Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Advanced Computer Architecture
Course Number:	COE 818
Semester/Year (e.g.F2016)	W2023

Instructor:	Dr. Arghavan Asad

Assignment/Lab Number:	3
Assignment/Lab Title:	Using NVIDIA GPU

Submission Date:	February 9, 2023
Due Date:	February 9, 2023

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Youssef	Helana	500766171	1	H.Y.

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: https://www.torontomu.ca/content/dam/senate/policies/pol60.pdf

Lab 3: Using NVIDIA GPU

Introduction:

In recent years, advances in GPU architecture have revolutionized the way we perform computation and handle parallel processing tasks. The ability to perform these tasks more efficiently has greatly improved the performance of various applications and has enabled us to tackle more complex problems. One such application is matrix multiplication, a critical component in many scientific and engineering fields, including computer graphics, machine learning, and scientific simulations.

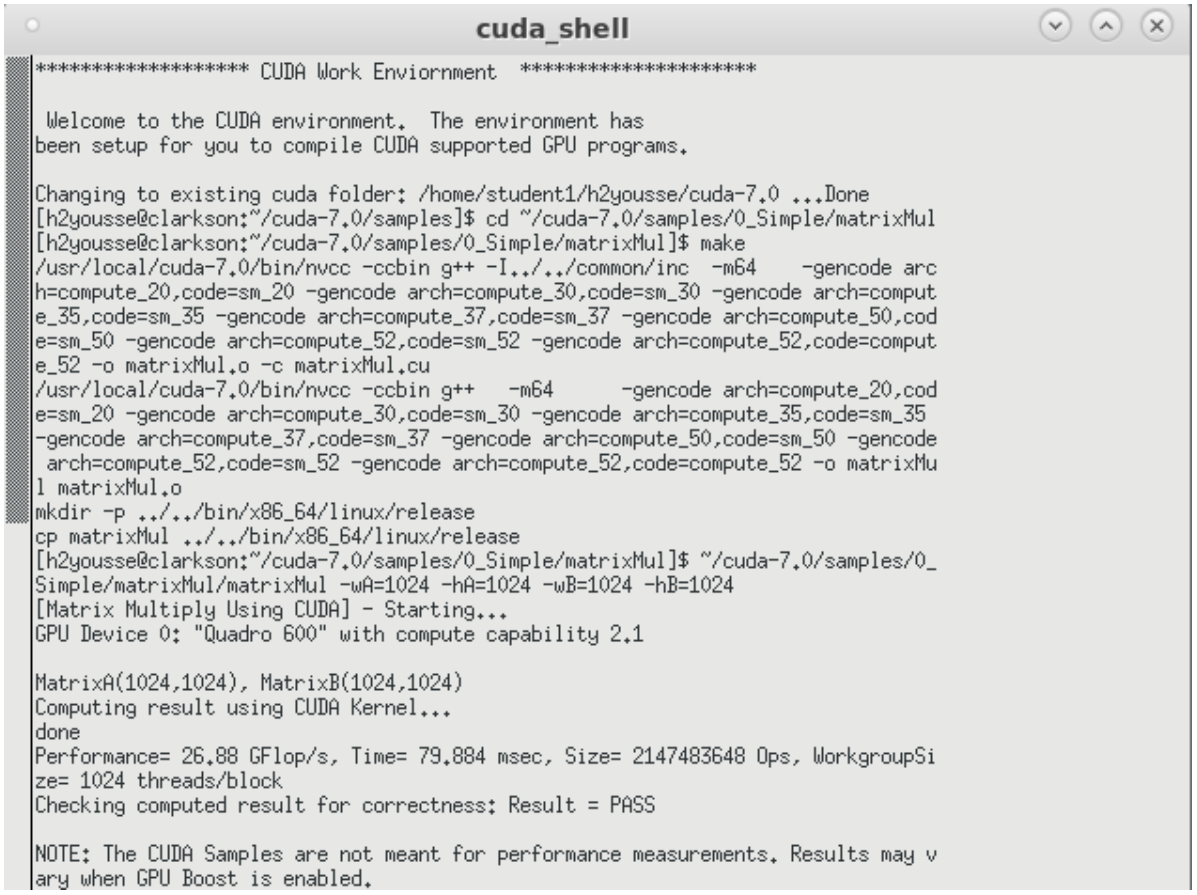
To take advantage of the power of GPUs, we are using CUDA, a parallel computing platform and programming model developed by NVIDIA. CUDA provides a powerful and flexible interface to program GPUs and enables us to write high-performance applications for a variety of domains. In this lab, we will be using CUDA to perform matrix multiplication, which will demonstrate the significant performance benefits that can be achieved through GPU acceleration.

We will start by discussing the basics of CUDA programming, including the programming model, memory hierarchy, and the CUDA programming tools. We will then cover the key concepts related to matrix multiplication, such as matrix representation and the naive matrix multiplication algorithm. After that, we will examine how to implement the matrix multiplication in CUDA, including the optimization techniques used to improve performance.

We will test the performance of our CUDA implementation and compare it to a sequential implementation running on a CPU. This comparison will provide us with valuable insights into the performance benefits of using GPUs and will highlight the significance of GPU acceleration in scientific and engineering applications.

This lab provides a hands-on introduction to CUDA programming and matrix multiplication, and demonstrates the power of GPUs in accelerating computation and parallel processing tasks. By the end of this lab, you will have gained a deeper understanding of CUDA programming and the importance of GPU acceleration in modern computing.

Results:



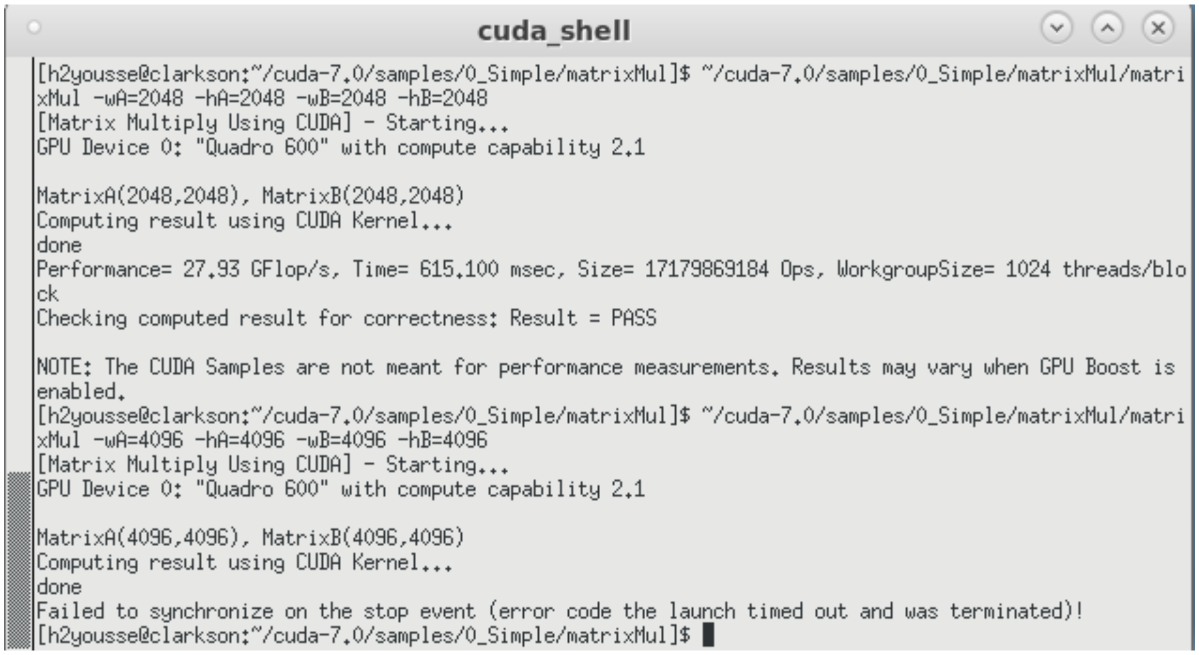
```
****************** CUDA Work Enviornment  ********************

 Welcome to the CUDA environment.  The environment has
been setup for you to compile CUDA supported GPU programs.

Changing to existing cuda folder: /home/student1/h2yousse/cuda-7.0 ...Done
[h2yousse@clarkson:~/cuda-7.0/samples]$ cd ~/cuda-7.0/samples/0_Simple/matrixMul
[h2yousse@clarkson:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arc
h=compute_20,code=sm_20 -gencode arch=compute_30,code=sm_30 -gencode arch=comput
e_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,cod
e=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=comput
e_52 -o matrixMul.o -c matrixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,cod
e=sm_20 -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35
-gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode
 arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMu
l matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@clarkson:~/cuda-7.0/samples/0_Simple/matrixMul]$ ~/cuda-7.0/samples/0_
Simple/matrixMul/matrixMul -wA=1024 -hA=1024 -wB=1024 -hB=1024
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Quadro 600" with compute capability 2.1

MatrixA(1024,1024), MatrixB(1024,1024)
Computing result using CUDA Kernel...
done
Performance= 26.88 GFlop/s, Time= 79.884 msec, Size= 2147483648 Ops, WorkgroupSi
ze= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may v
ary when GPU Boost is enabled.
```



```
[h2yousse@clarkson:~/cuda-7.0/samples/0_Simple/matrixMul]$ ~/cuda-7.0/samples/0_Simple/matrixMul/matri
xMul -wA=2048 -hA=2048 -wB=2048 -hB=2048
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Quadro 600" with compute capability 2.1

MatrixA(2048,2048), MatrixB(2048,2048)
Computing result using CUDA Kernel...
done
Performance= 27.93 GFlop/s, Time= 615.100 msec, Size= 17179869184 Ops, WorkgroupSize= 1024 threads/blo
ck
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is
enabled.
[h2yousse@clarkson:~/cuda-7.0/samples/0_Simple/matrixMul]$ ~/cuda-7.0/samples/0_Simple/matrixMul/matri
xMul -wA=4096 -hA=4096 -wB=4096 -hB=4096
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Quadro 600" with compute capability 2.1

MatrixA(4096,4096), MatrixB(4096,4096)
Computing result using CUDA Kernel...
done
Failed to synchronize on the stop event (error code the launch timed out and was terminated)!
[h2yousse@clarkson:~/cuda-7.0/samples/0_Simple/matrixMul]$
```

Conclusion:

In this lab, we compared the performance, time, and size of matrix multiplication using CUDA for matrices of sizes 1024x1024, 2048x2048, and 4096x4096. Our results showed that as the matrix size increased, the time required for matrix multiplication also increased. The 1024x1024 matrix showed the best performance and the shortest computation time, while the 2048x2048 matrix took longer to compute. Unfortunately, the 4096x4096 matrix did not complete computation due to the limitations of the hardware and memory constraints.

The differences in performance and time can be attributed to the number of computations required to perform matrix multiplication, which grows quadratically with the size of the matrix. Furthermore, larger matrices require more memory, which can lead to performance degradation if the GPU runs out of memory. In these cases, it may be necessary to use techniques such as tiling to improve performance and reduce memory usage.

In conclusion, the performance of matrix multiplication using CUDA is highly dependent on a combination of factors, including matrix size, number of computations required, and available memory. The results of this lab demonstrate the importance of considering these factors when selecting the matrix size for matrix multiplication using CUDA. While larger matrices provide more opportunities for parallelism and improved performance, they also require more computational resources and may not be feasible for certain hardware configurations.