Ryerson University

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

# COE818 – Advanced Computer Architecture

# LAB REPORT

| | |
|---|---|
| **Semester/Year:** | Winter 2025 |
| **Lab Number:** | 4 |

| | |
|---|---|
| **Instructor:** | Dr. Nagi Mekhiel |
| **Section No:** | 012 |
| **Submission Date:** | 2025-03-30 |
| **Due Date:** | 2025-03-30 |

| Student Name | Student ID | Signature |
|---|---|---|
| Carlo Dumlao | 501018239 | C.D |
| | | |
| | | |

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: www.ryerson.ca/senate/current/pol60.pdf

# 1.0 Objective

To investigate how the number of threads per block affects the scalability (or performance) of a GPU.

# 2.0 Introduction

GPUs were initially designed to accelerate the rendering of 3d graphics, however, over time, its hardware became more flexible and programmable, improving their capabilities. Nowadays, they are not only used for image and video processing, but other computational extensive applications as well such as machine learning, cryptocurrency mining, and performing AI models.

In this lab, the performance improvement of a GPU – in particular, Quadro 600 – was explored by executing a simple matrix multiplication, with size 512x512 and 1024x1024, under the CUDA environment. For each matrix size, the number of threads per block was incremented gradually from 2 to 32. By analysing the resulting execution times and scalability of the GPU, the optimum number of threads per block was determined – that is, the number of threads in which the GPU performs the fastest.

# 3.0 Hypothesis

The GPU that was used in this lab is an NVIDIA Quadro 600, which has the following specifications:

Maximum number of active blocks per SM = 8
Maximum number of threads per SM = 1536

These specifications must be taken into consideration as they set the limit on the number of threads (and blocks) within a streaming multiprocessor (SM). In other words, if the number of threads per block is not well optimized based on these values, it may lead to low occupancy in the SM which consequently degrades the overall parallelism and performance of the GPU. Table 3.1 and Figure 3.1 shows the occupancy with varying number of threads per block. The occupancy is calculated with the following formula:

$$Occupancy\ \% \ = \ \frac{(\#\ of\ Threads\ per\ Block)^2 (\#\ of\ Allowable\ Blocks)}{1536} \times 100 \qquad \text{Eq. 3.1}$$

Notice that in the Figure, 16 threads per block achieves 100% occupancy in the SM. Hence, this value is expected to have a higher GPU performance among the others.

Table 3.1: SM occupancy of Quadro 600 for a given number of threads per block.

| Threads per Block | Number of Allowable Blocks | Occupancy (%) |
|---|---|---|
| 2 | 8 | 2.083333333 |
| 4 | 8 | 8.333333333 |
| 8 | 8 | 33.33333333 |
| 16 | 6 | 100 |
| 32 | 1 | 66.66666667 |

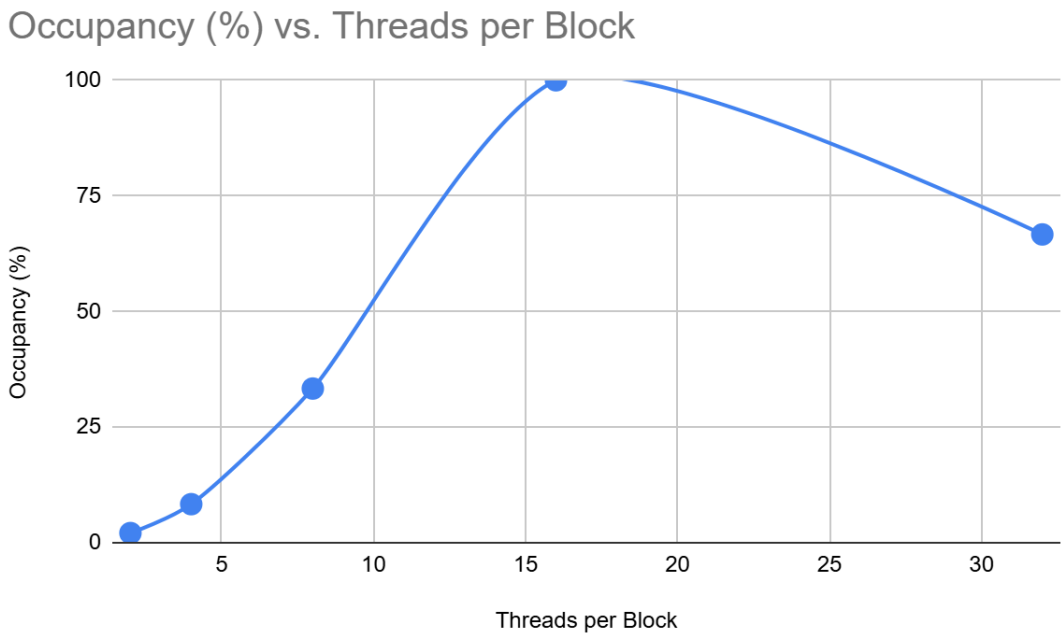


Figure 3.1: Relationship between the percentage SM occupancy and the number of threads per block.

# 4.0 Result

## 4.1 Execution Time

As shown in Figure 4.1 below, the execution time of the GPU is somewhat inversely proportional to the number of threads per block. When the number of threads is increased from 2 to 16, there is a drastic improvement in the speed up: the GPU with 16 threads is 20 times faster in comparison to the initial value of 2 threads. It is important to note, however, that beyond that 16 threshold, the GPU starts to experience a slight reduction in performance. In fact from 16 to 32, the execution time is increased by approximately 20%. The collected execution time of GPU and CPU, and its scalability for several numbers of threads per block are tabulated in Table 4.1 and Table 4.2.

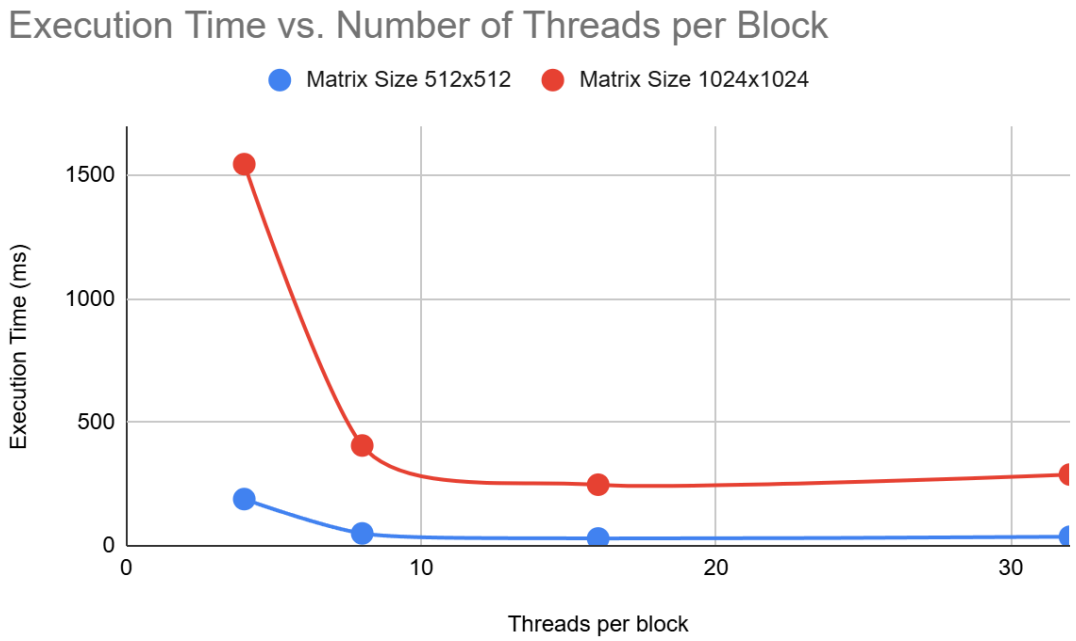


Figure 4.1:Relationship between the execution time of the GPU and the number of threads per block.

Table 4.1: Execution time of both the CPU and GPU, as well as the calculated scalability for a matrix size of 512x512.

| Threads per block | CPU (ms) | GPU (ms) | Scalability |
|---|---|---|---|
| 2 | 850 | 595.89 | 1.426437765 |
| 4 | 810 | 189.12 | 4.282994924 |
| 8 | 820 | 50.111 | 16.36367265 |
| 16 | 820 | 29.685 | 27.62337881 |
| 32 | 810 | 36.722 | 22.05762213 |

Table 4.2: Execution time of both the CPU and GPU, as well as the calculated scalability for a matrix size of 1024x1024.

| Threads per block | CPU (ms) | GPU (ms) | Scalability |
|---|---|---|---|
| 2 | 6840 | 5846.88 | 1.169854692 |
| 4 | 6980 | 1545.2 | 4.5172146 |
| 8 | 7380 | 406.05 | 18.17510159 |
| 16 | 6420 | 247.37 | 25.95302583 |
| 32 | 6550 | 288.5 | 22.70363951 |

## 4.2 Scalability

The scalability of the GPU over the number of threads per block is plotted in Figure 4.2. To calculate the scalability, the following formula is used:

$$Scalability = T_{seq}/T_{parallel} \qquad \text{Eq. 4.1}$$

where $T_{seq}$ is the time it takes to execute the matrix multiplication sequentially (i.e. in the CPU) and $T_{parallel}$ is the execution time using the GPU. Notice that the Figure is very similar to that of the theoretical Figure 3.1 – both graphs present an inverted parabola with a maximum point positioned at 16 threads per block. This implies that there is strong correlation between the occupancy of threads in the SM and the scalability; the higher that occupancy is (or the greater the number of threads active in the SM), the better scalability of the GPU will offer.

Furthermore, it is apparent that when the matrix size is further increased to 1024x1024, the resulting scalability is slightly worse. This degradation of performance is primarily due to the memory latency experienced in the GPU. Considering the 1024x1024 matrix involves a greater workload, the cores will now spend more of their time waiting to retrieve or fetch the necessary data (or operands) from the off-chip global memory, rather than doing any useful calculations.

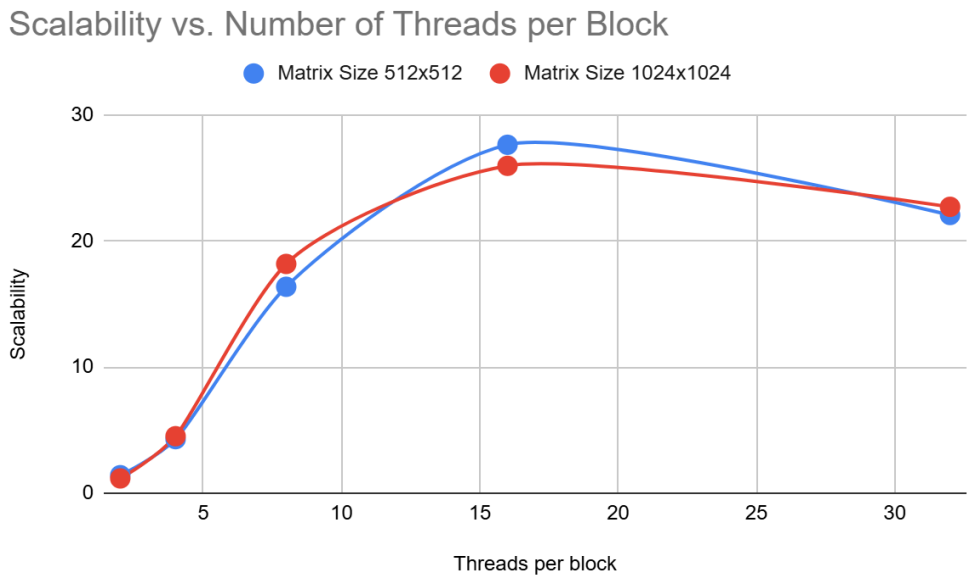


Figure 4.2: Relationship between the scalability of the GPU and the number of threads per block.

# 5.0 Conclusion

The SM occupancy correlates to the overall scalability of the GPU: setting the number of threads per block such that it maximizes the number of active threads in all of the multiprocessors would yield better performance and shorter execution time of the GPU. In this lab, if one wishes to get the best out of the NVIDIA Quadro 600 when calculating a matrix multiplication, the recommended number of threads per block is 16.

# 6.0 Appendix

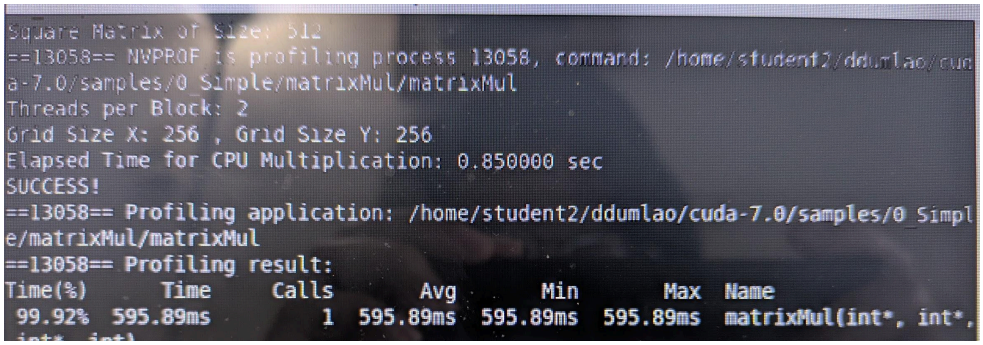


Figure 6.1: Output terminal of matrix size 512x512 with a threads per block of 2.

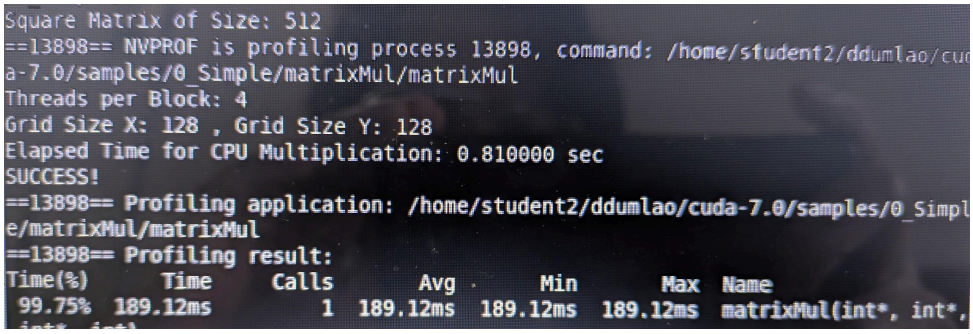


Figure 6.2: Output terminal of matrix size 512x512 with a threads per block of 4.

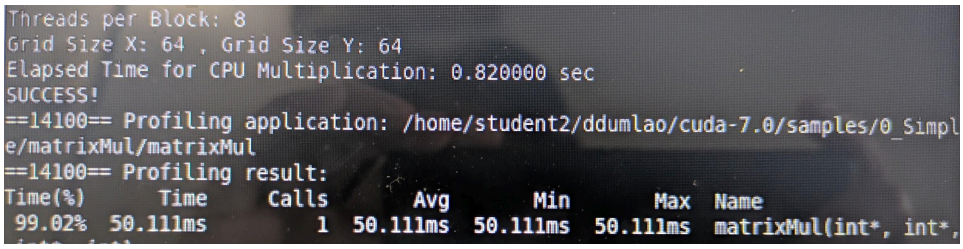


Figure 6.3: Output terminal of matrix size 512x512 with a thread per block of 8.

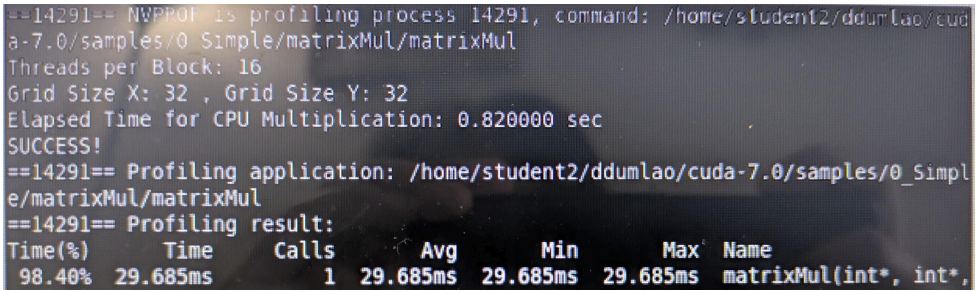


Figure 6.4: Output terminal of matrix size 512x512 with a thread per block of 16.

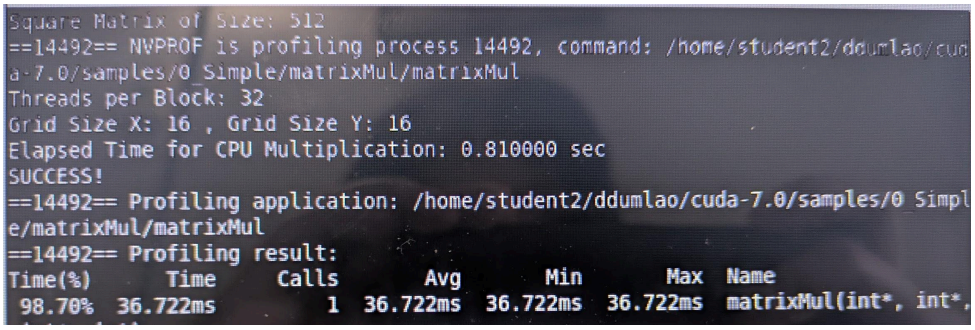


Figure 6.5: Output terminal of matrix size 512x512 with a thread per block of 32.

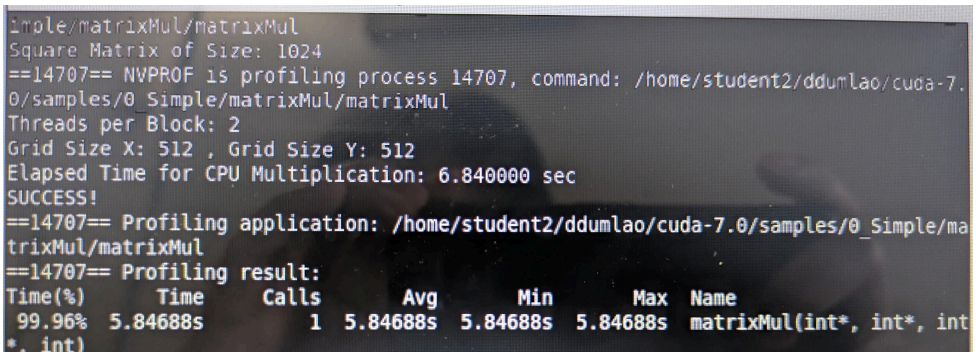


Figure 6.6: Output terminal of matrix size 1024x1024 with a thread per block of 2.

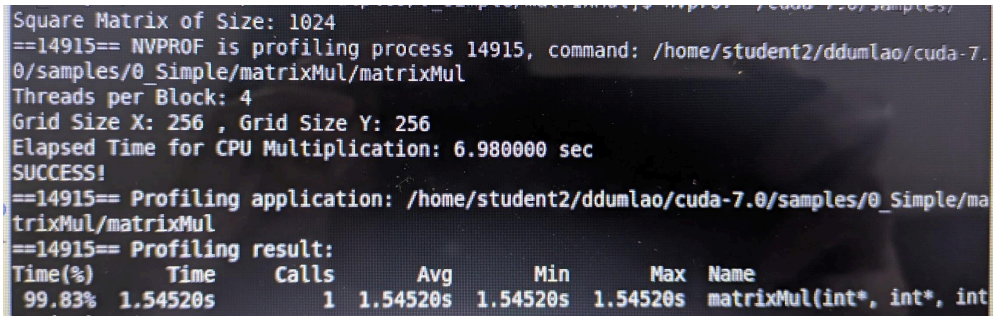


Figure 6.7: Output terminal of matrix size 1024x1024 with a thread per block of 4.

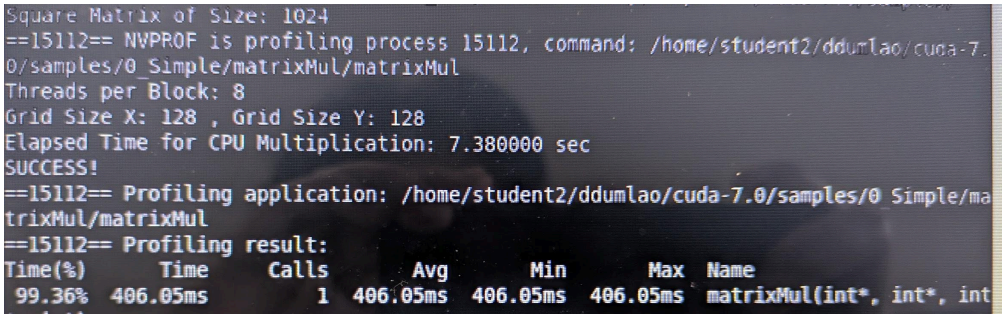


Figure 6.8: Output terminal of matrix size 1024x1024 with a thread per block of 8.

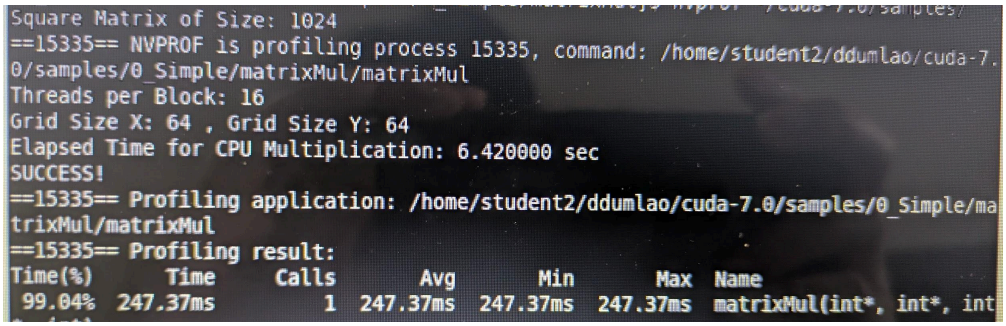


Figure 6.9: Output terminal of matrix size 1024x1024 with a thread per block of 16.

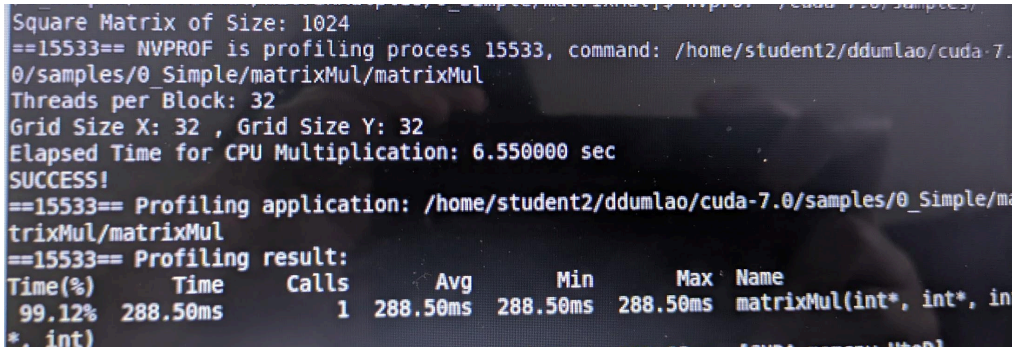


Figure 6.10: Output terminal of matrix size 1024x1024 with a thread per block of 32.