Embedded System Design

Assignment 1

This homework corresponds to

- Attendances: Week 1 and 2 (2% of total score)
- Homework 1: (5% of total score)

What to submit: Report and source codes

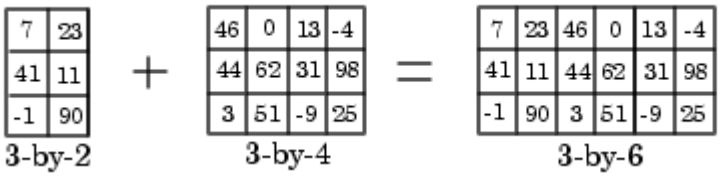
- There is no fixed report format, except the hard page limit, which is 5 pages.
- Zip your report and source code into a single file (your_student_num.zip, ex: 2015123456.zip) and submit it.

Deadline: Refer to I-Campus

Please read the following and implement source codes accordingly. You will implement source codes for CPUs and GPUs that perform matrix concatenation. Also you will compare the speedup of GPUs over CPUs with your source codes.

Concatenating Matrices

Matrix concatenation is the process of joining one or more matrices to make a new matrix. The below figure shows two matrices of the same height (i.e., same number of rows) being combined horizontally to form a new matrix.



You will implement source codes that concatenate two matrices. In this assignment, the sizes of two input matrices are as follows.

Input matrix 1: 512 by 1024 (# of rows: 512, # of columns: 1024)

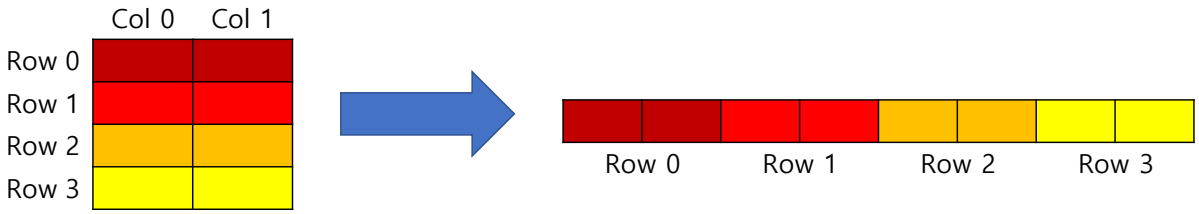
Input matrix 1: 512 by 512 (# of rows: 512, # of columns: 512)

All numbers in the matrices are integers (just use **`int`** type).

You may initialize all the elements in the input matrices as you want.

Here is what you need to do.

1. Implement the source code for CPU (C/C++)
   - Implement a single-threaded program. No need to implement a multi-threaded program with p-thread or OpenMP.
   - Add the code that measures the execution time.
   - For the matrices, simply use one-dimensional arrays. The sizes of the arrays would be 512*1024 (= 524,288) for input matrix 1, 512*512 (=262,144) for input matrix 2, and 512*1536 (= 786,432) for output matrix.
   - The following figure shows an example regarding how to map a matrix to an 1D array. This example maps a 4*2 matrix to a 1D array. Please apply this example to your implementation properly.



2. Implement the source code for GPU (CUDA)
   - Add the code that measures the execution time.
3. Measure the GPU execution time varying the number of threads in a thread block and the number of thread blocks.
   - Show all the execution time results with all the configurations that you analyzed.
   - Find the best configuration of # of threads and # of thread blocks for your CUDA code.
   - Find the best speedup of your CUDA program over the CPU program.