Department of Electrical and Computer Engineering
Queen's University

ELEC 374, Digital Systems Engineering

Machine Problem 3

Total Mark: 3%

Posted: Feb 16, 2023
Actual Deadline: **Mar 31, 2023** at 8:30am
3 days Grace Period: **Apr 3, 2023** at 8:30am
After grace period: **25% late penalty per day**

Matrix Multiplication

The objective of this machine problem is to implement a dense matrix multiplication routine with different number of blocks and threads per block. It is also to understand the impact of data transfer time on performance.

A matrix multiplication takes two input matrices M and N and produces an output matrix P. For simplicity, we will only handle square matrices, of which the elements are single-precision floating-point numbers. Write a matrix multiplication kernel and the host function that can be called with four parameters: pointer to the output matrix P, pointer to the first input matrix M, pointer to the second input matrix N, and the number of elements in each dimension. After the device matrix multiplication is invoked, it will compute the correct output matrix using the CPU and compare that solution with the device-computed solution. If it matches (within a certain tolerance), it will display "Test PASSED" on the screen before exiting.

Create randomly initialized matrices M and N. Write the host code by allocating memory for the input and output matrices, transferring input data to device, launching the kernel, transferring the output data to host, and freeing the device memory for the input and output data. Write a kernel that has each thread producing one output matrix element. Write the execution configuration parameters in your host code accordingly.

In this machine problem, we also want to understand the impact of data transfer from the host to the device and from the device to the host on the overall performance, and whether kernel offloading to GPU is beneficial for all cases.

Follow the instructions below:

1. Find the time it takes to transfer the two input matrices from the host to the device. Experiment with different matrix sizes (125 x 125, 250 x 250, 500 x 500, 1000 x 1000, and 2000 x 2000), and plot the data transfer time vs. matrix size. Repeat your experiments, but this time transfer these two matrices back from the device to the host. Plot the data transfer time vs. matrix size in this case as well. Do you see any performance difference between the host-to-device and device-to-host data transfers, and why?

2. Using the same matrix sizes that you experimented with in Part 1, compare the time it takes to do the matrix multiplication on the GPU with that of the CPU. For the GPU computation, just consider a single block and one thread for the block. Do not consider the data transfer time and ignore memory allocation/free time on the device - just the matrix multiplication time on the GPU and CPU. Now, if you take the total data transfer time into account (both ways, from host to device and from device to host), is it always beneficial to offload your matrix multiplication to the device?
3. In this part, we experiment with changing the number of blocks and the number of threads per block to see their impact on the GPU performance. Find the matrix multiplication kernel time for different matrix sizes as in Part 1 with a block width of 2, 4, 10, 20, and 25, respectively. Do not consider the data transfer time and memory allocation/free time on the device in your timing. Plot the kernel execution time vs. numblocks/block-width and discuss your findings. Also, answer the following questions:

    a) How many times is each element of each input matrix loaded during the execution of the kernel?
    b) What is the floating-point computation to memory-access (CGMA) ratio in each thread? Consider multiply and addition as separate operations and ignore the global memory store at the end. Only count global memory loads towards your off-chip bandwidth.

For this and other machine problems, you may consult the Lecture Slides on GPU Architectures and Computing and the GPU Server and CUDA Environment Tutorial on the course website. You may also consult the NVIDIA CUDA C Programming Guide. As for the timing, you should create CUDA events for the start and stop times, begin and stop recording, find the elapsed time, and destroy the events (*cudaEventCreate, cudaEventRecord*, *cudaEventElapsedTime*, *cudaEventSynchronize*, etc.). For timing measurements, make sure you repeat your experiments a sufficient number of times, remove any outliers due to the shared environment, and report the average time.

Submission Instructions:

This machine problem is to be done individually. Students must run through the assignment by themselves, include their own solutions and analyses, and turn in their own reports. Submit a **single zip file**, "MP3_YourLastName.zip", containing your code (e.g., MP3.cu, with your name and student ID on top of the code), and a report in pdf, MP3_YourLastName.pdf, where you present your work by including your CUDA code and the Visual Studio output screenshots, analyses of your results, answers to the questions, and discussions of any outstanding issues. Students must abide by the academic integrity expectations, and include in their report a statement that says:

- *"I do hereby verify that this machine problem submission is my own work and contains my own original ideas, concepts, and designs. No portion of this report or code has been copied in whole or in part from another source, with the possible exception of properly referenced material.".*

GPU Servers:

Four GPU (Tesla C2075) servers are available for the machine problems. The servers run Windows server operating system and provide the Visual Studio 2015 IDE for building CUDA projects. For remote connection to the GPU servers and information on CUDA and its environment, please consult the GPU Server and CUDA Environment Tutorial. The servers use a load balancer to balance the workload on the GPU servers, and it is possible that you may log in to a different GPU server each time. Therefore, it is very important to back up your files.