**Final project**

Reading following sequential code for Nbody Problem. You are required to implement a parallel version program of Nbody with CUDA.

You can use all kinds of mechanisms and techniques to speed up the parallel program.

The final report includes complete source code with annotations and additional explanation on the techniques you adopted to accelerate your program.

**Grading Policy:**

Total pts: 100

40 pts: the correctness of the parallel program

20 pts: sufficient and readable annotation in the source code

40 pts: additional accelerate Techniques (correctness+ annotation)

10 pts for each technique, Included but not limited to shared memory, loop optimization, compiler directions, joint programming with MPI etc.

**Deadline: 03-30-2021**

**Source codes:**

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#define SOFTENING 1e-9f

/\*

\* Each body contains x, y, and z coordinate positions,

\* as well as velocities in the x, y, and z directions.

\*/

typedef struct { float x, y, z, vx, vy, vz; } Body;

/\*

\* Do not modify this function. A constraint of this exercise is

\* that it remain a host function.

\*/

void randomizeBodies(float \*data, int n) {

 for (int i = 0; i < n; i++) {

   data[i] = 2.0f \* (rand() / (float)RAND\_MAX) - 1.0f;

 }

}

/\*

\* This function calculates the gravitational impact of all bodies in the system

\* on all others, but does not update their positions.

\*/

void bodyForce(Body \*p, float dt, int n) {

 for (int i = 0; i < n; ++i) {

   float Fx = 0.0f; float Fy = 0.0f; float Fz = 0.0f;

   for (int j = 0; j < n; j++) {

     float dx = p[j].x - p[i].x;

     float dy = p[j].y - p[i].y;

     float dz = p[j].z - p[i].z;

     float distSqr = dx\*dx + dy\*dy + dz\*dz + SOFTENING;

     float invDist = rsqrtf(distSqr);

     float invDist3 = invDist \* invDist \* invDist;

     Fx += dx \* invDist3; Fy += dy \* invDist3; Fz += dz \* invDist3;

   }

   p[i].vx += dt\*Fx; p[i].vy += dt\*Fy; p[i].vz += dt\*Fz;

 }

}

int main(const int argc, const char\*\* argv) {

 /\*

  \* Do not change the value for `nBodies` here. If you would like to modify it,

  \* pass values into the command line.

  \*/

 int nBodies = 2<<11;

 int salt = 0;

 if (argc > 1) nBodies = 2<<atoi(argv[1]);

 /\*

  \* This salt is for assessment reasons. Tampering with it will result in automatic failure.

  \*/

 if (argc > 2) salt = atoi(argv[2]);

 const float dt = 0.01f; // time step

 const int nIters = 10;  // simulation iterations

 int bytes = nBodies \* sizeof(Body);

 float \*buf;

 buf = (float \*)malloc(bytes);

 Body \*p = (Body\*)buf;

 /\*

  \* As a constraint of this exercise, `randomizeBodies` must remain a host function.

  \*/

 randomizeBodies(buf, 6 \* nBodies); // Init pos / vel data

 double totalTime = 0.0;

 /\*

  \* This simulation will run for 10 cycles of time, calculating gravitational

  \* interaction amongst bodies, and adjusting their positions to reflect.

  \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Do not modify this line of code.

for (int iter = 0; iter < nIters; iter++) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

  \* You will likely wish to refactor the work being done in `bodyForce`,

  \* as well as the work to integrate the positions.

  \*/

   bodyForce(p, dt, nBodies); // compute interbody forces

 /\*

  \* This position integration cannot occur until this round of `bodyForce` has completed.

  \* Also, the next round of `bodyForce` cannot begin until the integration is complete.

  \*/

   for (int i = 0 ; i < nBodies; i++) { // integrate position

     p[i].x += p[i].vx\*dt;

     p[i].y += p[i].vy\*dt;

     p[i].z += p[i].vz\*dt;

   }

 free(buf);

}