ECE 459: Programming for Performance
Assignment 3

Zhidong Zhang

March 10, 2020

Part 1: Crack it to me

1. Design Choices:

   - If the secret length is x and the number of possible characters is 36, then the number of possible secrets is 36^x. For the secret length of x, the problem size is 36^x, which can be set as the global work size of the kernel. In the kernel, get_global_id(0) returns a global id from 0 to $36^x - 1$. We can create a Base36 decoder to decode the global id into a possible secret with length x. In the same kernel, I verify the correctness of the possible secret by comparing the original signature and the current signature generated by the HMAC_SHA256 function. If the secret is correct, I will write the secret into the output buffer.

   - In the main.cpp, the JWT secret's max length is given as the gMaxSecretLen. Therefore, possible secret lengths are ranging from $[1 : gMaxSecretLen]$. To find the secret, for each possible secret length $x \in [1 : gMaxSecretLen]$, I run the kernel separately with the global size set to 36^x.

2. Results:

	$jwtcracker\_sin$	$jwtcracker\_omp$	$jwtcracker\_ocl$
$MaxSecretLen = 4$	2.593	**0.185s**	$1.424s$
$MaxSecretLen = 5$	$102.06s$	$7.444s$	**3.284s**
$MaxSecretLen = 6$	$4104s$	$306s$	**75s**

3. Speed up explanation

   - Here are the speedups observed from the above table. When the maxSecretLen is 4, jwtcracker_omp is the fastest. This is because jwtcracker_ocl spawning 36^4 number of threads whose overhead outweigh its speedup when the maxSecretLen is small. When the maxSecretLen is 5, jwtcracker_ocl is the fastest, which has 31 times speedup compared with jwtcracker_ocl and 2 times speedup compared with jwtcracker_omp. When the maxSecretLen is 6, jwtcracker_ocl is the fastest, which has 55 times speedup compared with jwtcracker_ocl and 4 times speedup compared with jwtcracker_omp. This is expected. When the secret length is longer, OpenCL shows a significant advantage over OpenMP, because all the threads or work items are distributed and parallelized by the 2560 CUDA Cores in the GPU.

Part 2: Coulomb's Law Problem

1. Design Choices:

   - In the OpenMP code, functions computeForces(), computeApproxPositions(), computeBetterPositions() and isErrorAcceptable() is applied with openMp parallel-for directive, which enables parallization. Therefore, in the OpenCL open, I need to transfer those functions into different kernels functions. I created three kernel functions which are computeForces(), computePositions(), isErrorAcceptable(), whose global work size is set to the total number of particles. In each kernel, get_global_id(0) returns the global id which is the particle id or particle index. Each kernel uses the particle id to compute the property just for that particle.
   - In Simulation::run() function, I substituted all openMP function calls with kernels calls, and created buffers and arguments correctly.

2. Results:

	protons_sin	*protons_omp*	*protons_ocl*
$numProtons = 25, numElections = 25$	**0.025s**	$3.570s$	$1.114s$
$numProtons = 500, numElections = 500$	4.541	$4.809s$	**1.186s**
$numProtons = 1000, numElections = 1000$	$16.347s$	$5.905s$	**1.532s**
$numProtons = 2500, numElections = 2500$	$109.609s$	$17.624s$	**2.075s**
$numProtons = 5000, numElections = 5000$	$441.628s$	$56.321s$	**3.145s**
$numProtons = 7500, numElections = 7500$	$997.164s$	$109.303s$	**4.265s**
$numProtons = 10000, numElections = 10000$	$1998.533s$	$202.066s$	**5.216s**

3. Speed up explanation

   - From the above results, when the number of particles is small, the protons_sin works the best because it has the lowest overhead. However, when the number of particles is bigger (over 500), the protons_ocl shows a significant advantage over protons_omp and protons_sin by achieving more speed up as the number of particles increases. Especially, when the total number of elections is 10000, protons_ocl has around 400 times speedup over the single-thread solution and 40 times speedup over the OpenMP solution. This is as expected. When the number of particles is small, the GPU is under-utilized. As the number of particles increases by x time, the protons_sin's and protons_omp's execution time increase by x^2 time, whereas, the protons_ocl's execution time only increase by x time, because the outer loop of function computeForce() and computerPositions() are fully parallelized on the GPU.