Take-Home Exam CUDA_SCAN

Parallel Programming & Architectures

Consideration

Your code is automatically graded using a script, and therefore, if your file/folder names are wrong you will receive a grade of **zero**. Please read and follow the instructions carefully. Common mistakes include

- Different file or folder names
- Different formatting of input or output
- Not paying attention to case sensitiveness of C++ and Linux

Go to the folder ~/the/cuda_scan/ in your home directory on the server and put your codes in this directory and remove any compiled binaries and test cases.

Make sure your code compiles and runs without any error **on the server**. Your grade will be **zero** if any compile or runtime error occurs on the server. **Any!**

The provided test cases, examples and sample codes (if any) are only to better describe the question. They are **not** meant for debugging or grading. It is your responsibility to think of and generate larger and more complex test cases (if necessary) in order to make sure your software works correctly for all possible scenarios.

Start early and don't leave everything to the last minute. Software debugging needs focus and normally takes time.

Just leave your final programs on the server. **Don't** email anything!

Your grade is divided into several parts. In all cases, if you miss **correctness** (i.e. your code doesn't satisfy desired functionality), you miss other parts (e.g. speed, coding style, etc.) too. This rule is applied separately for each section of a take-home exam. So for example, in cuda_mm, your code might not be correct for M >= x but still you will get your grade for lower M values.

Talking to your friends and classmates about this take-home exam and sharing ideas are *OK*. Searching the Internet, books and other sources for any code is also *OK*. However, copying another code is **not OK** and will be automatically detected using a similarity check software. In such a case, grades of the copied parts are **multiplied by -0.5**. Your work must be 100% done only by yourself, and you **should not** share parts of your code with others or use parts of other's codes. Online resources and solutions from previous years are part of the database which is used by the similarity check software.

Grade: 20% correctness, 80% speed

1 Introduction

In this exercise, there is a problem similar to the scan problem that its algorithm should be implemented on the GPU. In this problem, Galois fields are used. For this reason, we introduce these fields below and then state the problem.

2 Galois Fields

A "field" is a set containing a number of elements. Addition and multiplication are defined among the members of this field. Addition and multiplication in the field are defined differently from regular addition and multiplication. In the problem that will be discussed here, each member of this field is represented by an 8-bit number and is stored in a variable of type uint8_t.

The sum of two variables like a and b is equal to the xor operation of their corresponding bits. In other words, a^b represents the sum of two elements.

Example:

a=187, b=84 → a+b=239

The multiplication of two elements a and b is such that if either of them is equal to zero, the result of the multiplication is zero. Otherwise, it is obtained as follows:

alpha_to[(uint32_t(index_of[a]) +uint32_t(index_of[b])) % 255]

In the above expression, “alpha_to” and “index_of” are two arrays of length 256 and of type uint8_t. These two arrays are given to you.

Note: In the mentioned multiplication algorithm, you do not need index_of[0]. However, in the array provided to you, we have set the value of index_of[0] to 255, and the only index i for which index_of[i] is equal to 255 is zero.

Based on the definition of addition and multiplication, it is possible to define matrix multiplication as well.

3 problem description

A set of vectors of size 2^m is given, where all the elements of these vectors belong to the Galois Field $GF(2^8)$. Each vector has 4 elements. Additionally, a 4×4 matrix named A is given, whose elements belong to $GF(2^8)$. Let's denote

the i-th vector as x_i. In the end, we want n output vectors, and the i-th output vector, denoted as y_i, is equal to:

$$y_i = A^i x_0 + A^{i-1} x_1 + A^{i-2} x_2 + \cdots + x_i$$

In the above expression, A^i represents the matrix A raised to the power of i.

4 Execute

After running the program, scan is performed once with the CPU, and the number of incorrect elements obtained with the GPU is calculated. If this number is zero, your program has been executed correctly. You are only required to complete the file scan.cu in this program. Ultimately, the correctness and speed of your program will be evaluated based solely on the file scan.cu.

Compile:

```
/usr/local/cuda/bin/nvcc -O2 scan.cu scan_main.cu finite_field.cpp -o main.exe
```

Execute:

```
./main.exe m
```

In the above expression, m is $0 \leq m \leq 26$ and 2^m is the number of vectors.