EE/CSCI 451
Fall 2024
Programming Homework 6

Assigned: November 5, 2024
Due: November 19, 2024, submit via Brightspace
Total Points: 100

General Instructions

- You may discuss the algorithms. However, the programs have to be written individually.
- Submit **the source code, makefile and a simple report** via **Brightspace**. The report should include the screenshots on the output from CARC. The source codes should be named 'p1.cu' and 'p2.cu'. Put all the files in a zip file with file name `<firstname>_<uscid>_phw<programming homework number>.zip` (do not create an additional folder inside the zip file). For example, `alice_123456_phw6.zip` should contain only the source codes and the report.
- Your program should be written in C or C++. You can use any operating systems and compilers to develope your program. However, we will test your program on a x86 linux with the latest version of g++ and gcc. Make sure you makefile could compile the executables for **all** the problems (hint: set multiple targets for the makefile) and the name of the executables are correct. If your program has error when we compile or run your program, you will lose at least 50% of credits.
- While you can also use your own platform, we recommend you run this work on CARC using the P100 GPU.

1 Examples

"async.cu" implements an exmaple program that uses asynchronous memory copy to overlap computation and communication. The program creates *nStreams* of CUDA streams, partition the original task into *nStreams* sub-tasks, and distribute the sub-tasks to each stream. The example code shows two approaches to schedule the asynchronous tasks. The first approach enqueue the sub-tasks into the streams in a round-robin fashion by looping over {memory copy, kernel function, memory copy}; the second approach first enqueue all the memory copy sub-tasks to the streams, and then all the kernel functions, and so on. In other words, there are three loops, each loops over one type of sub-task (e.g., kernel function). For more details on the difference between the two approaches, you may refer to: https://developer.nvidia.com/blog/how-overlap-data-transfers-cuda-cc/

2 Matrix Multiplication [100 points]

In the lecture and discussion, we discussed the naive matrix multiplication ($C = A \times B$) without any optimization. In this assignment, your task is to implement 1024×1024 matrix multiplication using two approaches to overlap computation and communication. In the original algorithm (synchronous version), we first copy the entire matrices A and B, perform matrix multiplication, and copy the result C back. In the asynchronous version, there are multiple streams, and each stream is responsible to update a tile of C. Figure 1 shows an example for nStreams = 4, which means there will be four asynchronous memory copy (one in each stream) each copying a tile of A, and computes a tile of C. In order to compute a tile of C, the entire matrix B is required. You may perform a synchronous memory copy for the entire matrix B before performing asynchronous memory copy for matrix A. In the original algorithm, we have only one function call for the matrix multiplication kernel, with grid configuration (64,64). In the asynchronous version, we have nStreams function calls, one call for each stream; subsequently, the grid configuration becomes (64/nStreams,64). Each function call computes a tile of C.

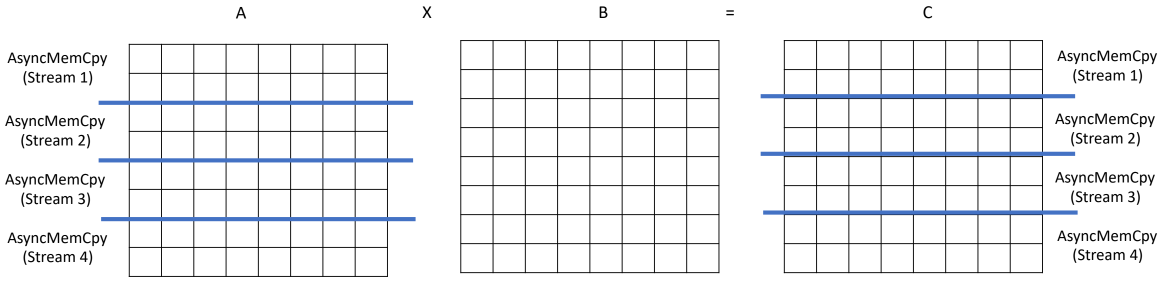

Figure 1: Example diagram of nStreams = 4

- Approach 1 (loop over {memory copy, kernel function, memory copy}):
  - Name this program as 'p1.cu'
  - The value of each element of $A[i][j]$ is i
  - The value of each element of $B[i][j]$ is j
  - Thread block configuration: 16×16

  - Grid configuration: (64/nStreams) × 64
  - After computation, print the value of $C[451][451]$

- Approach 2 (separate loops for memory copy, kernel function, and memory copy):
  - Name this program as 'p2.cu'
  - The value of each element of $A[i][j]$ is i
  - The value of each element of $B[i][j]$ is j
  - Thread block configuration: 16 × 16
  - Grid configuration: (64/nStreams) × 64
  - After computation, print the value of $C[451][451]$

- Report: try nStreams = 1, 4, and 16; measure the execution time of the kernel of Approach 1 and Approach 2, respectively. Briefly discuss your observations.

  Note: please use the cudaEvent API to obtain your program execution time (including memory copy and kernel execution). You can refer to "async.cu" to see how this is done.