COL380 - Assignment 4

Parallel Matrix Multiplication using CUDA

Due: 11:59 pm, 28 April, 2023

1 Introduction

You will use CUDA-C++ to create a massively parallel application for CUDA-capable GPU. Here are some resources on CUDA-C++:

- Chapters 3 to 6: Programming Massively Parallel Processors by D. Kirk and W. Hwu
- NVIDIA technical blog: An Even Easier Introduction to CUDA
- NVIDIA slides: CUDA Basics
- NVIDIA slides: CUDA Optimizations
- NVIDIA slides: CUDA Streams

2 Problem Statement

Recall matrix multiplication: given a matrix A of size $(p \times q)$ and a matrix B of size $(q \times r)$, the output matrix, $C = AB$, $C_{ij} = \sum_{k=0}^{r-1} A_{ik} \times B_{kj}$ where C_{ij} indexes the element in i^{th} row and j^{th} column of C.
Your goal is to calculate C' where $C'_{ij} = max(C_{ij}, MAX\_VAL)$ and $MAX\_VAL = 2^{32} - 1$ (4 bytes). You are supposed to parallelize this using CUDA.

3 Suggestions

A few heads ups before using CUDA:

- Rather than using your CPU, you are sending computation over to a different device. Often times, the bottleneck can be this transfer of data itself.
- CUDA has multiple types of memories (global, constant, shared, registers). Take into account their individual behavior and decide which one you should adopt. Your decision will also dictate the optimal way threads should read data.
- Read about coalescing global memory and bank conflicts in shared memory to reduce data traffic.
- Read about the relation between the block size and SM specifications. The grid and block dimensions also play a role.
- Read up on CUDA streams to parallelize data transfer and computation or to overlap kernels.
- You can use any of these profilers to get some intuition: NVPROF, NVVP, NSight.

These are all just suggestions. You need not incorporate all of them (nor are you limited to them)! We advise that you start simple. Then iterate over different optimizations. Always have a working implementation kept aside in case something goes wrong. Some iterations may involve major revisions, some may not. Another important thing is to realize when to stop. There may be some bottlenecks that are inherent to the program and can't be done away with.

4 Input

The input format for this assignment is similar to that for A1, except that the matrices need not be symmetric. Formally, assume that A_{n*n} is subdivided into blocks of $m * m$ elements. Only the blocks with at least one non-zero element are given in the input file. The other blocks are assumed to have all 0s. For example, we can divide a $100 * 100$ square matrix into $5 * 5$ element blocks, i.e. 400 total blocks, which themselves have a $20 * 20$ configuration, allowing us to index blocks as (i, j).

The input file is a little-endian binary file containing the square matrix in the following format:

1. First 4 bytes represent n (dimension of the matrix is $n \times n$)
2. next 4 bytes represent m (dimension of the block is $m \times m$)
3. next 4 bytes represent numbers of 'non-0' blocks given as input(say k)
4. now, each block is represented as
   (a) First 4 bytes represent the block's i^{th} index of the block (from top to bottom)
   (b) next 4 bytes represent the j^{th} index of the block (from left to right)
   (c) next $2m^2$ bytes represent the block in a row-major order, where 2 bytes correspond to the 1 element(cell) of the block.

Note:-

1. The 4 bytes representation of a number has the most significant byte at the end.
2. Please note that these blocks are meant only for matrix formatting. The CUDA-block sizes are to be independently decided by you and can be totally independent of m.
3. For this assignment, assume m can be either 4 or 8
4. n can have a value upto 2^{15}
5. When m = 4, k would have a maximum value of $3.3 * 10^7$
6. When m = 8, k would have a maximum value of $8.8 * 10^6$

You will be provided scripts to generate matrices and cross check your answers. We will also share some shared inputs to save space on the css machines.

5 Output

Output the resultant matrix in a binary file using the following format. Only the blocks with at least one non-zero element must be output.

1. First 4 bytes represent n (dimension of the matrix). This should be the same as the input file.
2. next 4 bytes represent m (dimension of the block). This should be the same as the input file.

3. next 4 bytes represent numbers of blocks(say k). The block indexes should be sorted with the first index value i, and then the second index value j.

4. now, each block is represented as

   (a) First 4 bytes represent the i^{th} index of the block

   (b) next 4 bytes represent the j^{th} index of the block

   (c) next $4 * m^2$ bytes represent the block where 4 bytes corresponds to the 1 element(cell) of the block. (Recall, the stored value $b' < 2^{32} - 1$.)

6 Analysis

Submit a report where you explain the design choices of your algorithm. Describe in detail the approach you took for each optimisation and include your observations.

7 COL380 vs COV880

You must incorporate CUDA shared memory in your algorithm.

8 Submission Instructions

We will run your code on HPC with K40 GPUs. Therefore, you must ensure that it runs on them. Use the **-arch** and **-mach** flags to achieve this. We will test your code on matrices of different sizes. Your algorithm should be robust to matrix size.
Name your report **report.pdf** and the main source file as **main.cu**. All kernel calls should be made from **main.cu**. Submit a zip file (<kerberos>.zip) that contains the following directory structure:

```
kerberos/
|-- Makefile
|-- report.pdf
|-- main.cu
|-- other code files
```

For example, for a student with kerberos cs5180442, the zip file should be named cs5180442.zip. Running unzip cs5180442.zip should create a folder named cs5180442 with the following contents:

```
cs5180442/
|-- Makefile
|-- report.pdf
|-- main.cu
|-- other code files
```

The following commands need to be executed to run your program:

```
make
```

This should create an executable (named **exec**) for your program. After this, we should be able to run the executable with proper command line arguments using the following command:

```
./exec inputFile.bin outFile.bin
```

This should dump your output into the **outFile.bin** file.
Note that there would be a 25% penalty for not following the submission instructions.