Parallel & Distributed Computing
CSE525

Assignment **#6** - to be submitted to **Dr. Masroor Hussain**

Report on any selected CUDA based Simulation's Performance

Submitted by,
Quswar Mahmood Abid, CS2003
Muhammad Sherjeel, CS1913

CUDA Project

Download and CUDA code, run the code and submit a report including presentation.

In this assignment, we are required to source a code based on CUDA and run it. We are also required to present its performance. The code we chose simulate N-Body with Barnes Hut Algorithm. The coded implementation I have sourced is freely published by Raghav Pandya (on GitHub) and provides parallel versions implemented in OpenGL, CUDA, and OpenMP alongside its serial implementation. Availability of implementation in different platforms and coding models with same data was the reason to select it. An edited fork of his repository is available at this link [https://github.com/quswarabid/raghavpandya]

To get started, clone this repository, or use the file attached. To clone, execute this.

```
$ git clone https://github.com/quswarabid/raghavpandya.git
```

Navigate to **`/2.cuda`** for the parallelized version of serial code provided in **`/1.serial`** directory.

Execute ./a.out in both folders to run the file. To re-compile in case of any editions, execute:

```
$ nvcc n_body_cuda.cu
```

This project uses six set of n-bodies to evaluate serial vs. parallel performances. These are 4, 120, 240, 480, 600, and 960 bodies, respectively. Input vectors are saved in **`Nbody.h`** file for position, velocity, acceleration, and mass. Code for benchmarking is included with in the file. Now we will run this simulation for 4 bodies and look at changes made in it. Then we will just run it with different number of bodies, just to benchmark.

```
[u1@hpc 1. serial]$ ./a.out

Body 1:
Mass: 100000000272564224.000000
Position(x ,y, z): 0.000000, 0.000000, -1000.000000
Velocity(x, y, z): 0.000000, 0.000000, 0.000000
Acceleration(x ,y, z): 0.000000, 0.000000, 0.000000


Body 2:
Mass: 10.000000
Position(x ,y, z): 0.000000, 200.000000, -1000.000000
Velocity(x, y, z): -3.000000, -3.000000, -3.000000
Acceleration(x ,y, z): 0.000000, 0.000000, 0.000000


Body 3:
Mass: 10.000000
Position(x ,y, z): -200.000000, 0.000000, -1000.000000
Velocity(x, y, z): 3.000000, 3.000000, 3.000000
Acceleration(x ,y, z): 0.000000, 0.000000, 0.000000


Body 4:
Mass: 20.000000
Position(x ,y, z): 0.000000, 0.000000, -800.000000
Velocity(x, y, z): 4.000000, -3.000000, 1.000000
Acceleration(x ,y, z): 0.000000, 0.000000, 0.000000

Time Taken by Serial implementation: 23.574744 ms
[u1@hpc 1. serial]$
```

Figure 1. a simulation of 10,000 rounds with only 4 bodies

This simulation took 23.57 ms to complete. If we do not print the bodies, it takes a little less time, since printing on outstream also consumes time. Look at following results after using different number of bodies in serial.

```
[u1@hpc 1. serial]$ g++ n_body.cpp
In file included from NBody-120.h:8,
                 from n_body.cpp:4:
VectorMath.h:23: warning: 'typedef' was ignored in this declaration
[u1@hpc 1. serial]$ ./a.out
Time Taken by Serial implementation: 15607.026700 ms
[u1@hpc 1. serial]$
```

Figure 2. a serial simulation with 120 bodies

```
[u1@hpc 1. serial]$ nano n_body.cpp
[u1@hpc 1. serial]$ g++ n_body.cpp
In file included from NBody-240.h:8,
                 from n_body.cpp:4:
VectorMath.h:23: warning: 'typedef' was ignored in this declaration
[u1@hpc 1. serial]$ ./a.out
Time Taken by Serial implementation: 57966.237457 ms
[u1@hpc 1. serial]$
```

Figure 3. a serial simulation with 240 bodies

```
[u1@hpc 1. serial]$ nano n_body.cpp
[u1@hpc 1. serial]$ g++ n_body.cpp
In file included from NBody-480.h:8,
                 from n_body.cpp:4:
VectorMath.h:23: warning: 'typedef' was ignored in this declaration
[u1@hpc 1. serial]$ ./a.out
Time Taken by Serial implementation: 235538.369417 ms
[u1@hpc 1. serial]$
```

Figure 4. a serial simulation with 480 bodies

```
[u1@hpc 1. serial]$ nano n_body.cpp
[u1@hpc 1. serial]$ g++ n_body.cpp
In file included from NBody-600.h:8,
                 from n_body.cpp:4:
VectorMath.h:23: warning: 'typedef' was ignored in this declaration
[u1@hpc 1. serial]$ ./a.out
Time Taken by Serial implementation: 369415.276886 ms
[u1@hpc 1. serial]$
```

Figure 5. a serial simulation with 600 bodies

```
[u1@hpc 1. serial]$ nano n_body.cpp
[u1@hpc 1. serial]$ g++ n_body.cpp
In file included from NBody-960.h:8,
                 from n_body.cpp:4:
VectorMath.h:23: warning: 'typedef' was ignored in this declaration
[u1@hpc 1. serial]$ ./a.out
Time Taken by Serial implementation: 941173.343297 ms
[u1@hpc 1. serial]$
```

Figure 6. a serial simulation with 960 bodies

After we look how OpenMP based N-body simulation work, we will compare the two w.r.t. time.

Number of bodies	Time to simulate for 10,000 rounds (in ms)
4	23.574744
120	15,607.0267
240	57,966.237457
480	235,538.369417
600	369,415.276886
960	941,173.343297

Now let us take a look at how CUDA based code perform.

```
[u1@hpc raghavpandya]$ cd 3.\ cuda/
[u1@hpc 3. cuda]$ nano n_body_cuda.cu
[u1@hpc 3. cuda]$ nvcc n_body_cuda.cu
VectorMath.h(24): warning: declaration requires a typedef name

VectorMath.h(24): warning: declaration requires a typedef name

[u1@hpc 3. cuda]$ ./a.out
Time Taken by CUDA implementation: 150.459789 ms
```

Figure 7. a parallel simulation with 4 bodies

```
[u1@hpc 3. cuda]$ nano n_body_cuda.cu
[u1@hpc 3. cuda]$ nvcc n_body_cuda.cu
VectorMath.h(24): warning: declaration requires a typedef name

VectorMath.h(24): warning: declaration requires a typedef name

[u1@hpc 3. cuda]$ ./a.out
Time Taken by CUDA implementation: 5465.990291 ms
[u1@hpc 3. cuda]$
```

Figure 8. a parallel simulation with 120 bodies

```
[u1@hpc 3. cuda]$ nano n_body_cuda.cu
[u1@hpc 3. cuda]$ nvcc n_body_cuda.cu
VectorMath.h(24): warning: declaration requires a typedef name

VectorMath.h(24): warning: declaration requires a typedef name

[u1@hpc 3. cuda]$ ./a.out
Time Taken by CUDA implementation: 11295.018827 ms
```

Figure 9. a parallel simulation with 240 bodies

```
[u1@hpc 3. cuda]$ nano n_body_cuda.cu
[u1@hpc 3. cuda]$ nvcc n_body_cuda.cu
VectorMath.h(24): warning: declaration requires a typedef name

VectorMath.h(24): warning: declaration requires a typedef name

[u1@hpc 3. cuda]$ ./a.out
Time Taken by CUDA implementation: 22375.269365 ms
```

Figure 10. a parallel simulation with 480 bodies

```
[u1@hpc 3. cuda]$ nano n_body_cuda.cu
[u1@hpc 3. cuda]$ nvcc n_body_cuda.cu
VectorMath.h(24): warning: declaration requires a typedef name

VectorMath.h(24): warning: declaration requires a typedef name

[u1@hpc 3. cuda]$ ./a.out
Time Taken by CUDA implementation: 28459.379164 ms
[u1@hpc 3. cuda]$
```

Figure 11. a parallel simulation with 600 bodies

```
[u1@hpc 3. cuda]$ nano n_body_cuda.cu
[u1@hpc 3. cuda]$ nvcc n_body_cuda.cu
VectorMath.h(24): warning: declaration requires a typedef name

VectorMath.h(24): warning: declaration requires a typedef name

[u1@hpc 3. cuda]$ ./a.out
Time Taken by CUDA implementation: 49771.065545 ms
```

Figure 12. a parallel simulation with 960 bodies

Following table shows the performance of parallel algorithm in CUDA w.r.t. number of bodies.

Number of bodies	Time to simulate for 10,000 rounds (in ms)
4	150.459789
120	5,465.990291
240	11,295.018827
480	22,375.269365
600	28,459.379164
960	49,771.065545

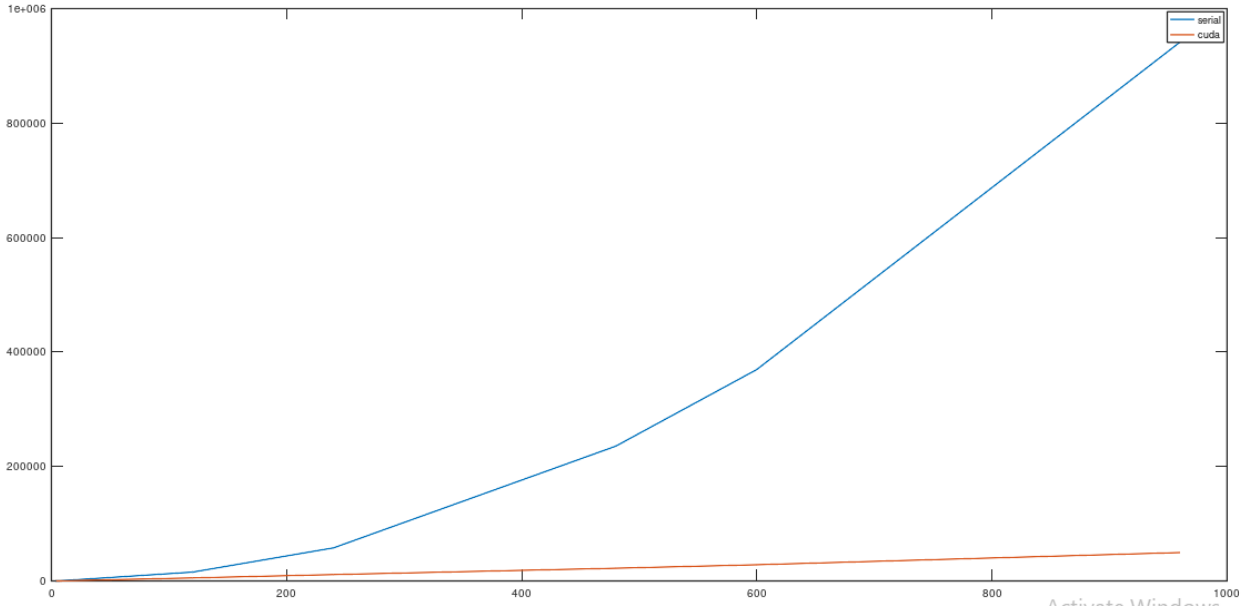

Figure 13. a comparison of time taken by serial and parallel algorithm to simulate n-bodies