Part 1

Program.cl

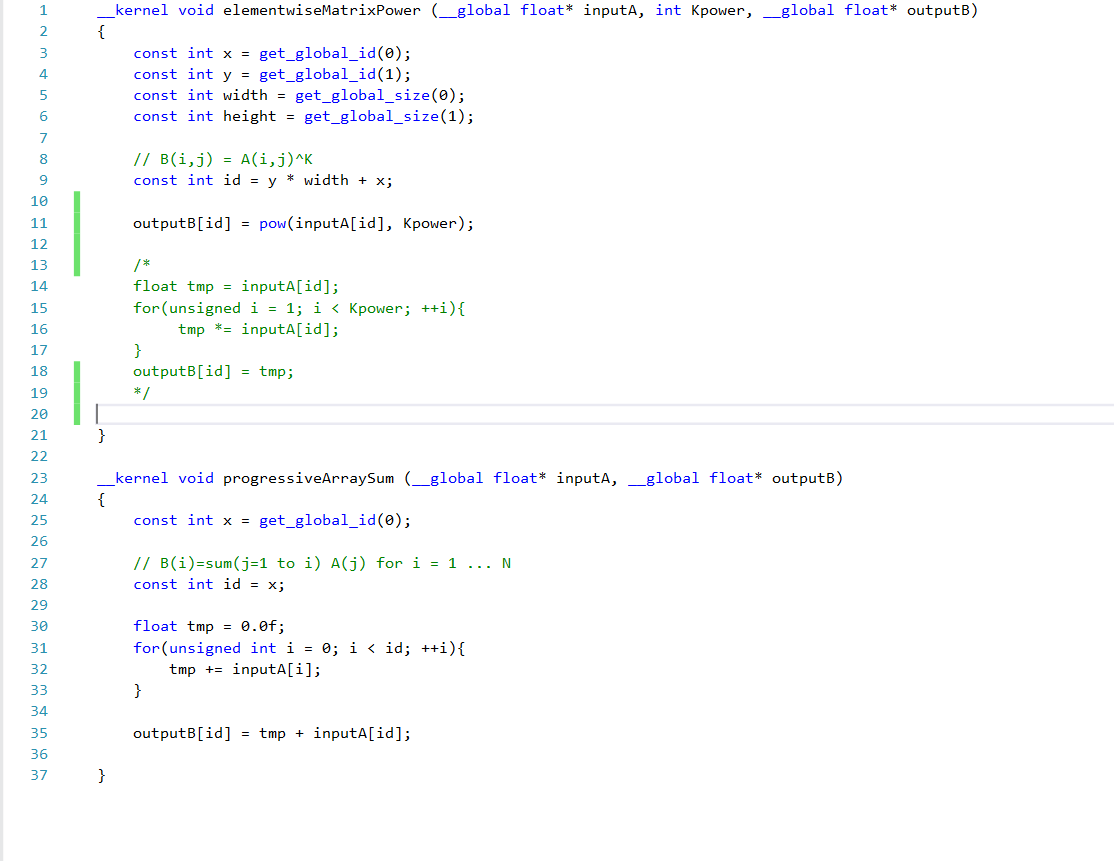


Figure-1

For elementwiseMatrixPower, both the manual and kernel function pow() works. And they are verified with Matlab output shown in figure-2, 3, and 4.

Some output might fail due to floating point precision and overflow. Kernel built using single precision floating value, while Matlab isn’t. If Matlab was using single precision floating point value to calculate, the same output would be produced.

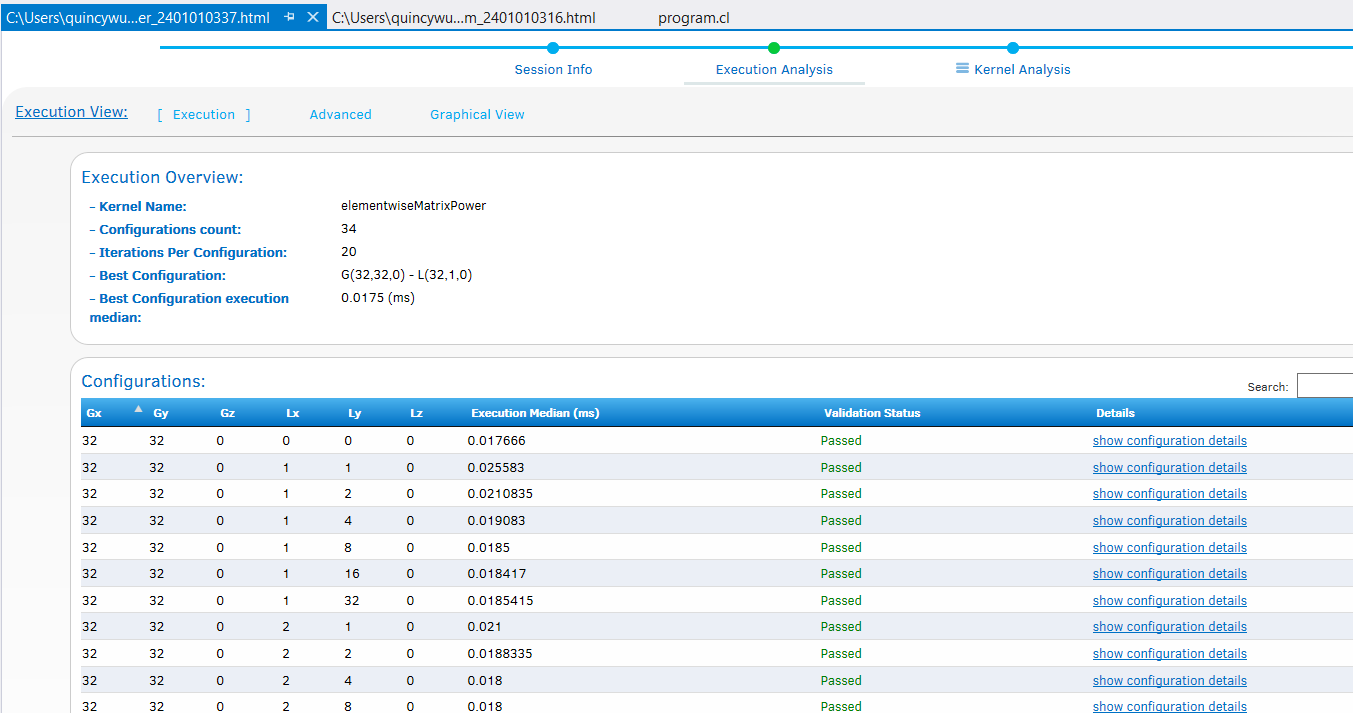
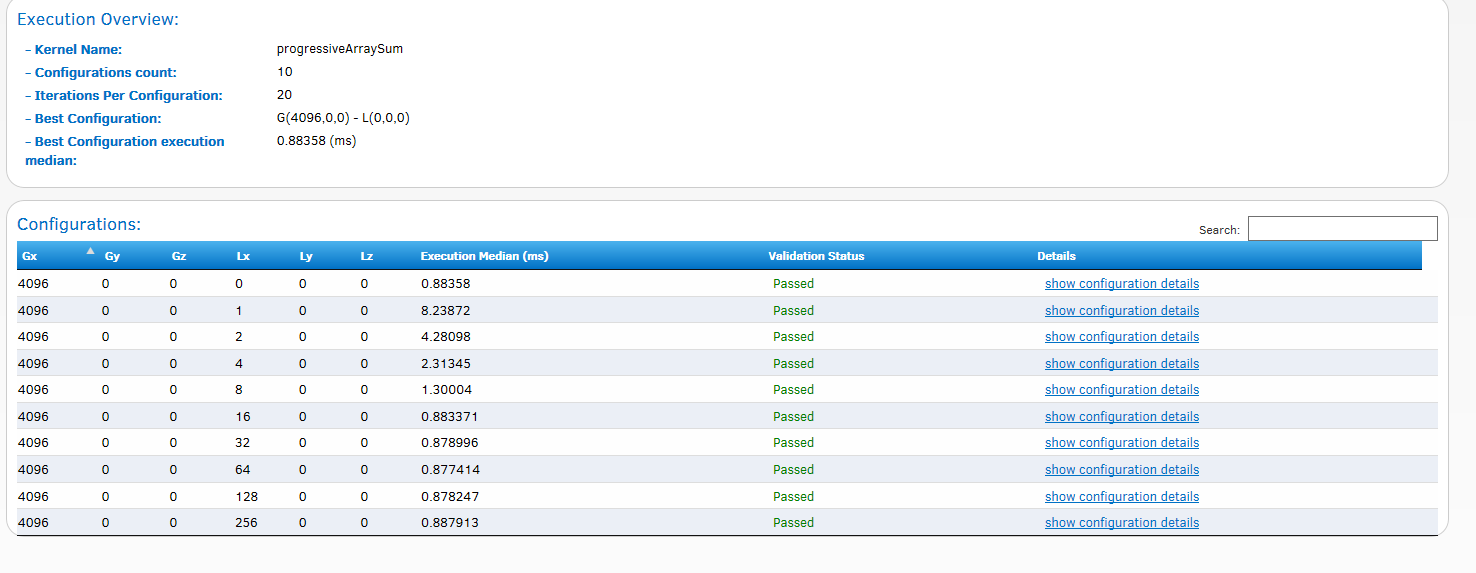


Figure-2, elementwiseMatrixPower passed verification with restricted input.



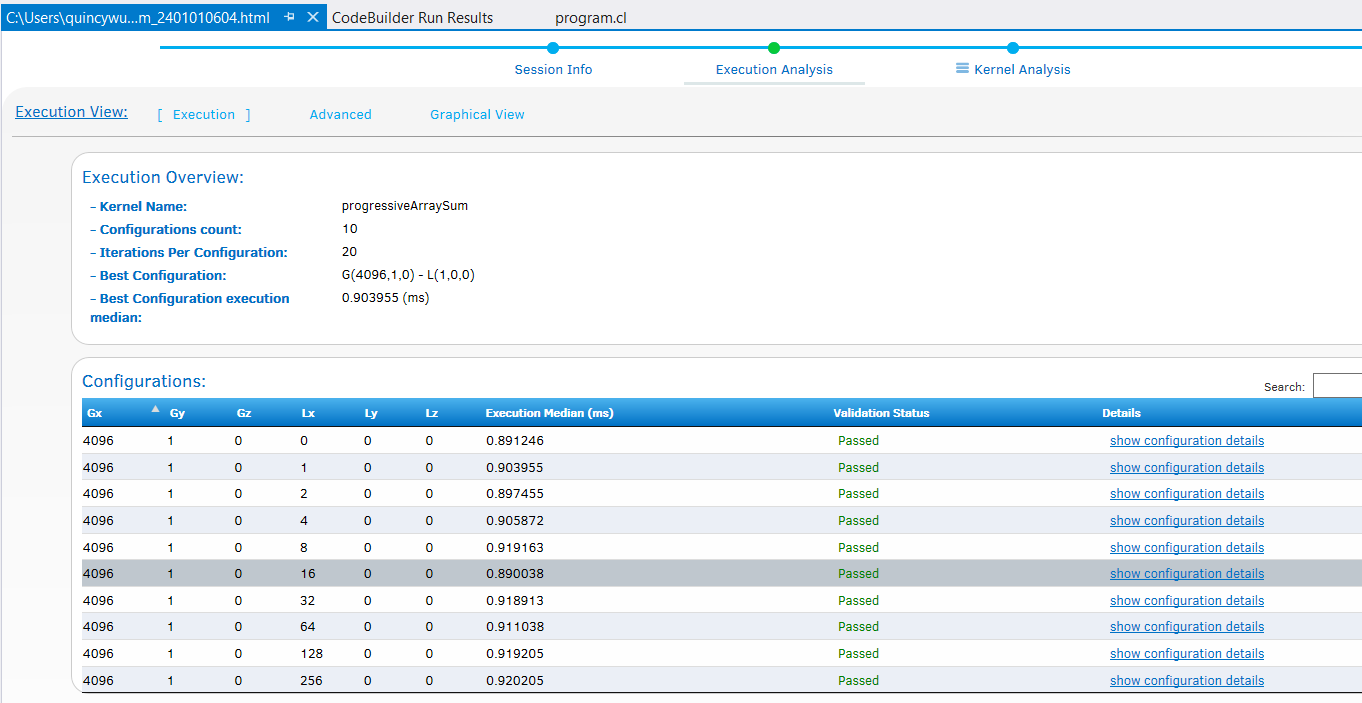


Figure-3, 4, progressiveArraySum passed verification with restricted input.

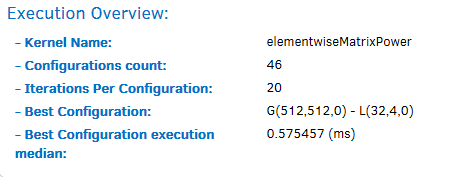
Part 2

Using Global sizes

* + - 1. elementwiseMatrixPower = {512,512,0}
      2. progressiveArraySum ={4096,0,0}

Using random number generated by kernel. Tested on smaller size and restricted input, output is correct. Without restricting input, output would be incorrect due to floating point precision error.

Built using GPU targets



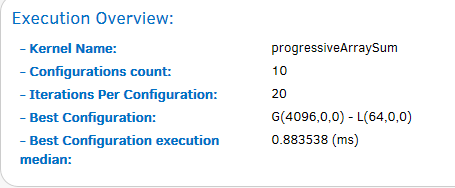
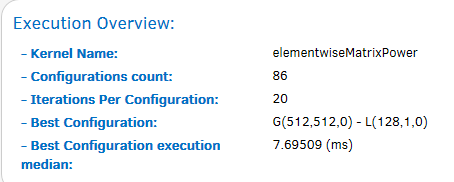


Figure-5, 6 execution result of building kernel with GPU targets

Built using CPU targets



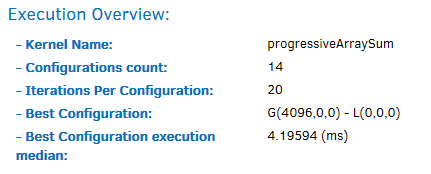
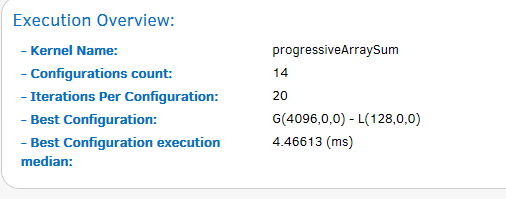
 

Figure-7, 8, 9, execution result of building kernel with CPU targets

Two execution over for progressiveArraySum was created due to the unknown result of local work group size of (0, 0, 0). The local work group size should at least be (1, 0, 0). And result on figure 9, the local work group size of (128, 0, 0) is used for the rest of the report.

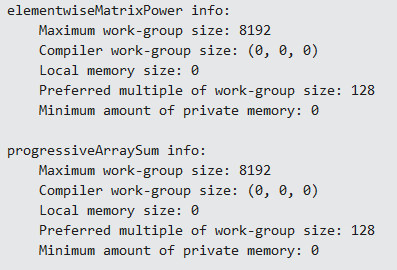


Figure-10 CPU Preferreed multiple of work-group size

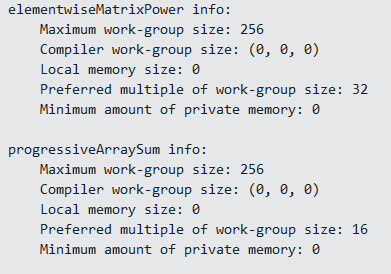


Figure-11 GPU Preferreed multiple of work-group size

For CPU, the Preferreed multiple of work-group size is **128**, the same for both kernel, and it is shown in the execution overview on Figure 7, 9. The best configuration is matching the work-group size **128**. In Figure 8, Best Configuration local work group size is (0, 0, 0). It might be due the problem size is too small, and setup overhead is more affecting the result.

For GPU, the preferred multiple of work-group size is different for both kernel. They are also different from the best configuration on kernel by runtime. elementwiseMatrixPower, has the preferred work-group size of **32**, while the best configuration work-group size is **(32, 4, 0).** The progressiveArraySum has preferred work-group size of 16, while the best configuration work-group size determined by runtime is **(64, 0, 0)**. The difference might due to number of iteration ran on the kernel is not enough and theory for preferred work-group size might be different than runtime determined work group size.

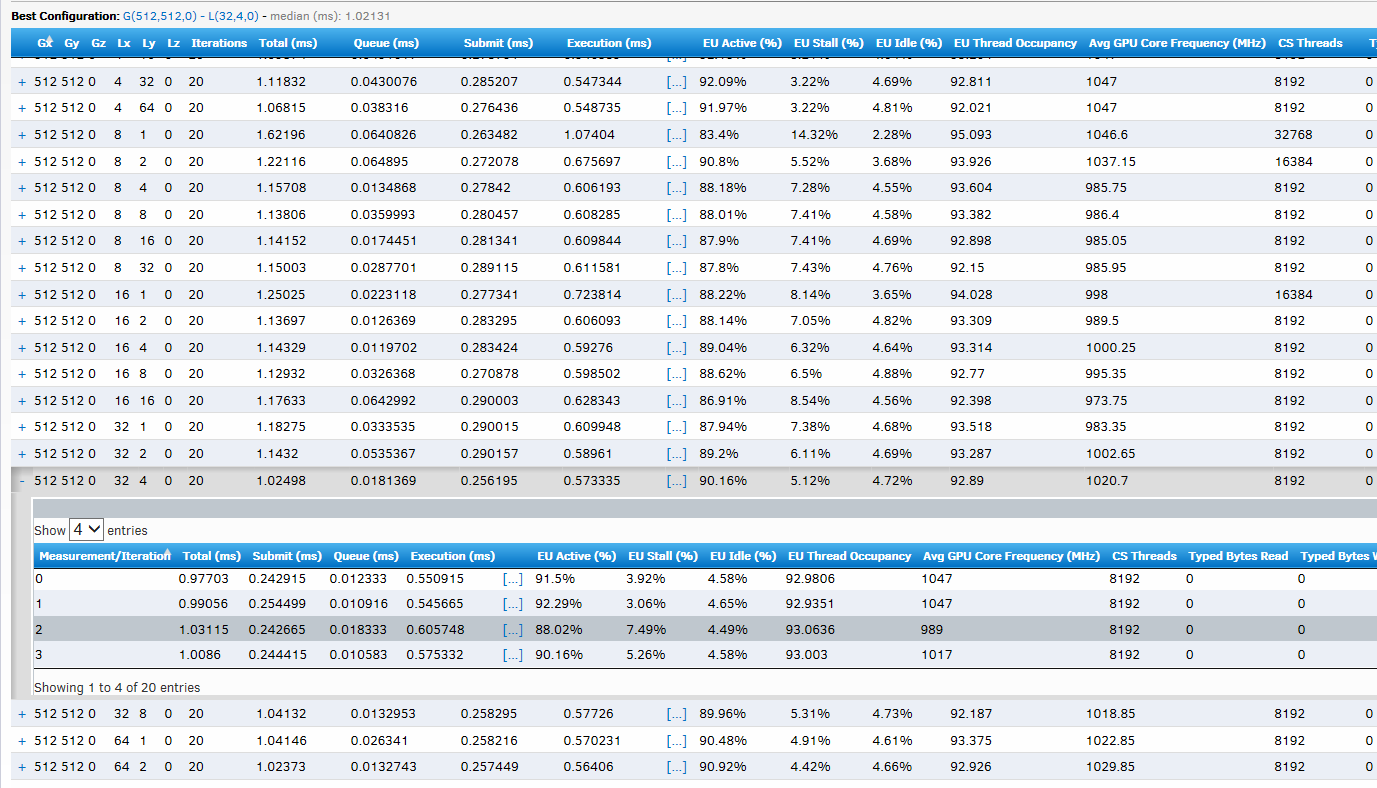


Figure - 12 Advance option on execution analysis

Best configuration for local size is not simply based on fastest execution time. It is combination of queue, submit and execution time to produce a total time. EU Active percentage also has effect on the determination of best local size.

GPU

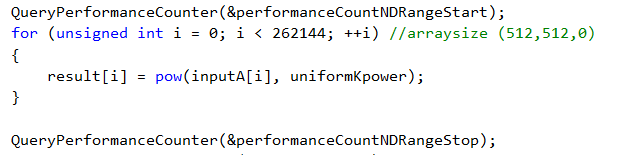


Figure – 16, Sequential code for elementwiseMatrixPower using GPU target



Figure – 17, elementwiseMatrixPower sequential timing using GPU target

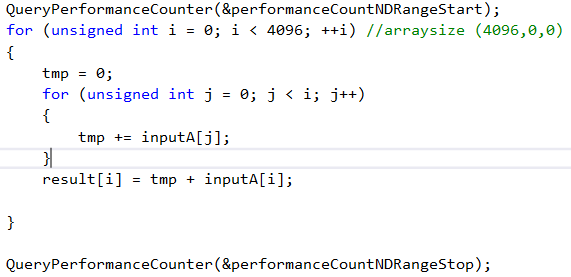


Figure – 13, Sequential code for progressiveArraySum using GPU target. (Slow, without caching result)

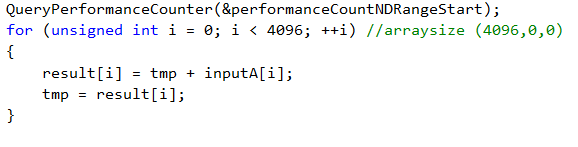


Figure – 14, Sequential code for progressiveArraySum using GPU target. (Fast, with caching result)

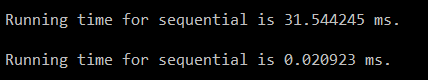


Figure – 15, progressiveArraySum sequential timing

Both function ran on GPU and collected the timing. On elementwiseMatrixPower, this profiling timing for the sequential code was much slower than the kernel. The kernel was faster than sequential by about **7.8** times. On progressiveArraySum, the problem was analyzed in two ways. Analyzing the kernel without caching the previous result would be solving the same problem as the kernel, and would resulted in 31.54 ms, which is about **7** times slower than the kernel. However, this problem can be solve by caching the last result calculated to reduce the amount of computation needed. The kernel was running in parallel, therefore the kernel cannot utilize caching the result. Sequential code was about **200** times faster than kernel after caching.

CPU



Figure – 16, elementwiseMatrixPower sequential timing using CPU target

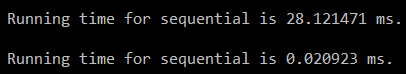


Figure – 17, progressiveArraySum sequential timing using CPU target

AVX2- SIMD optimization option turned on



Figure – 18, elementwiseMatrixPower sequential timing using CPU target

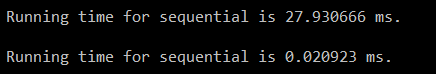


Figure – 19, progressiveArraySum sequential timing using CPU target

Without the AVX2 option turned on, both function were tested targeting the CPU and compared against kernel execution time. ElementwiseMatrixPower was slower than kernel by **6.7** times, and progressiveArraySum without cache was slower than kernel by **6.4** times**,** and with cache was fast than kernel by **200** times. (AVX2 optimization was not turned off, but was not forced to use AVX2 on CPU)

With AVX2 optimization turned on, both function observed a faster timing. The sequential elementwiseMatrixPower function targeting the CPU was **5.6%** faster after the optimization turned on. And progressiveArraySum was **0.67%** faster after the optimization.