CS 421: OutLab: Back To Basics: Odd-Even Cuda

- Released: Sunday March 4 1:30 PM. Due: Thursday March 8 10:00 PM
- Please do not forget the honor code.

Overview

This is an individual assignment. Any discussions should be constrained to public Piazza postings. You are not to discuss with ANY other person (same class, same batch, different batch, unknown person, ...). The only exception is your professor.

Important Note: Since you know the algorithm inside out, you are unlikely to need help from the internet for coding; you are not expected to take code from the Internet. Any links considered for review or for studying must be cited as usual. (See an important exception below for Cuda errors.)

Task A

We are going to work with odd even sort again, this time with CUDA Please note the following constraints

1. The program should adhere to the general style of openMP odd even sort that has been given to you earlier. For example, you should be ready to either generate the test input, or read from standard input.
2. In particular, you will implement the function `Odd_even` as a CUDA kernel as follows in increasing order of complexity (each of them will result in a file called `ansX.cu`). Each of the cases will result in elapsed time being printed on the standard output in milliseconds (round time to the nearest integer). Debug output, such as the sorted output, and indeed any thing other than the time taken, should go to `stderr` so that we can compare the answers. Please ask your friends on Piazza on how to do this if this is not clear.
   (a) The number of elements equal to the number of blocks. (Recall that the code fragment provided asks for (on the command line) the number of threads as input. For simplicity, the user will simply repeat the number to indicate the number of blocks, which is the number of elements.) Assume data will fit in global device memory.
   (b) The number of blocks is one (Recall that the code fragment provided asks for the number of threads. For simplicity, the user will simply repeat the number to indicate the number of threads.) (Also, reflect on the differences – if any – between the previous case – see below on reflection essay).
   (c) No (so-called) shared memory was used in the previous two cases. Now use `__shared__` using one block. Also multiple kernel launches are considered bad. Minimize kernel launches. Assume data fits in global memory; do not assume data will fit in shared memory.
   (d) Now, use 128 blocks and 1024 threads per block. Set your data size so that the number of data items is one per thread. You are free to use or not use shared memory. (Recall that the code fragment provided asks for the number of threads. Use the product of the numbers in the input for thread count.)
   (e) Assume 16 blocks and 1024 threads per block. Assume data size is large but will fit in GPU memory. Set chunksize=1024, and let each thread process the data in chunks. You are free to use or not use shared memory but the goal is to get the best possible time, with the restriction that we are going to use odd-even sort everywhere for sorting. (Recall

that the code fragment provided asks for the number of threads and the size of the input. Use the product of blocks and threads for thread count. Number of elements is similarly computed.)

Task B

This is the reflection essay. Explain your method and difficulty faced in each of the above cases. Also compare your time with the best MPI version and the best openMP version available to you on the web page. Report the speedup with respect to the serial version of the program. Use tables as necessary; a good pdf is nice to have (and good for your records). There are many subsets of comparisons possible (e.g. a vs b, or (a,b) vs (c)). Why would you choose one over the other, if you would?

Take time to write your reflection essay since it will be graded harshly. Write it nicely so that we know you understood the differences.

How We will Grade You

- (a) through (d) will be based on correctness
- (e) will be based on correctness and quality (time taken). This is 30%. In particular, we will take the times from the submissions and do some sort of binning (after a linear scale). The best submission will get a bonus.
- The reflection essay carries an additional 25% marks, i.e., it is over and above the programming part (which carries 100% for a net of 125%.)
- Code quality is also important. Make sure to document your code and always use cuda error checking (see this link). Unlike Cuda errors, you don't have to worry about the code being general to handle arbitrary input from a C-programming perspective (but be nice to use macros instead of placing hard coded numbers inside code).

Submission

You can ask for hints from the teaching team if you get stuck. As always, stay tuned on Piazza for clarifications Your group number is the last two digits of your roll number with some exceptions.

1. Do include a readme.txt (telling me whatever you want to tell me). Do include group members if any (name, roll number), group number, honour code, citations, etc.
2. As mentioned, you should periodically commit to GitHub.
3. As usual, submit to Moodle. Grading is based on Moodle submission. The folder and its compressed version should be similarly named. (For example: the folder is `lab02_group07_outlab` and the related `tar.gz` is `lab02_group07_outlab.tar.gz`
4. Your submission should look something like

```
cudaOutlab_groupXY
├── ansA.cu
├── ansB.cu
├── ansC.cu
├── ansD.cu
├── ansE.cu
├── reflection.txt
├── makefile
└── readme.txt
```