ECE 459: Programming for Performance Assignment 3

Jinming Zhang

March 11, 2020

Note: All the tests are done on ecetesla2

1 Part 1: Crack it to me

This part of the assignment is trying to crack the JWT by using OpenCL. I have implemented my design to parallelized the validation for all the possible combinations of the secret using OpenCL. In my design, I use the OpenCL kernel to validate one possible combination of the secret. The local work group will execute the kernel in parallel to try all the combinations. The key idea is to generate secret using a base alphabet length conversion (If there are 36 alphabets, it will be a base 36 conversion). Because each execution on the kernel has a distinct ”global_id” (ranged from 0 to (the total number of execution - 1)), the base 36 conversion (there use base 36 as an example) will cover most of the possible combinations of the secret. The tricky part is that the base 36 conversion is missing the combinations where it starts with the first alphabet, and its length is smaller than the max secret length. In this case, my design will be padding the secret by inserting the first alphabet and validate the padded secret one by one until it reaches the max secret length. Because of that, my design will try all the secret combinations and crack the JWT. The Average time (using hyperfine with a warm-up run) for cracking JWT with the length of 4,5,6 using different methods is shown below:

JWT Length	Sequential Version (s)	OpenMP Version (s)	OpenCL Version (s)
4	5.288	0.4710	0.641
5	91.385	6.741	2.130
6	4212.175	316.18	44.786

The result comparison for the sequential Version and OpenMP version is done in the previous assignment. Here will focus on the OpenCL version only. The speedup for the JWT Length 4, 5, 6 is 8.24x, 42.9x and 94x, respectively. For length 4, since there are fewer combinations to validate and the overhead for creating local work group is high compared to the overall time, the speedup is not as significant as length 6 speedup.

Part 2: Coulomb’s Law Problem

This part of the assignment is trying to solve Coulomb’s Law problem with OpenCL. For the given OpenMP code, all the directives are used to parallelize the for-loops in the ”computeForces”, "computeApproxPositions", "computeBetterPositions" and "isErrorAcceptable" functions. To substitute OpenMP with OpenCL, we can use the same idea as part one of this assignment that "global_id" is distinct and ranged from 0 to (the total number of execution - 1). Instead parallelize the for-loops, I used the "global_id" as the loop counter and let local work group to parallelize each iteration of the for-loops in those functions. Also, I have strictly follow the operation order (follows the seven steps in the assignment manual) in the OpenMP version code to preserved the behaviour of my design. The speedup can be shown in the average time table (using hyperfine with a warm-up run) for solving Coulomb's Law Problem with the different number of particles using different methods:

Number of Particles	Sequential Version (s)	OpenMP Version (s)	OpenCL Version (s)
50	0.0071	1.894	0.5415
500	1.069	3.088	0.6822
5000	90.374	54.621	1.752

The speedup for OpenMP for 50, 500 and 5000 particles is 0.003x, 0.35x and 1.65x respectively. The speedup for OpenCL for 50, 500 and 5000 particles are 0.013x, 1.57x and 51.58x, respectively. First of all, we can see that when there are fewer particles (i.e. 50) for the problem, using OpenMP and OpenCL will result in a slow down when compared to the sequential version. This is because the overhead of threads in parallelization is far too high (when compared to overall run-time) when running the code on the given sample test cases, as the test cases have at most 50 particles. However, when there are more and more particles presented, OpenCL will outperform OpenMP and the sequential version by a lot. This is because OpenCL uses GPU, which has way more cores than the CPU. The parallelization that GPU offers will out weight the overhead of threads/work group creation and dominate the performance, as shown in the above table.