Assignment II, Spring 2017
PLEASE START EARLY!!!!!

Due Date: 11:59PM EST, April 7th, 2017

Goals

- Become familiarized with GPU architecture and environment
- Learn basic CUDA programming

Problem Description and Tasks

In this assignment, you are expected to write a CUDA program to compute the "trail" of a simulated moving object with high performance. The technique to locate the position of the object is called *2D trilateration*. Trilateration is one of the most basic algorithms used in GPS. This assignment focuses on its simplified version on a 2D plane.

2D Trilateration If we know the (x, y) positions of three "guard" points a, b, c and the distances of a point p to a, b, c – say da, db, dc – it is easy to know the (x, y) position of point p via basic geometry. A description can be found at:

`http://electronics.howstuffworks.com/gadgets/travel/gps1.htm`

There are numerous online articles that contain the formula for 2D trialateration.

The GPU Algorithm Your CUDA kernel function includes the following input, either as parameters or global variables (you decide):

- The constant value of guarding point a's location.

- The constant value of guarding point b's location.
- The constant value of guarding point c's location.
- a very large set (say, of number `NUM`) of 3-place tuples, each represented by three distances of *da*, *db*, *dc* as we described in the 2D trilateration algorithm

The kernel program performs 2D triangulation for each tuple, and *for every 4 consecutive trilateration results*, it computes the average. In other words, the output of the kernel function should only contain `NUM`/4 number of floating point numbers.

Simulated Data Generation Your CPU-side of the program should generate the (large set of) data. In the worst case, you can manually type in thousands of numbers into a file and read from it. A much easier way is to define a "random trail" to populate your data array(s). For instance, you can first assume the first location to be some (x_0, y_0). To compute the location (x_k, y_k) for step k, your program randomly generates two values – Δx, Δy (making sure their absolute values are very small), and make $x_k = x_{k-1} + \Delta x$, $y_k = y_{k-1} + \Delta y$. With (x_k, y_k) around, the distances of that point to a, b, c — the tuple (da_k, db_k, dc_k) you need for step k — are simply the Euclidean distance.

With this generation strategy, your program is in fact self-verifiable – the average of every 4 consecutive (x_k, y_k) values above should correspond with the result of your CUDA program.

Optional: 3D Trilateration (10 points bonus) If you prefer, upgrade your algorithm to the 3D space.

Requirements and Guidelines

Some requirements and guidelines:

- All coordinate numbers should be floating point numbers.
- Your trilateration algorithm should be able to tolerate some error in the input data. It is standard knowledge in 2D geometry that, given 3 arbitrary distances, a position may not be found in 2D space. For instance, if the three guarding points are (0, 0), (0, 0), (500, 0), and the 3 distances are 0, 500, 0, obviously no such position would exist in a 2D plane. Your algorithm should set a threshold, so that it can return a position that is "close enough" when the error is within the threshold. (When it is beyond the threshold, it is your art on how your algorithm should behave.)

- Please execute your program with the following settings: (1) `NUM` is either (a) 2^{12}, (b) 2^{13}, (c) 2^{14}, (d) 2^{15}; (2) Let your CUDA card has U number of SMs, and each SM has V number of cores. Execute your kernel with $< block\_size, thread\_size >$ with the following configurations: (i) $< U, V >$, (ii) $< 2 * U, V >$, (iii) $< U, 2 * V >$, (iv), $< 2 * U, 2 * V >$, (v), $< U/2, V >$, (vi) $< U, V/2 >$, (vii) $< U/2, V/2 >$. Note that you need to execute your program with all combinations, that is 4 * 7 = 28 settings.

  You need to plot the execution times of all 28 executions in a bar chart (or a group of bar charts). Feel free to use any tool that helps you plot. Some possibilities are Microsoft Excel or Plotly (`https://plot.ly`)

  To find out what U and V are, please look into CUDA SDK. In particular, there is a program called `devicequery`. You could either run `devicequery` as a standard alone program, or you could copy/paste some of the related API invocations in the source code fragments of `devicequery` into your program, making your program more adaptable for hardware variations. If you choose to run `devicequery` as a standard alone program, you need to submit a copy of the screenshot for its output.

- For each of the 28 executions, you need to print out both the result of the GPU and the result from self-verification (see "simulated data generation"). If some of them are identical, you do not need to repeat the results. Just use English such as "result identical to setting 17"

- If you choose to work on 3D space too, please keep your 2D algorithm and 3D algorithm separate. (We need both.) For the 3D part of the assignment, the requirement for reporting results is identical (we need all 28 settings).

Work Environment

Even though you may install and configure your own CUDA environment on a variety of CUDA-enabled cards and operating systems, you are expected to compile and execute your program on department lab machines.

The command to compile a CUDA program is `nvcc`. Your typical source code program should have a suffix of `.cu`. For instance, if your program is defined as `myprog.cu` and compiled to binary code through `nvcc`, your should be able to execute the binary code by typing

```
./a.out
```

More options of using `nvcc` are described if you type `nvcc -h`. More examples of simple CUDA programs are located in the CUDA SDK. You can find some samples at

`http://docs.nvidia.com/cuda/cuda-samples/index.html`

Submission Guidelines

Your final submission should:

- include a README file (a .txt file) in your folder that includes
  - the names and email addresses of all of your group members
  - descriptions of your platform (OS, environment peculiarities)
  - descriptions on whether your program is able to compile and execute
  - brief descriptions on anything special about your submission that the TA should take note of
- zip up your folder. Suppose your account name is `david`. What you need to is to first change the directory to the parent folder of `david` and do

  ```
  tar -cvf david3.tar david/
  ```

- Use the Digital dropbox in the Blackboard to upload the tared file you created above.
- One submission is enough for each group. You can freely decide who is going to do the submission in your group.