HPCA Programming Assignment 2022

Optimizing performance of Reducing Matrix Multiplication

Due: 30th November 2022, 11:59 pm

In this assignment, your task is to implement and optimize the "Reducing Matrix Multiplication (RMM)" of two n*n matrices. A sample single-threaded *unoptimized* implementation is also provided via GitLab — check the bottom of this document for the same. You will be required to optimize the single threaded implementation, provide and optimize a multi-threaded implementation, and finally accelerate the program on a GPU.

Input: Two n*n matrices. n = 2^k, where k is a natural number and k >= 2.

Output: A matrix of size (n/2) * (n/2)

You will have to work on the programming assignment **in teams of at most two.** The assignment should be submitted on gradescope. Please start early.

Explanation of RMM:

Reducing Matrix Multiplication is a variant of matrix multiplication.

In vanilla matrix multiplication, each entry of the output matrix, C[i][j] is determined as the dot product of the row i of matrix A and column j of matrix B.

In RMM, each entry of the output matrix C[i][j] is determined by two rows (rows (i*2) and (i*2+1)) and two columns ((j*2) and (j*2+1)). The value of C[i][j] is determined as the **sum** of the following dot products:

1. `Row [i*2] of A   * column [j*2] of B`
2. `Row [i*2] of A   * column [j*2+1] of B`
3. `Row [i*2+1] of A * column [j*2] of B`
4. `Row [i*2+1] of A * column [j*2+1] of B`

Notice that the output matrix C has size (n/2) x (n/2).

You can find the code for the above in the Gitlab repository provided.

Objectives

1. Optimized single threaded implementation of RMM on the CPU. You will hand optimize the single threaded implantation of RMM. You may use hardware performance counters (e.g., perf) to guide you. Relevant files: PartA/header/single_thread.h

2. Implement and optimize multi-threaded version of RMM on CPU. You **must** use "pthreads" as your threading library. Do not use any other libraries. Test and report the scalability of your implementation. Relevant files: PartA/header/single_thread.h

3. Implement and optimize GPU (CUDA) version of RMM on GPU. You **must** use CUDA as your GPU library. Do not use any other libraries (OpenCL). You must try to maximize the utilization of GPU resources (i.e., please do not copy the CPU version and run with a single thread on the GPU). Relevant files: PartB/header/gpu_thread.h

You may consult *relevant* documentation from the Internet, but you are **not** allowed to use code from the Internet. Any instance of plagiarism will be severely punished.

The repository has a README file with additional details on compiling, generating inputs, running, and modifying the code. Please check the README for more details.

Deliverables

Submit the assignment as a **tar.gz** file, with the following format: YourName-SrNo.tar.gz. The archive file (tar.gz file) must contain the *PartA/header* and *PartB/header* folders where you have implemented your optimizations, and a folder names *reports* which must contain your reports. Please write detailed reports with graphs. Include details such as runtime of unoptimized and optimized code, bottlenecks observed, the rationale for your optimizations, the effect of your optimizations on the observed bottlenecks, scalability of your multi-threaded code etc. Please note the above is not an exhaustive list and use your discretion to provide more relevant details and numbers.

Submission instructions will be posted separately on Teams, so please keep checking Teams every once in a while for updates.

Where to run/test your CUDA program:

You may access GPUs in the department server to run your program. However, expect the GPUs to be very busy during the end of the semester. Therefore, we recommend using GPGPU- sim (see below for installation instructions). This is a software that runs on the CPU but emulates a GPU. It runs CUDA application out-of-the-box. You can install this simulator in your local machine and do your homework.

Prerequisites:

In order to compile CUDA code (whether for a GPU or GPGPU-Sim), you will need to install the CUDA compiler (nvcc). You can download and install this from the official NVIDIA website (latest version is 11.8, but GPGPU-Sim may not work with the latest version). You do NOT need to install the driver if you plan to use GPGPU-sim. GPGPU-Sim currently works with CUDA versions 4.5 - 11.0.

Instructions for using GPGPU-sim:

- Clone GPGPU-sim using the following link. https://github.com/gpgpu-sim/gpgpu-sim_distribution

- Run your code for the SM7_TITANV configuration. You can find the configuration file at : gpgpu-sim_distribution/configs/tested-cfgs/

- You can use GPGPU-sim profiler to optimize the performance your CUDA application. If you are running on the GPU hardware, look for NSight and/or NVprof tools.

Gitlab

https://gitlab.com/bpratheek/hpca-course-assignment-2022.git

Questions?

Please ask any questions on Teams.