CSc 656 Project 3
GPU development in CUDA

Due Monday 5/14/2018, 3pm (no late submissions)
7% of your grade

For this project, follow the instructions to access the tiger server at

http://unixlab.sfsu.edu/~whsu/csc656/info.html

Part a: CUDA warmup (max element in each column of a matrix) (25/100)

You were given CUDA code to sum the columns of a matrix, and store the sums in a result array, in

~whsu/csc746/Code/SumCol.cu

Consider a variation of SumCol.cu, that find the max of each column of the matrix; the max of each column is stored in a result array:

~whsu/csc746/Code/S18/MaxCol.cu

Note that the kernels have been left blank in MaxCol.cu. Fill in code for the kernel; do not change the main program, which initializes the matrix, and checks the GPU results against the CPU code.

Measurements

For MaxCol.cu, follow the timing code in SumCol.cu, and measure the execution times for the kernel for matrices of size 1024 x 1024, 2048 x 2048, and 4096 x 4096, with 8, 16, 32, 64 and 128 threads per block. Display the timings in a table, similar to Intro to GPU Computing Slide 50.

For each measurement, run the code with the same configuration five times. Discard the max and min run times, and average the remaining three.

Part b: count number of red particles in a system

You are given a C program that initializes an array containing (x, y, z) coordinates of particles **`pos[]`** . Each particle also has a *color* (see below). The NEIGHBORHOOD constant specifies a radius around each particle that has to be checked. The number of particles is NUMPARTICLES, always a power of 2. For each particle, the program checks the nearby particles specified by the NEIGHBORHOOD constant, and counts the number of red particles nearby.

When the program finishes, numReds[i] contains the number of red particles within a distance of NEIGHBORHOOD from particle i. This array is dumped to an output textfile, **`dump.out`** . The source code for the program can be found on tiger.sfsu.edu at:

~whsu/csc746/Code/S18/findReds.c

The initPos() function initializes the array of particles. Note that the layout of the array looks like this (the number of particles is NUMPARTICLES):

pos[0] = particle[0].x
pos[1] = particle[0].y
pos[2] = particle[0].z
pos[3] = particle[0].color
pos[4] = particle[1].x
pos[5] = particle[1].y
pos[6] = particle[1].z
pos[7] = particle[1].color
pos[8] = particle[2].x
…
pos[NUMPARTICLES*4-1] = particle[NUMPARTICLES-1].color

Hence, to access (x, y, z) coordinates of particle i, we need pos[i*4], pos[i*4+1], pos[i*4+2]. (This is like an *array of structs*.)

Color is encoded as an integer. A particle can only be one of three colors, one of them being red (see findReds.c).

The bulk of the computation is done in findRedsCPU ():

```
void findRedsCPU (float *p, int *numI)
```

called with these parameters:

```
findRedsCPU(pos, numReds);
```

pos[] is the input array of 3*NUMPARTICLES floats, containing (x, y, z) coordinates of the particles, as described above. For each particle i, findRedsCPU() checks each particle

j; if the distance between i and j is less than NEIGHBORHOOD, numReds[i] is incremented. When all the computations are done, the dumpResults() function writes numReds[] into an output file, **dump.out** , for verification.

All code for this project must run on tiger.sfsu.edu. (In case you're considering developing on your own machine, tiger runs CUDA 8.0.)

Goal: rewrite findReds.c to run on the GPU

Replace the findRedsCPU() function with a findRedsGPU() function to count the number of red particles close to each particle, for the pos[] array on the GPU. A driver program is provided on tiger.sfsu.edu at

~whsu/csc746/Code/S18/findRedsDriver.cu

You will have to change/extend some of the functions, and add a kernel function. Follow the general approach shown in the examples in class; your code in findRedsGPU() should

allocate data arrays on the GPU with cudaMalloc()
copy the host array pos[] to the corresponding device array
call a kernel function (named, for example) findRedsGPU() to count the possible red particles close to each particle
(findRedsGPU() writes a device array, corresponding to numReds[])
copy the results from the device to a host array

Your findRedsGPU() kernel function should run with NUMPARTICLES threads total, divided into blocks. Each thread takes care of one iteration of the outermost i loop in findRedsCPU(). Hence, thread T goes through all the other particles, counts the number of particles that are within NEIGHBORHOOD from particle T, and writes the count to the result array.

The code in the main program handles timing, calls initPos() to initialize the pos[] array, calls findRedsGPU() to count the nearby red particles for each particle on the GPU, and writes results to the dump.out file.

The dump.out file generated by your GPU code should be identical to the one generated by the CPU code. (For floating point results, it's ok for some of the distances to be off by a tiny amount; there may be slight floating point precision issues with the GPU.)

Measurements

Now you have two versions of the program: findReds.c, and findRedsGPU.cu (with CUDA kernel). For each version, measure the run times for NUMPARTICLES = 1024, 8192, and 32768. For the GPU code, run on the Titan on tiger, with threads per block = 4, 16, and 64. Follow the instructions here:

http://unixlab.sfsu.edu/~whsu/csc656/info.html

For each data set size and configuration, make five run-time measurements. Discard the max and min run times, and average the remaining three.

Submission:

Submit all code, data and analysis as zip file via the iLearn submission link. Bring a paper copy of the submission to class before the final exam.

Your submission should include:

Part a:

1) source file MaxCol.cu
2) table of measurements for 3 matrix sizes and 5 GPU thread configurations

Part b:

1) source file findRedsGPU.cu, with unambiguous compile and run instructions
2) table of average run-times for 3 data sizes, 1 CPU + 3 GPU thread configurations
3) appendix containing all raw data for run-times