**CSUS COLLEGE OF ENGINEERING AND COMPUTER SCIENCE**
**Department of Computer Science**

**CSc 163 / 296P**
**Fall 2024 - Dr. Muyan**

# Assignment#1 (VectorAdd, BasicMatrixMultiplication, TiledMatrixMultiplication)

## Due Date: Monday, October 7th 11:59PM

Steps for solving the assignment (refer to “Assignment Environment” lecture slides for details):

- Download the assignment from Canvas and unzip it (now you have “A1” directory - you must have already done this to read this a1-instruction.pdf file).
- Build the source using CMake (in "build" directory that you create under "A1" directory).
- Copy datasets directly under “build” directory (under “A1/datasets” directory, we have three dataset directories, copy each of them to “A1/build”, e.g., copy contents of “A1/datasets/VectorAdd” to “A1/build/VectorAdd”).
- For each problem in the set:
    - Set the project properties as indicated in the below “Project Setup” instructions (e.g., set command line options).
    - Update the template code (template.cu) as indicated in the below “Coding” instructions to solve the problem.
    - Build your solution under Visual Studio (right click on the project and hit “Build”) and run it under Visual Studio (right click on the project and hit “Debug -> Start new instance”).
    - Make sure your executable (located under “Debug” directory that resides under “build” directory) works from the command line as indicated in the below “Command-line Execution” instructions.
    - Create a new directory by giving it the name of the problem.
    - Copy the executable file (.exe file) and the updated template code (template.cu) to this newly created directory. If the problem has questions, also copy the pdf file that includes your answers to these questions.

- Zip all newly created problem directories, name this ZIP file as YourLastName-YourFirstName-a#.zip (e.g., Doe-Jane-a1.zip).
- Build (using CMake and Visual Studio) and test your assignment in a **remotely accessible lab machine** (see the syllabus for tips) and **create a TEXT (i.e., not a pdf, doc etc.) file called "readme-a1.txt"** that indicates whether your submission works on the remotely accessible lab machine. You may also include additional information you want to share with the grader in this text file. **Make sure you include the executable files (.exe files) generated/tested on the lab machine in the zip file mentioned above.**
- Submit the **TWO files you have created above (zip file and txt file) SEPERATELY to Canvas** (i.e., do NOT place the txt file inside the zip file). You will receive the grader comments on your text file when grades are posted.

Each dataset directory provides **one or more test cases**. You should test your program with all the provided test cases by updating the command line options provided in the below "Project Setup" instructions. For instance, to test *test_case_1* of VectorAdd (located at "build\VectorAdd\Dataset\1") you should update the command line option provided below (which tests *test_case_0* located at "build\VectorAdd\Dataset\0") by changing all subtexts that say "…\Dataset\0\..." with "…\Dataset\1\...".

If you want to quickly check whether all datasets of all problems included in A#1 work from the command prompt, you can use the attached text file (A1_batch.txt) that lists the command-line execution lines for the datatsets excluding the last part from the line that directs the output to result.txt. This allows us to display all results on the console instead of writing them to result.txt files. Copy/paste the contents of these text files to command prompt when you are under "build" directory to see whether your code returns correct results for all datasets of all problems.

## Specific Problem Instructions:

## Problem#1: VectorAdd

### Objective

The purpose of this problem is to introduce the student to the CUDA API by implementing vector addition. The student will implement vector addition by writing the GPU kernel code as well as the associated host code.

## Coding

Edit the template.cu to perform the following:

- Allocate device memory
- Copy host memory to device
- Initialize thread block and kernel grid dimensions
- Invoke CUDA kernel
- Copy results from device to host
- Free device memory
- Write the CUDA kernel

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines in the template.cu.

## Project Setup

To test your program on test_case_0:

- Right click on the VectorAdd_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
-e .\VectorAdd\Dataset\0\output.raw

-i .\VectorAdd\Dataset\0\input0.raw,.\VectorAdd\Dataset\0\input1.raw

-o .\VectorAdd\Dataset\0\myoutput.raw -t vector

> .\VectorAdd\Dataset\0\result.txt
```

**(all in one line - do not forget to put spaces between the sub-lines)**

You will see the output of your execution in `result.txt` which resides under `build\VectorAdd\Dataset\0\`

## Command-line Execution

The executable generated as a result of compiling the project (`build\Debug\VectorAdd_Template.exe`) can be run from the command-line using the following command (make sure you are in `build` directory):

```
.\Debug\VectorAdd_Template -e .\VectorAdd\Dataset\0\output.raw

-i .\VectorAdd\Dataset\0\input0.raw,.\VectorAdd\Dataset\0\input1.raw

-o .\VectorAdd\Dataset\0\myoutput.raw -t vector

> .\VectorAdd\Dataset\0\result.txt
```

**(all in one line - do not forget to put spaces between the sub-lines)**

## Questions

(1) How many floating operations are being performed in your vector add kernel in terms of N, the size of the vector? explain.
(2) How many global memory reads are being performed by your kernel in terms of N, the size of the vector? explain.
(3) How many global memory writes are being performed by your kernel in terms of N, the size of the vector? explain.
(4) Name three applications of vector addition.

## <u>Problem#2: BasicMatrixMultiply</u>

## Objective

Implement a basic matrix multiplication routine. Optimizations such as tiling using shared memory should not be incorporated.

## Coding

Edit the code in the code tab to perform the following:

- set the dimensions of the product matrix
- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions

- invoke CUDA kernel
- copy results from device to host
- deallocate device memory
- write the CUDA kernel

Instructions about where to place each part of the code is demarcated by the //@@ comment lines.

## Project Setup

To test your program on test_case_0:

- Right click on the BasicMatrixMultiplication_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
-e .\BasicMatrixMultiplication\Dataset\0\output.raw

-i

.\BasicMatrixMultiplication\Dataset\0\input0.raw,.\BasicMatrixMultipli
cation\Dataset\0\input1.raw

-o .\BasicMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

> .\BasicMatrixMultiplication\Dataset\0\result.txt
```

**(all in one line - do not forget to put spaces between the sub-lines - above you see five sub-lines)**

You will see the output of your execution in `result.txt` which resides under

```
build\BasicMatrixMultiplication\Dataset\0\
```

## Command-line Execution

The executable generated as a result of compiling the project (`build\Debug\BasicMatrixMultiplication_Template.exe)` can be run from the command-line using the following command (make sure you are in `build` directory):

```
.\Debug\BasicMatrixMultiplication_Template

-e .\BasicMatrixMultiplication\Dataset\0\output.raw

-i

.\BasicMatrixMultiplication\Dataset\0\input0.raw,.\BasicMatrixMultipli
cation\Dataset\0\input1.raw

-o .\BasicMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

> .\BasicMatrixMultiplication\Dataset\0\result.txt
```

**(all in one line - do not forget to put spaces between the sub-lines - above you see six sub-lines)**

## Questions

(1) How many floating operations are being performed in your matrix multiply kernel in terms the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.
(2) How many global memory reads are being performed by your kernel in terms of the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.
(3) How many global memory writes are being performed by your kernel in terms of the following variables: *numCRows* and *numCColumns*? explain.

[Note: In the above questions, in addition to the listed variables, you can also use numbers (e.g., 2, 4, etc.) or constant values defined in the provided template code (e.g., TILE_WIDTH, etc.) in your answer].

(4) Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.
(5) Name three applications of matrix multiplication.

## Problem#3: TiledMatrixMultiply

### Objective

Implement a tiled matrix multiplication routine using shared memory.

### Instructions

Edit the code in the code tab to perform the following:

- set the dimensions of the product matrix
- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory
- write the CUDA kernel

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

### Project Setup

To test your program on test_case_0:

- Right click on the TiledMatrixMultiplication_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
-e .\TiledMatrixMultiplication\Dataset\0\output.raw

-i

.\TiledMatrixMultiplication\Dataset\0\input0.raw,.\TiledMatrixMultipli
cation\Dataset\0\input1.raw

-o .\TiledMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

> .\TiledMatrixMultiplication\Dataset\0\result.txt
```

**(all in one line - do not forget to put spaces between the sub-lines - above you see five sub-lines)**

You will see the output of your execution in `result.txt` which resides under

```
build\TiledMatrixMultiplication\Dataset\0\
```

## Command-line Execution

The executable generated as a result of compiling the project (`build\Debug\TiledMatrixMultiplication_Template.exe`) can be run from the command-line using the following command (make sure you are in `build` directory):

```
.\Debug\TiledMatrixMultiplication_Template

-e .\TiledMatrixMultiplication\Dataset\0\output.raw

-i

.\TiledMatrixMultiplication\Dataset\0\input0.raw,.\TiledMatrixMultipli
cation\Dataset\0\input1.raw

-o .\TiledMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

> .\TiledMatrixMultiplication\Dataset\0\result.txt
```

**(all in one line - do not forget to put spaces between the sub-lines - above you see six sub-lines)**

## Questions

(1) How many floating operations are being performed in your matrix multiply kernel in terms the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.

(2) How many global memory reads are being performed by your kernel in terms the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.

(3) How many global memory writes are being performed by your kernel in terms of the following variables: *numCRows* and *numCColumns*? explain.

[Note: In the above questions, in addition to the listed variables, you can also use numbers (e.g., 2, 4, etc.) or constant values defined in the provided template code (e.g., TILE_WIDTH, etc.) in your answer].

(4) Describe what further optimizations can be implemented to your kernel to achieve a performance speedup.

(5) Compare the implementation difficulty of this kernel compared to the *BasicMatrixMultiply* problem. What are the new code additions that programmers can make errors with this implementation?

(6) Suppose you have matrices with dimensions bigger than the max thread dimensions. Describe an approach that would perform matrix multiplication in this case.

(7) Suppose you have matrices that would not fit in global memory. Describe an approach that would perform matrix multiplication in this case.