ECE 459: Programming for Performance
Assignment 3

David Chau (20623345)

March 10, 2019

Part 1: Crack it to me

The task in this section was to parallelize the `jwtcracker` code from Assignment 2 using OpenCL instead of OpenMP. This was done by first taking the recursive-based code from Assignment 2, converting it to use an iterative-based approach, and finally implementing it in the `bruteForceJWT` kernel. This code allows all combinations of a given set of characters to be computed and checked to see whether it is the correct secret or not. Next, a parallelization strategy was decided upon. Specifically, `globalSize` was set to be a 3D range with equal dimensions of `gAlphabet.length()`. This enables each work item processed by `bruteForceJWT` to be allocated 3 of `gMaxSecretLen` characters as a starting secret, effectively reducing the amount of work a single thread has to perform and distributing it among many threads.

The results of testing using `hyperfine` are shown below. It can be seen that with this approach, performance has decreased across the three given test cases. This may indicate that the overhead of parallelizing using OpenCL is more costly than that of OpenMP. However, it should be noted that the results for OpenCL were computed during a period of time where server load was high, so the values shown here may not be indicative of the true performance. Regardless, OpenCL proves to be a viable method for parallelization and also shows potential to surpass the OpenMP implementation in terms of speed given the correct optimizations.

Table 1: Results for different parallelized `jwtcracker` executions on `ecetesla0`

Description	**Mean (s)**	σ **(s)**	**Min (s)**	**Max (s)**
OpenMP, `JWT_TOKEN_4`	0.2269	0.0781	0.1558	0.3759
OpenMP, `JWT_TOKEN_5`	4.574	1.098	2.705	5.589
OpenMP, `JWT_TOKEN_6`	257.061	63.625	143.784	348.999
OpenCL, `JWT_TOKEN_4`	0.700	0.004	0.6942	0.7063
OpenCL, `JWT_TOKEN_5`	9.143	0.331	8.980	10.040
OpenCL, `JWT_TOKEN_6`	377.647	73.612	276.335	463.215

Part 2: Coulomb's Law Problem

The task in this section was to parallelize, using OpenCL, a given program that runs a simulation of Coulomb's Law. Essentially, converting CPU code into GPU code. Following the CPU code, this was done by creating the following three kernels:

- `computeForces`: used to calculate `k0` or `k1` for a given particle
- `computeApproxPositions`: used to calculate `y1` for a given particle
- `computeBetterPositionsAndCheckError`: used to calculate `z1` for a given particle and check if the error for that particle is within the acceptable range

In this case, each kernel must be enqueued and run on all particles before the next kernel can begin. The main reason behind this dependency is that kernels executing later require the output of the kernels preceding it. To note, for a kernel to run on all particles, the `globalSize` variable was set to be equal to the total number of particles.

The results of testing using `hyperfine` with arguments of `h=0.001` and `e=0.00001` can be seen below in Table 2. Larger test files were created manually by copy and pasting the values from `s42-50.in`. These files were used primarily to test for performance, whereas `s42-50.in` was used to test for correctness and performance. It can be seen that for input files of sufficiently large size, `protons_ocl` outperforms `protons_sin`. In this case, it ran roughly 26x faster on `big_10000.in`. In all other cases, it can be seen that the overhead from parallelization lead to `protons_ocl` running slower than its sequential counterpart. Interestingly enough, this trend also follows when comparing `protons_ocl` to `protons_omp`, however in this case the speed up is only about 2.47x.

Table 2: Results for different Coulomb's law executions on `ecetesla0`

Description	**Mean (s)**	σ **(s)**	**Min (s)**	**Max (s)**
`protons_sin`, `s42-50.in`	0.0138	0.0021	0.0092	0.0187
`protons_sin`, `big_1000.in`	0.1238	0.0025	0.1177	0.1293
`protons_sin`, `big_10000.in`	8.537	0.243	8.224	8.981
`protons_omp`, `s42-50.in`	0.0119	0.0032	0.0068	0.0288
`protons_omp`, `big_1000.in`	0.1302	0.0338	0.057	0.1789
`protons_omp`, `big_10000.in`	0.8110	0.0647	0.6541	0.8652
`protons_ocl`, `s42-50.in`	0.2166	0.0128	0.2070	0.2572
`protons_ocl`, `big_1000.in`	0.2415	0.0116	0.2277	0.2738
`protons_ocl`, `big_10000.in`	0.3281	0.0058	0.3211	0.3411