CMPE 297 Lab#3

Due: Friday, Sep 9, 5:15pm
Total Score: /100

Instructor: Hyeran Jeon

Computer Engineering Department, San Jose State University

Group ID	// identical to your board id			
Member 1	Name		Student ID	
Member 2	Name		Student ID	

In this Lab, you will complete three versions of Matrix Multiply code by defining input/output matrices in different memories. Follow the steps below. Submit the completed code to Canvas before leaving the classroom.

Step 1: Download skeleton code

- Open web browser (*epiphany*), go to Canvas→Labs→Lab3 and download following file
  - lab3.tar.bz2
- Open *terminal* and type following commands
  - cd "*folder that you downloaded the files"*
  - tar –xvjf lab3.tar.bz2

Step 2: Copy the code to three different versions

- Copy cmpe297_matMul.cu to cmpe297_matMul_global.cu by using the following commands
  - cp cmpe297_matMul.cu cmpe297_matMul_global.cu

Step 3: Complete cmpe297_matMul_global.cu

- Open cmpe297_matMul_global.cu with *gedit* or *vi*
- In this code, input and output matrices will be stored in global memory. The kernel code in this version of code will be used as a baseline code for the shared memory and constant memory version code.
- Locations that you need to modify are marked with ***"// FILL HERE:"*** and ***"TODO"***.
  - MatrixMul function should be translated to a CUDA kernel

        - For better performance, use local variable for storing intermediate result of output element and then copy the final value to matrix C. Refer to the lecture slide entitled “Alternative Solution”.
    - GPU memory allocation for input/output matrices are already given.
    - Input matrices A and B should be copied from host to device. Use the memory copy API for global memory data. (The APIs can be found in the last lecture slide)
    - Set values of two integer variables, **blocksPerGrid** and **threadsPerBlock** so that you can run matrixMul kernel with ***1 block of WIDTH*WIDTH threads.*** WIDTH is predefined as 32 in the code.
    - MatrixMul function call should be changed to kernel call invocation statement
    - The code for copying output matrix C from device to host is already given.
- Refer to the lecture slide
    - To read the lecture slide in the Jetson board, you should download pdf version lecture slide

Step 4: Compile cmpe297_matMul_global.cu

- Type following command to compile cmpe297_matMul_global.cu
    - make global
- When compiling the code, you will see a few lines of message. The messages are about resource usage of the code. Check how much space is used in each memory by this code and fill TABLE 1. If the specified indicator is not shown in the message, assume that the code doesn’t use the memory and fill 0 in the table.

Step 5: Run matrixMul_global

- Type following command to run matrixMul_global
    - ./matrixMul_global
- If the following sentence appears, you are done!
    - “Test PASSED”

Step 6: Complete Shared Memory version code

- Copy cmpe297_matMul_global.cu to cmpe297_matMul_shared.cu
- Open cmpe297_matMul_shared.cu with gedit or vi
- Locations that you need to modify are marked with ***“// FILL HERE:”*** and ***“TODO”***.
    - Modify MatrixMul kernel to store and read matrices to/from shared memory.
- Repeat Step 4 and 5 but using “shared” instead of “global” like below
    - make shared

- ./matrixMul_shared

Step 7: Complete Constant Memory version code

- Copy cmpe297_matMul_global.cu to cmpe297_matMul_const.cu
- Open cmpe297_matMul_const.cu with gedit or vi
- Locations that you need to modify are marked with ***"// FILL HERE:"*** and ***"TODO"***.
  - Define matrix A in constant memory. Use name ***d_A***.
  - Modify MatrixMul kernel to read matrix A from constant memory.
  - Delete d_A allocation code in the host side main function.
  - Modify the data copy API for d_A from host to device.
  - Modify MatrixMul kernel function call accordingly.
  - Delete cudaFree operation on d_A.
- Repeat Step 4 and 5 but using "const" instead of "global" like below
  - make const
  - ./matrixMul_const

Step 8: Check and compare performance of the three code

- In all three code files,
  - Add following argument as MatrixMul function's input parameter
    - `unsigned long long* runtime (i.e. MatrixMul (float* A, float* B, float* C, unsigned long long* runtime))`
  - Add the following line in the beginning of MatrixMul function
    - `Unsigned long long start_time = clock64();`
  - Add the following two lines in the end of MatrixMul function
    - `unsigned long long stop_time = clock64();`
    - `runtime[tid] = (unsigned long long)(stop_time - start_time);`
  - In the kernel calling statement, pass following variable to MatrixMul function as an input parameter
    - `d_runtime (i.e. MatrixMul<<<.., ..>> (d_A, d_B, d_C, d_runtime);)`
- Compile all three code again by giving the following commands
  - make clean; make def=TM
- Run each of the executables and check execution time. Fill the execution time of each code to TABLE 2.

Step 9: Submit the completed three code files to Canvas and this sheet to TA

– One copy per group

TABLE 1

Code	**Const mem (bytes) (marked as “cmem[3]”)**	**Shared mem (bytes) (marked as “smem”)**
Global memory version		
Constant memory version		
Shared memory version		

TABLE 2

Code	**Execution time (cycles)**
Global memory version	
Constant memory version	
Shared memory version	