CMPE 214 HW#4

Due: Tue, May 1, 11:59pm
Total Score: /100

Instructor: Hyeran Jeon

Computer Engineering Department, San Jose State University

Homework is not a group work. Each student should submit his/her own homework solution.

In this Homework, you will modify Matrix Multiply code developed in Lab3 to evaluate the performance of cuBLAS library.

cuBLAS library: So far, we have learned how to implement an optimized kernel functions in CUDA. Now it's time to understand how to use library functions. There are several CUDA libraries such as cuBLAS, cuDNN, cuSPARSE, cuRAND, cuSOLVER, cuFFT, CUDA Math library and many more. These libraries provide precompiled kernel functions that are highly optimized. When you develop a large product-level code that needs to be accelerated on CUDA architecture, it would be better considering using these libraries instead of implementing individual kernel functions by yourself. Out of many libraries, cuBLAS library supports basic linear algebra subroutines (BLAS) such as max, min, sum, dot, normalized by 2, scalar multiplication, swap, vector-matrix multiplication, matrix-matrix multiplication and many more.

Data layout: cuBLAS library uses column-major storage, which means that the elements in the same column of a matrix are stored in the consecutive memory addresses as illustrated in Figure 1. In Figure 1, the value 1 on (0,0) is in the lowest memory address and the value 5 on (0,1) is in the memory address that has 4-element distance from the first element. In C, as value 1 and 5 are consecutive in the same row, they are stored in the consecutive memory addresses. Thus, the data index of multi-dimensional data structure should be carefully calculated. The macro explored

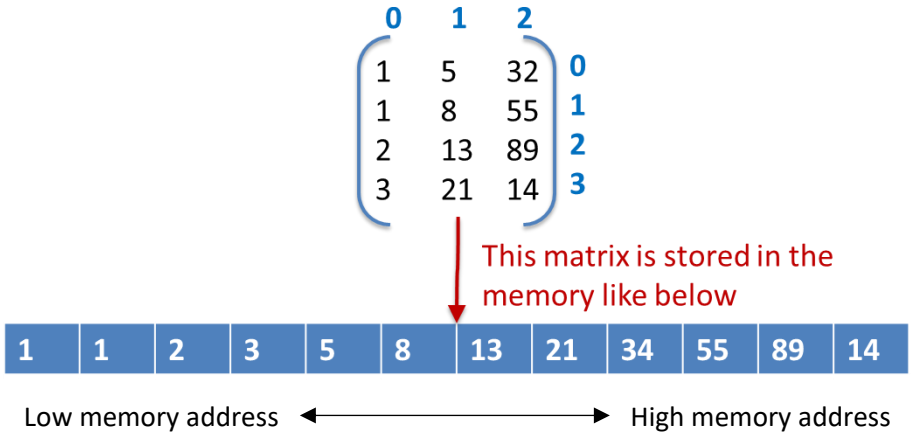

Figure 1. Matrix Column-major ordering

in the lecture like below can be used for the index calculation.

```
#define IDX2C (i, j, ld) j * ld + i
```

This macro receives the row-major index information such as the row number, i, and the column number, j, and the dimension width, ld, and calculates the associated column-major index. This macro can be used for filling elements to a multi-dimensional structure as shown on page 9 of lecture slide 10 and pointing to a specific element on page 10 of the same slide. Some library functions also provide transpose operations on the input data as one of the input parameters. If the input parameter is set, the input multi-dimensional data structure will be transposed to column-major order. Thus, the data doesn't need to be filled in column-major order before running the library function.

Library handle: To run cuBLAS library functions, a handle should be created like below. All the cuBLAS functions executed by the same CPU process should be linked to the same handle.

```
cublasHandle_t handle;

err = cublasCreate(&handle);
```

The handle should be passed to the cuBLAS functions as the first input parameter like below.

```
cublasSscal (handle, M-1, 12.0f, &devPtrA, M);
```

Requirements and guidelines:

In this homework, we are going to modify global memory version matrix multiply code to run cuBLAS library function.

Details are like below:

1. Copy the global memory version Lab3 code and use it as a baseline. The original Lab3 code will be used for comparing the performance so should not be deleted.
2. Change WIDTH value as 16 (16x16 matrix multiplication).
3. Change the input matrix filling code to like below:

```
for (int i = 0; i < WIDTH; ++i)
 {
     for (int j = 0; j < WIDTH; ++j)
     {
         h_A[i*WIDTH + j] = j/10;
         h_B[i*WIDTH + j] = i/10;
     }
 }
```

4. Go to cuBLAS user guide page https://docs.nvidia.com/cuda/cublas/index.html and choose the function that runs matrix multiply on single-precision floating-point data.
5. Replace the kernel invocation line of Lab3 code with the library function call. You can delete the existing kernel function.
6. Delete the existing timer related code and put event timer that you used for HW3 and measure the time of library function call (do not include printf or parameter setting code lines in the measurement; we want to compare the actual function call time only).

7. Modify the Makefile to use -lcublas linker option.
8. The code should generate a correct output; this will be checked by seeing the output message “Test PASSED” as in Lab3. To pass the test, you may need to check the data layout carefully as discussed above.
9. Once your code is correctly working, change the WIDTH value to the values below and plot the execution times of the kernel version and library version in one graph. Analyze the graph (i.e. library function outperforms when the data size become X or kernel version doesn’t support if the WIDTH becomes larger than Y). You don’t need to rewrite the existing Lab3 code except for WIDTH modification. Show the graph and the analysis in one document. For better analysis, use the grid, block, registers, and shared memory usage information that you can acquire from `nvprof` with `--print-gpu-trace` option.
    a. 8
    b. 16
    c. 32
    d. 64
    e. 128

10. Submit the code and the document to Canvas before the deadline.