GPU Programming Assignment 5: DPC++

25 July 2022

Problem Statement

The Intel DevCloud is a development sandbox to learn about programming cross-architecture applications with OpenVino, High-Level Design (HLD) tools – oneAPI, OpenCL, HLS and RTL. DevCloud offers different devices on which you can run your workload, including CPUs, GPUs and FPGAs. Find the complete list of all the devices offered by DevCloud here.
Intel provides the DPC++ compiler as a part of the oneAPI base toolkit. The above toolkit is available and ready to use on Devcloud. In this assignment, you will be writing and executing DPC++ codes on Devcloud. This assignment is divided into three parts. The first consists of setting up your DevCloud account, and the second and third parts require writing DPC++ codes.

Setting up DevCloud

1. Create an account on Devcloud: https://devcloud.intel.com/oneapi/get_started/

2. Go to the link in step 1 and sign in. Click on the link "Launch JupyterLab" at the bottom of the page to open the jupyter lab interface.

3. Click the + button and open a terminal. You will observe that the terminal prompt reads: $(u < userid > @sxxx - nyyy)$. $sxxx - nyyy$ specifies the compute node which you are currently using. Each compute node has a specific CPU associated with it. The nodes might or might not have an accelerator associated with them. To see a complete list of nodes available use the command: pbsnodes

4. You can compile your dpcpp code using: dpcpp $< file >$. You will need to submit a bash script to a compute node of your choice to execute your

code. Include the line "source /opt/intel/oneapi/setvars.sh" at the top of your bash script. Then include the appropriate commands to compile and run your code.

5. DPC++ codes will be executed by submitting the bash script to a queue using qsub (Don't confuse this queue with the queue objects you would have studied in the DPC++ programming model). Submit the job through the command: qsub -l nodes=1:gpu:ppn=2 script.sh. For more details about the job submission, you can visit the link: Job Submission (Note that the above command will submit your job to compute node which has a GPU associated with it)

Converting CUDA Code to DPC++ (60 points)

Examine the CUDA code present in the file cuda_code.cu. In this section you will be working on translating the given CUDA code to DPC++ and analysing it's performance. You would have learned about two data management methods in the DPC++ - buffers and USM. Your first task is to implement the given CUDA code using the USM API. There are three flavors of the USM API which you can try out here:

- using a combination of malloc_device and malloc_host **(15 points)**
- using only malloc_shared **(3 points)**
- using only malloc_host **(2 points)**

Measure the average execution time of each of the three codes above for multiplying matrices of Test case 6, 7 and 8 using the Chrono library present in C++. You only need to measure the kernel duration. The average should be measured over at least 100 iterations. **(10 points)**

Which one of them is performing the best? Can you think of the reason for the difference in their performance? **(10 points)**

The second data management method offered by DPC++ is the buffers API. Implement the given CUDA code in DPC++ using the buffers API **(10 points)**

One of the promises of DPC++ is to offer a single API for programming both CPUs and GPUs. You are currently executing your code on the GPU. Execute

the code on a CPU as well.Executing the code on CPU might take too much time so, limit the execution of codes on CPU from Test case 1 to Test case 5.**(2 points)**

Compare the execution time on the CPU and GPU and report which one is faster and the speedup obtained **(8 points)**

Shared Memory Tiled Matrix Multiplication (30 points)

Tiled Matrix multiplication is a common method to improve the performance of matrix multiplication on GPU by utilising shared memory. Go through the blog here explaining Shared memory based tiled matrix multiplication: https://penny-xu.github.io/blog/tiled-matrix-multiplication Based on your experience with the previous question, choose a data management method and implement a shared memory based tiled matrix multiplication in DPC++. **(20 points)**

Measure the execution time for each of the test cases and report if you observe an improvement in the performance. If you observe an improvement what is the speed up obtained? **(10 points)**

Note: Make sure your codes at least pass the sample test cases provided in the assignment folder before you proceed to any kind of timing collection. While measuring timings you can ignore the execution time of your first kernel launch because that can take exceptionally longer time as compared to consecutive kernel launches.

Input Format

The test cases are present in the folder TestCases in the assignment folder. The first line of each test case contains 3 numbers representing matrix sizes m, n, k. We are multiplying matrix A(mxk) with matrix B(kxn). The first line is followed by m*k + k*n numbers representing matrices A and B respectively.

Correctness

You only need to run the script check_correctness.sh to evaluate the correctness of your DPC++ codes on the given test cases. Provide the DPC++ file name as a command line argument to the script. Your code should take input from

stdin and write output to stdout. Don't write anything apart from the final resultant matrix in the output while evaluating correctness of your code using the script.

Submission

You are required to submit 4 DPC++ codes from the section titled converting CUDA code to DPC++. You also need to submit 1 DPC++ code implementing shared memory based tiled matrix multiplication. You are also required to submit a report answering all the questions stated in the above sections.