Florida Institute of Technology
Department of Electrical Engineering & Computer Sciences
ECE5550 – High Performance Computing

Module 5 Lab: Basic Matrix Multiplication

Due Date: 8-Mar-2025

1. Objective

The purpose of this lab is to introduce mathematical computation on the GPU. Matrix-matrix multiplication, or matrix multiplication in short, is an important component of the Basic Linear Algebra Subprograms standard. It is the basis of many linear algebra solvers, such as LU decomposition. It is also an important computation for deep learning using convolutional neural networks.

Deployment Platform: Google Colab Jupyter Notebook

Environment Setup: See the Mod 5 Lab notebook

These lines are shell commands executed within the Jupyter Notebook using the `!` prefix. They are primarily focused on setting up and verifying the CUDA environment, which is necessary for running code on NVIDIA GPUs.

1. `!nvidia-smi`: This command displays information about the available NVIDIA GPUs on the system, including their model, memory usage, and current processes running on them. This is useful for verifying that your GPU is recognized and accessible.
2. `!nvcc --version`: This command displays the version of the NVIDIA CUDA Compiler (`nvcc`) installed on the system. This is important because CUDA code needs to be compiled with a compatible compiler version for the target GPU architecture.

2. Procedure

Step 1: Copy the lab files to a folder in Google Drive under your student (fit.edu) account.

Step 2: Edit the file `main.cu` to implement the following where indicated:

a) Allocate device memory
b) Copy host memory to device
c) Copy results from device to host
d) Free device memory

Step 3: Edit the file `kernel.cu` to initialize the thread block and kernel grid dimensions and invoke the CUDA kernel, and to implement the matrix multiplication kernel code.

Step 4: Compile and test your code.

```
!rm -rf *.o

!nvcc -arch=sm_75 -o sgemm kernel.cu support.cu main.cu

!./sgemm                 # Uses the default matrix sizes
```

```
!./sgemm <m>            # Uses square m x m matrices

!./sgemm <m> <k> <n>    # Uses (m x k) and (k x n) input matrices
```

It is a good idea to test and debug initially with small input dimensions. Your code is expected to work for varying input dimensions – which may or may not be divisible by your block size – so don't forget to pay attention to boundary conditions.

Step 5: Write a python script to perform matrix to matrix multiplication:

Write the python code that does matrix to matrix multiplication. C is the variable name for the output of the 2 matrices $A\ x\ B$. $C = AxB$ The default $m\ x\ n$ size of the matrices is 1000 by 1000. Input dimensions $m\ x\ n$ can be given to specify the matrix size. The matrics A and B should be populated with random numbers between 1-100. Placing timing calls around ONLY the matrix multiplication call and print the timing results along with the dimension of the matrix.

Step 6: Answer the following questions in a new file named **answers.pdf** (generate a pdf) for submission.

1. How many times is each element of each input matrix loaded during the execution of the kernel? Discuss and provide details.
2. Populate the table comparing the runtime results of the Python code versus the Cuda code on similar size matrices. Write a discussion on your results then add the table and discussion to the **answers.pdf** file. Plot a line graph of the results.

   Sample Table

Python vs Cuda Runtime Comparison		
Matrix Dimension	**Python (ms or sec)**	**Cuda (ms or sec)**
10x10		
100x100		
500x500		
1000x1000		
2500x2500		
5000x5000 (if possible)		
10000x10000 (if possible)		

Submission process

Zip all the files required in the submission into one format (*.zip, *.7z, *.tar) compressed formats.

Name the compressed file using the following naming convention, lastname_assignment_number.zip.
Example "smith_Lab5.zip"
Note: CANVAS will be restricted to accepting these formats **ONLY**.

Files required in submission

File(s)	**Percentage of Total Score**
1. lastname_assignment_number.zip (includes see below) a. main.cu b. kernel.cu c. Mod5-Lab.ipynb (with Python code and plotting) d. answers.pdf	100%

Your code submission will be graded based on the following criteria.

Score	**70%**	**80%**	**90%**	**>90%**
Description	A "valid attempt" to use and develop concepts that are required for the lab subject.	The submission must complete the minimum required to receive 70%. The submission must be able to be compiled with **NO** errors and produced and executable.	The submission must complete the minimum required to receive 80%. The submission must reproduce the expected output results.	The submission must complete the minimum required to receive 90%. The submission must demonstrate neatness and clarity in the code and proficiency in coding practices.

***NOTE**: All late submissions will start with a grade equivalent to the lowest grade of on-time submission. Then subject to the same criteria as defined in the grading rubric.

Note: This is a simple but essential exercise. Please write out the code and do not copy it from other examples or lecture slides. That process is most important.