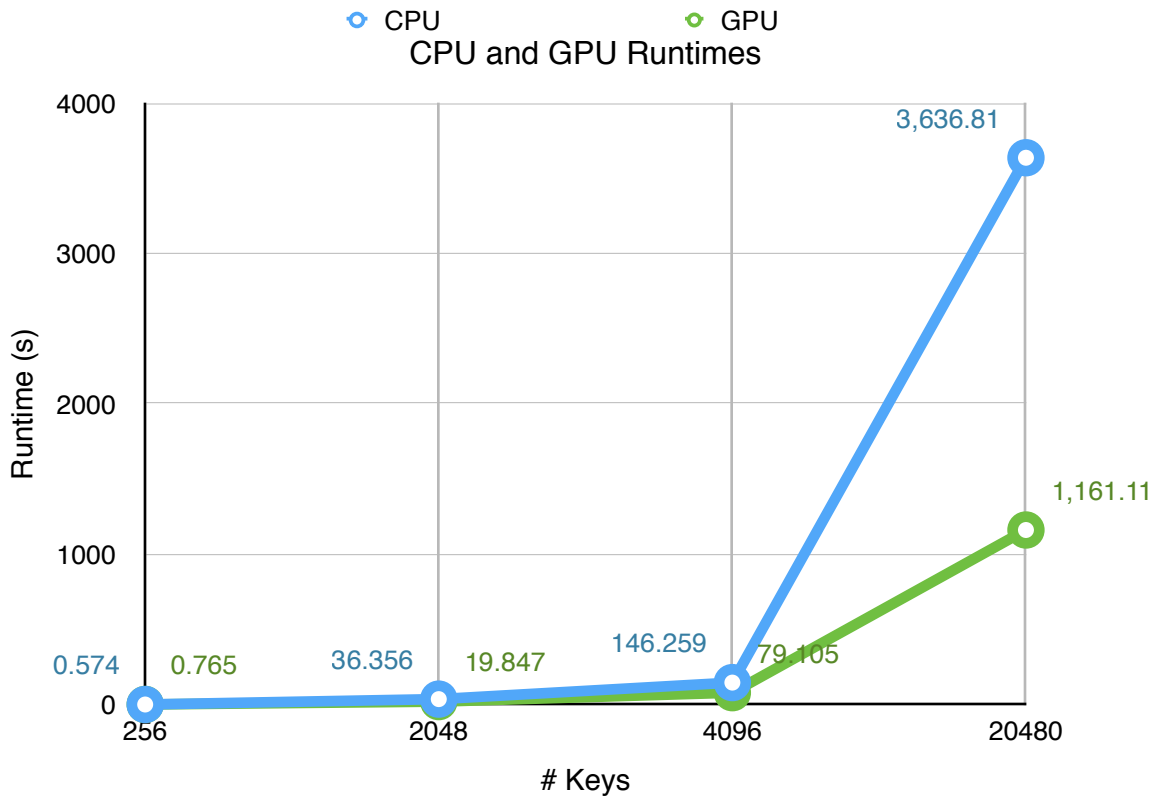
	256	2048	4096	20480
CPU	0.574	36.356	146.259	3636.81
GPU	0.765	19.847	79.105	1161.11



Mitchell Rosen, Christopher Wu

==23149== NVPROF is profiling process 23149, command: ./main ../256-keys.txt 256 cracked
==23149== Profiling application: ./main ../256-keys.txt 256 cracked
==23149== Profiling result:
Time(%) Time Calls Avg Min Max Name
99.99% 318.66ms 1 318.66ms 318.66ms 318.66ms cuda_factorKeys(integer const *, unsigned short*, int, int, int, int)
0.00% 6.9450us 1 6.9450us 6.9450us 6.9450us [CUDA memcpy DtoH]
0.00% 6.4640us 1 6.4640us 6.4640us 6.4640us [CUDA memcpy HtoD]
0.00% 2.8160us 1 2.8160us 2.8160us 2.8160us [CUDA memset]

==23137== NVPROF is profiling process 23137, command: ./main ../2048-keys.txt 2048 cracked
==23137== Profiling application: ./main ../2048-keys.txt 2048 cracked
==23137== Profiling result:
Time(%) Time Calls Avg Min Max Name

```
100.00%  19.7552s        10  1.97552s  1.23738s  2.46753s  cuda_factorKeys(integer const *,
unsigned short*, int, int, int, int)
  0.00%  67.424us        10  6.7420us  6.6240us  7.2000us  [CUDA memcpy DtoH]
  0.00%  45.856us         1  45.856us  45.856us  45.856us  [CUDA memcpy HtoD]
  0.00%  20.641us        10  2.0640us  1.8560us  3.6480us  [CUDA memset]

==23086== NVPROF is profiling process 23086, command: ./main ../4096-keys.txt 4096
cracked
==23086== Profiling application: ./main ../4096-keys.txt 4096 cracked
==23086== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
100.00%  79.0264s        36  2.19518s  1.23505s  2.47253s  cuda_factorKeys(integer const *,
unsigned short*, int, int, int, int)
  0.00%  241.41us        36  6.7050us  6.5920us  7.2640us  [CUDA memcpy DtoH]
  0.00%  89.377us         1  89.377us  89.377us  89.377us  [CUDA memcpy HtoD]
  0.00%  69.442us        36  1.9280us  1.8560us  3.6480us  [CUDA memset]
```

Our CUDA implementation was noticeably faster than our CPU(GMP) implementation starting with 2048 keys. This is probably because at the smaller key sets, the overhead from GPU functions has more impact. The 256 key set only used 1 kernel. As the key sets become larger, the memory overhead has less impact because the gcd calculations account for virtually the entire runtime, so the performance benefit from parallel computations takes precedent.

Our CUDA implementation worked well because we allocated 1 warp for each pair comparison. With 32 threads in a warp, 1 thread can handle parallel computation for 1 int32 calculation.

If we were to start over, we experiment with smaller kernel sizes. The CUDA occupancy calculator stated that the Quadro 5000 could only run 3 warps in parallel for each SM, times 11 for 33 warps in parallel. Our tile size was 512, which turns into 128 blocks for the x and y grid dimensions which equals 16384 blocks. Each block has 16 warps. This is way more than the effective occupancy. In about 5% of our runs the kernel timed out probably because there were just too many blocks. We would also try to use less shared memory.