CS4990 Spring 2024 Homework 2

Total points: 100

Due date: Friday, April 12, 2024

Task Description:

In this assignment, you are tasked with developing a complete CUDA C program for matrix-matrix multiplication, $C = AB$, which includes a new optimized kernel that implements a "tiled" version of matrix multiplication using dynamically allocated shared memory space:

1. **Develop** a single program file named "sgemm2.cu" containing all necessary code to compute matrix multiplication $C = AB$, with the following command-line arguments for compiling and execution.
    a. To compile: nvcc -o sgemm2 sgemm2.cu
    b. To execute**: ./ sgemm2 <m> <k> <n>**

    where m, k, and n specify the dimensions of matrices A, B, and C. Specifically, matrix A is of size $m \times k$, matrix B is of size $k \times n$, and matrix C is of size $m \times n$. The program fills each element of matrix A and matrix B with random float-point values generated using "rand()%100/100.0".

2. Within "sgemm2.cu", **implement** one host function and two CUDA kernels to perform matrix multiplication, respectively. Specifically,
    a. A host function exclusively handling matrix multiplication using CPU-only computation, which is the same as the function you implemented for part 2.1 in Homework 1.
    b. A CUDA kernel where each thread calculates one element of the output matrix, which is the same as the kernel you implemented for part 2.b in Homework 1.
    c. **A new optimized CUDA kernel** that presents a "tiled" version of matrix multiplication by using dynamically allocated space in shared memory. Here we assume each thread calculates one element of the output matrix.

    Note that regarding the host function for part 2.a and the CUDA kernel for part 2.b of Homework 2, you can simply re-use your implemented function and kernel of part 2.a and 2.b in Homework 1.

3. **Ensure** that when calling the new kernel of Part 2.c, your code can **dynamically configure the amount of shared memory** to be used for each block **according to the device query result** and **supply that as a third parameter to the kernel call**.

4. **Ensure** that your code accommodates **varying input dimensions**, for instance, m = 1234, k = 1567, and n = 1890. Please also **consider** cases where the matrix dimensions may not be divisible by your chosen block size – so don't forget to pay attention to boundary conditions to ensure proper handling in such cases.

5. **Utilize** timing techniques such as CPU timers or CUDA events, **to measure the performance of** your implementation of the host function and two CUDA kernels as specified above.

We also suggest **structuring** the "sgemm2.cu" **by implementing the following macros, host functions, and CUDA kernels**:

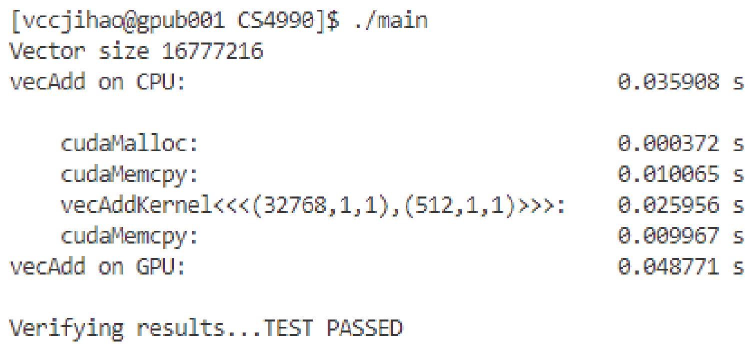
- #define CHECK(call)
    - A macro for error checking.
- myCPUTimer()
    - A timer for measuring execution time.
- basicSgemm_h(int m, int k, int n, const float *A_h, const float *B_h, float* C_h)
    - A host function for CPU-only matrix multiplication.
- __global__ void matrixMulKernel_1thread1element (int m, int k, int n, const float *A_d, const float *B_d, float* C_d)
    - A CUDA kernel where each thread computes one output matrix element.
- __global__ void **matrixMulKernel_tiled**(int m, int k, int n, const float *A_d, const float *B_d, float* C_d, unsigned Adz_sz, unsigned Bdz_sz)
    - A CUDA kernel where a "tiled" version of matrix multiplication is presented, which uses dynamically allocated space in shared memory. Here we assume each thread calculates one element of the output matrix.
- void basicSgemm_d_1thread1element (int m, int k, int n, const float *A_h, const float *B_h, float* C_h)
    - A host function for handling device memory allocation and free, data copy, and calling the specific CUDA kernel, matrixMulKernel_1thread1element().
- void **basicSgemm_d_tiled** (int m, int k, int n, const float *A_h, const float *B_h, float* C_h)
    - A host function for handling device memory allocation and copy, device query, and dynamically configuring the amount of shared memory and calling the specific CUDA kernel, matrixMulKernel_tiled().
- int main(int argc, char** argv)
    - The main entry point of the program
- bool verify(float* CPU_Answer, float* GPU_Answer, unsigned int nRows, unsigned int nCols)
    - A function to validate if the computed matrix C using a CUDA kernel matches that of the CPU-only function.
    - Note that you may call this verification function two times in your main function, in order to
        - Compare the matrix-multiplication result of basicSgemm_h() with that of basicSgemm_d_1thread1element().
        - Compare the matrix-multiplication result of basicSgemm_h() with that of basicSgemm_d_tiled().

What to Submit?

Please **compress** the following two required files into a zip file, named following the format "yourname_p2.zip", and **submit** the zipped file on Canvas.

- A single **"sgemm2.cu"** program file containing all required code.
- An **"output.jpg"** file presenting a screenshot of the verification results of the "verify" function and the timing results.

To demonstrate the necessary information for your screenshot, here's an example of a screenshot displaying the output of my vector-addition program. It includes the verification results and the timing results. However, please note that this example is not for matrix multiplication.



```
[vccjihao@gpub001 CS4990]$ ./main
Vector size 16777216
vecAdd on CPU:                                  0.035908 s

    cudaMalloc:                                 0.000372 s
    cudaMemcpy:                                 0.010065 s
    vecAddKernel<<<(32768,1,1),(512,1,1)>>>:    0.025956 s
    cudaMemcpy:                                 0.009967 s
vecAdd on GPU:                                  0.048771 s

Verifying results...TEST PASSED
```