#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include "timer.h"

#include "files.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;

/\*

\* Calculate the gravitational impact of all bodies in the system

\* on all others.

\*/

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

int gid = threadIdx.x + blockDim.x \* blockIdx.x;

int stride = blockDim.x \* gridDim.x;

for (int i = gid; i < n; i += stride) {

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) {

// The assessment will test against both 2<11 and 2<15.

// Feel free to pass the command line argument 15 when you generate ./nbody report files

int nBodies = 2<<11;

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

// The assessment will pass hidden initialized values to check for correctness.

// You should not make changes to these files, or else the assessment will not work.

const char \* initialized\_values;

const char \* solution\_values;

if (nBodies == 2<<11) {

initialized\_values = "09-nbody/files/initialized\_4096";

solution\_values = "09-nbody/files/solution\_4096";

} else { // nBodies == 2<<15

initialized\_values = "09-nbody/files/initialized\_65536";

solution\_values = "09-nbody/files/solution\_65536";

}

if (argc > 2) initialized\_values = argv[2];

if (argc > 3) solution\_values = argv[3];

const float dt = 0.01f; // Time step

const int nIters = 10; // Simulation iterations

int bytes = nBodies \* sizeof(Body);

float \*buf;

cudaMallocManaged(&buf,bytes);

Body \*p = (Body\*)buf;

read\_values\_from\_file(initialized\_values, buf, bytes);

double totalTime = 0.0;

/\*

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

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

\*/

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

StartTimer();

/\*

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

\* and potentially the work to integrate the positions.

\*/

bodyForce<<<256,256>>>(p, dt, nBodies); // compute interbody forces

cudaDeviceSynchronize();

/\*

\* 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;

}

const double tElapsed = GetTimer() / 1000.0;

totalTime += tElapsed;

}

double avgTime = totalTime / (double)(nIters);

float billionsOfOpsPerSecond = 1e-9 \* nBodies \* nBodies / avgTime;

write\_values\_to\_file(solution\_values, buf, bytes);

// You will likely enjoy watching this value grow as you accelerate the application,

// but beware that a failure to correctly synchronize the device might result in

// unrealistically high values.

printf("%0.3f Billion Interactions / second\n", billionsOfOpsPerSecond);

cudaFree(buf);

}