Lab 5: Matrix Operations in OpenCL

ECE327: PROJECT 5

November 31, 2019

Bradley Selee
Clemson University
Department of Electrical and Computer Engineering
bselee@g.clemson.edu

Abstract

In project 5, OpenCL was used to compare the the speed of serial processing versus parallel processing. To begin, a C program was created which read in three binary files: two matrices and one vector. Once read, the C program added the two matrices. Then, the results of the addition was then multiplied by the vector. The resulting addition and multiplication were then wrote to an output binary file. Once the C program was finished, OpenCL was used to create the same program but work in parallel. The main difference was that the arithmetic was performed using kernels, allowing the program to run in parallel. Once both programs were created, each was run on the FPGA and the addition, multiplication, and pipeline of each program was timed. The results indicated that at a small sample size, 64x64 matrix, the speed was relatively similar. However, as the problem size group by a factor of 2, the time of the serial processing increased exponentially, while the the parallel processing time was expressing a quadratic growth [2].

1 Introduction

Project 5 gives more in-depth knowledge of OpenCL, unlike Project 2. The goal of this project is to compare the run-time of serial processing versus parallel processing. This was accomplished by adding two matrices and the multiplying this result with a vector. Therefore, no VHDL was required for this project.

Unlike previous labs, one of the first steps was verification. The C program must work correctly, in order for the arithmetic to be properly timed. Initially, the program was tested with a very small sample size, 3x3 matrices and a 3x1 vector. Once the program was verified for correctness, larger matrices were used: 64x64, 128x128, 256x256, 512x512 and 1024x1024. If anything smaller than a 64x64 matrix was used, the timing would be too fast to be accurate. The final results of the serial and parallel portions were timed and plotted against each other.

2 Serial Processing

Serial processing is any sort of processing that involves a single process. Because of this, every instruction must be executed consecutively.

2.1 C Program

To begin this lab, a C program was created to read in three binary files: two matrices and one vector. This program added the two matrices, then multiplied this addition by the vector. The program then wrote two binary files, one contained the addition and the other contained the multiplication. This program was written with only one process in order to compare it with the speed of parallelism.

2.2 Verification

While a 3x3 matrix is will execute incredibly fast and the timing will be inaccurate, it serves well to verify the correct addition and multiplication has been accomplished. In order for the timing to be accurate, the program first must be correct. The figures below show the sample binary files used for testing and their outputs:

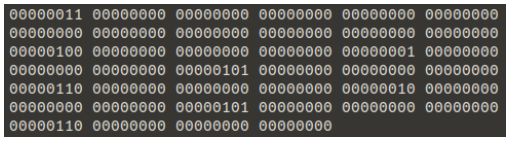

Figure 2.1: 3x3 Matrix 1 Binary File

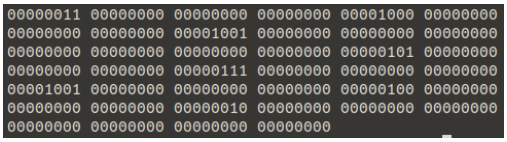

Figure 2.2: 3x3 Matrix 2 Binary File

Figure 2.3: 3x1 Vector Binary File

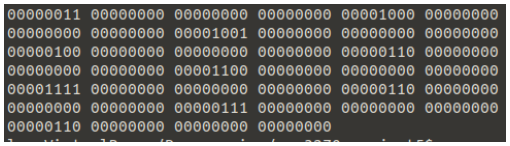

Figure 2.4: Result of the matrix addition

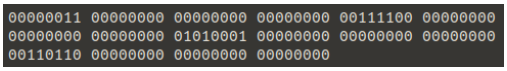

Figure 2.5: Result of the matrix-vector multiplication

3 Parallel Processing

Once the C program was created and verified, the same program was created in OpenCL. The main difference is that the arithmetic was performed in parallel using kernels.

3.1 OpenCL

As in the C program, the I/O for the OpenCL program was exactly the same. The main difference was creating the kernels to perform the arithmetic in parallel. This was very minimal code and the diagram shown below, from the notes, made it very easy to accomplish:

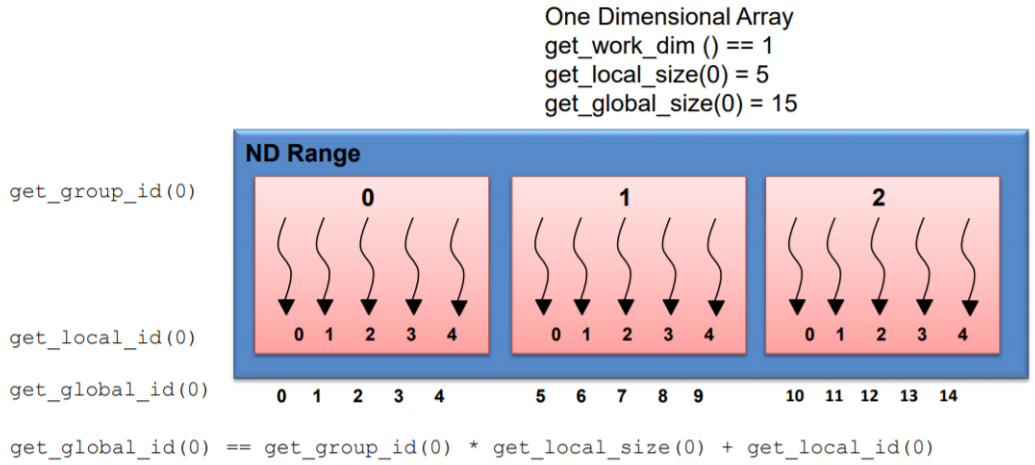

Figure 3.1: Notes diagram explaining ND Range [1]

3.2 Verification

Similarly to the serial portion of this lab, the parallel portion was tested in the same manner. The same matrices and vectors were used and the output files were read to ensure the proper values were created.

4 Timing

Once the serial and parallel portions of the lab were working, both versions had various benchmarks to compare their performance. The addition, multiplication, and pipeline were timed and plotted for both. There were two trials recorded and the average of these two trials were used.

4.1 Serial

The serial processing data represents matrix addition and multiplication with a single process. A comparison of serial versus parallel processing is explained in the results section.

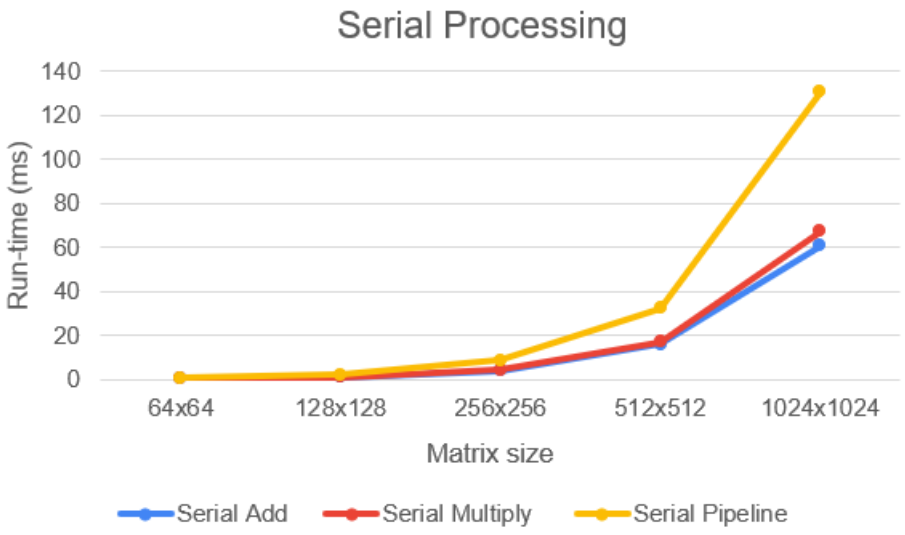

Figure 4.1: Timing analysis of serial processing

4.2 Parallel

The parallel processing data represents matrix addition and multiplication with multiple processes using OpenCL kernels. A comparison of serial versus parallel processing is explained in the results section.

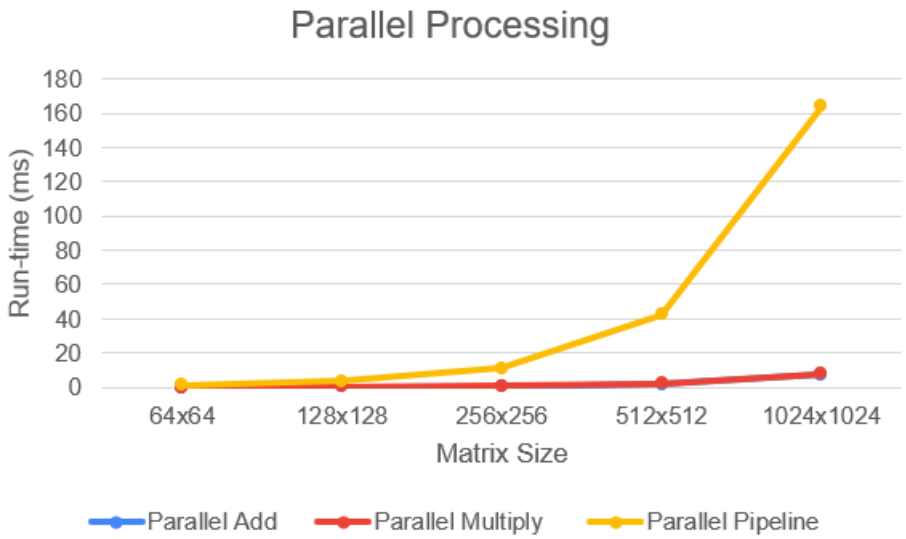

Figure 4.2: Timing analysis of serial processing

5 Discussion and Results

Lab 5 was much different than the other labs, it required no VHDL code or use of Quartus. The goal of this lab was to time the matrix addition, matrix-vector multiplication, and the pipeline for the entire process using serial processing and parallel processing. Once this was accomplished, the timings of the serial and parallel processes were plotted in figure 4.1 and 4.2, respectively. According to the graphs, the serial process appeared start out faster than the parallel process at a small sample size for the arithmetic. However, as the sample size increased, the parallel process began to perform exponentially faster than the serial process. As the sample size increased, there appears to be no substantial difference in the entire pipeline time of both processes. Next, a speed up analysis was run to compare the OpenCL versus the C using the equation below and the results are summarized in a table

$$Speedup = \frac{Time_C}{Time_{CL}}$$

Size	Add Speedup	Mult Speedup
64x64	2.594	3.346
128x128	5.148	5.356
256x256	7.250	7.333
512x512	8.238	8.272
1024x1024	8.287	8.792

From the table, one can clearly that the parallel, OpenCL variation is much faster for the arithmetic. More so, as the matrix size increases, the efficiency of parallelism also

increases, making it much more plausible for large calculations.

6 Conclusion

This lab demonstrated a known fact that calculations run in parallel are substantially faster than processing serially. I believe this lab was successful because through the graphs and data, the OpenCL program was shown to calculate the matrices and vectors faster. There are many causes of error in this lab, the biggest one being the precision of C's built-in timer. If I were to redo this lab, I would try to find a more efficient way of performing the matrix-vector multiplication serially.

References

[1] W. J. Dally, R. C. Harting, and T. M. Aamodt. *Digital Design Using VHDL: A Systems Approach.* Cambridge University Press, Cambridge, 2015.

[2] M. Smith. *Lab 5: Matrix Operations in OpenCL.* Clemson University, 2019.

7 Appendix

No further references were needed for this project.