

Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2022-23 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 9

Exam Seat No: 2019BTECS00064

Name – Kunal Santosh Kadam

Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

Problem Statement 1:

Implement Vector-Vector addition using CUDA C. State and justify the speedup using different size of threads and blocks.

Screenshot #:

The first screenshot shows the execution of a CUDA program for vector addition. The code uses `nvcc` to compile and run the program. The output indicates success and provides instructions for using `nsys` to generate a performance report. The report for `./vector-add-no-prefetch` is displayed, showing CUDA API and Kernel statistics.

CUDA API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
60.2	256933717	3	85644572.3	23741	256829364	cudaMallocManaged
34.6	147725131	1	147725131.0	147725131	147725131	cudaDeviceSynchronize
5.1	21772601	3	7257533.7	6647171	8442305	cudaFree
0.0	50943	1	50943.0	50943	50943	cudaLaunchKernel

CUDA Kernel Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
100.0	14777516	1	14777516.0	14777516	14777516	_addVectorsInto(float*, float*, float*, int)

The second screenshot shows the same notebook with a JavaScript code cell that generates a remote desktop link. The output of the `nsys` command is also visible, showing a different performance profile for the same program.

CUDA API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
58.2	255007657	3	85002552.3	28374	254894752	cudaMallocManaged
36.8	161068231	1	161068231.0	161068231	161068231	cudaDeviceSynchronize
5.0	21858186	3	7286062.0	6619404	8459467	cudaFree
0.0	46040	1	46040.0	46040	46040	cudaLaunchKernel

CUDA Kernel Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
100.0	161058318	1	161058318.0	161058318	161058318	_addVectorsInto(float*, float*, float*, int)

Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

The image displays a Jupyter Notebook interface within a web browser, showing the execution of a CUDA profiling script. The script uses `nsys profile` to generate a report for a vector-add operation. The output includes CUDA API and Kernel statistics.

CUDA API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
61.1	257051292	3	85683764.0	17585	256950616	cudaMallocManaged
33.6	141411304	1	141411304.0	141411304	141411304	cudaDeviceSynchronize
5.3	22105089	3	7368363.0	6606135	8613358	cudaFree
0.0	46030	1	46030.0	46030	46030	cudaLaunchKernel

CUDA Kernel Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
100.0	141397869	1	141397869.0	141397869	141397869	addVectorsInto(float*, float*, float*, int)

Open the Remote Desktop

Run the next cell to generate a link to the remote desktop. Then, read the instructions that follow in the notebook.

```
In [16]: %js
var port = (window.location.port == 80) ? "" : (":" + window.location.port);
var url = 'http://' + window.location.hostname + port + '/nsight/vnc.html?resize=scale';
let a = document.createElement('a');
a.setAttribute('href', url);
a.setAttribute('target', '_blank');
a.innerText = 'click to open remote desktop';
element.append(a);
```

[Click to open remote desktop](#)

The bottom screenshot shows the NVIDIA Nsight Systems 2021.3.1 interface. It displays a timeline view of the execution, with a red bar indicating the duration of the vector-add operation. The timeline includes various system events, such as context switches, memory transfers, and kernel launches. A tooltip for the 'poll' event shows the following details:

- Begin: 0.86197s
- End: 0.872542s (+10.072 ms)
- Call stack at 0.86197s: `poll`
- Begin: 0.872542s
- End: 0.872542s
- Begin: 0.872542s
- End: 0.872542s

Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Information #:

```
#include <stdio.h>

void initWith(float num, float *a, int N)
{
    for(int i = 0; i < N; ++i)
    {
        a[i] = num;
    }
}

__global__ void addVectorsInto(float *result, float *a, float *b, int N)
{
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    int stride = blockDim.x * gridDim.x;

    for(int i = index; i < N; i += stride)
    {
        result[i] = a[i] + b[i];
    }
}

void checkElementsAre(float target, float *vector, int N)
{
    for(int i = 0; i < N; i++)
    {
        if(vector[i] != target)
        {
            printf("FAIL: vector[%d] - %0.0f does not equal %0.0f\n",
i, vector[i], target);
            exit(1);
        }
    }
}
```

Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

```
printf("Success! All values calculated correctly.\n");
}

int main()
{
    int deviceId;
    int numberOfSMs;

    cudaGetDevice(&deviceId);
    cudaDeviceGetAttribute(&numberOfSMs,
        cudaDevAttrMultiProcessorCount, deviceId);

    const int N = 2<<24;
    size_t size = N * sizeof(float);

    float *a;
    float *b;
    float *c;

    cudaMallocManaged(&a, size);
    cudaMallocManaged(&b, size);
    cudaMallocManaged(&c, size);

    initWith(3, a, N);
    initWith(4, b, N);
    initWith(0, c, N);

    size_t threadsPerBlock;
    size_t numberOfBlocks;

    threadsPerBlock = 256;
    numberOfBlocks = 32 * numberOfSMs;

    cudaError_t addVectorsErr;
```

Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

```
cudaError_t asyncErr;  
  
addVectorsInto<<<numberOfBlocks, threadsPerBlock>>>(c, a, b, N);  
  
addVectorsErr = cudaGetLastError();  
if(addVectorsErr != cudaSuccess) printf("Error: %s\n",  
cudaGetErrorString(addVectorsErr));  
  
asyncErr = cudaDeviceSynchronize();  
if(asyncErr != cudaSuccess) printf("Error: %s\n",  
cudaGetErrorString(asyncErr));  
  
checkElementsAre(7, c, N);  
  
cudaFree(a);  
cudaFree(b);  
cudaFree(c);  
}
```

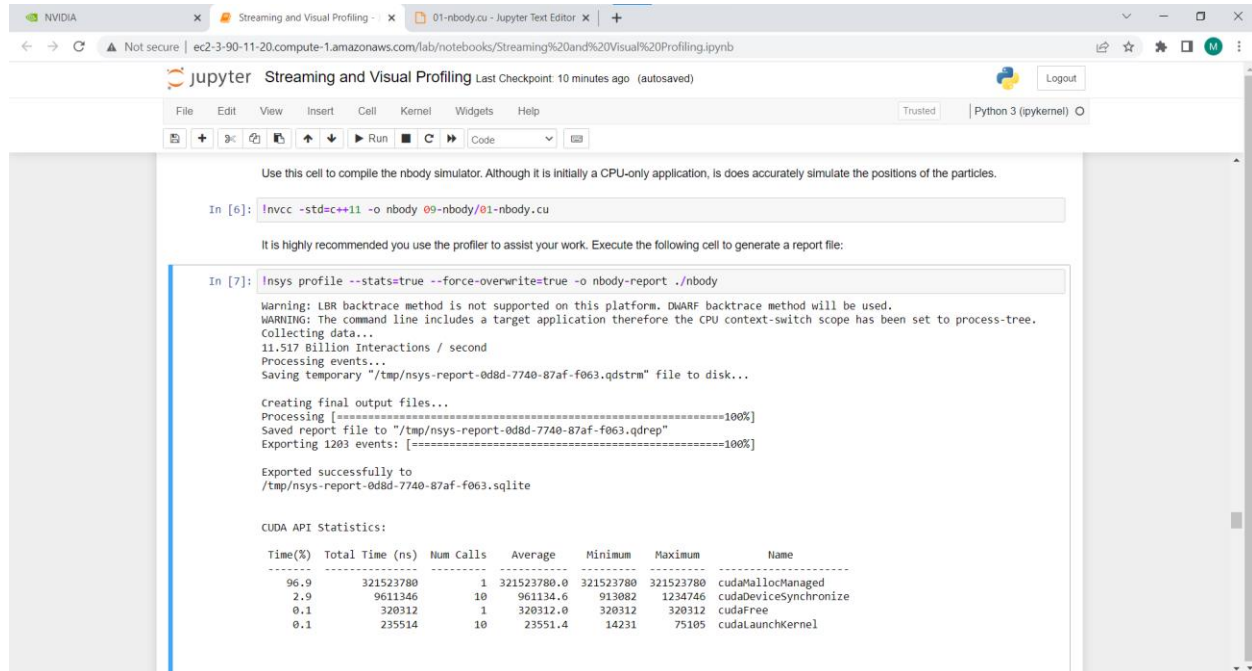
Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

Problem Statement 2:

Implement N-Body Simulator using CUDA C. State and justify the speedup using different size of threads and blocks.

Screenshot #:



The screenshot shows a Jupyter Notebook interface with the following content:

Use this cell to compile the nbbody simulator. Although it is initially a CPU-only application, it does accurately simulate the positions of the particles.

```
In [6]: !nvcc -std=c++11 -o nbbody 09-nbody/01-nbody.cu
```

It is highly recommended you use the profiler to assist your work. Execute the following cell to generate a report file:

```
In [7]: !nsys profile --stats=true --force-overwrite=true -o nbbody-report ./nbbody
```

Warning: LBR backtrace method is not supported on this platform. DWARF backtrace method will be used.
WARNING: The command line includes a target application therefore the CPU context-switch scope has been set to process-tree.
Collecting data...
11.517 Billion Interactions / second
Processing events...
saving temporary "/tmp/nsys-report-0d8d-7740-87af-f063.qdstrm" file to disk...

Creating final output files...
Processing [=====100%]
Saved report file to "/tmp/nsys-report-0d8d-7740-87af-f063.qdrep"
Exporting 1203 events: [=====100%]

Exported successfully to
/tmp/nsys-report-0d8d-7740-87af-f063.sqlite

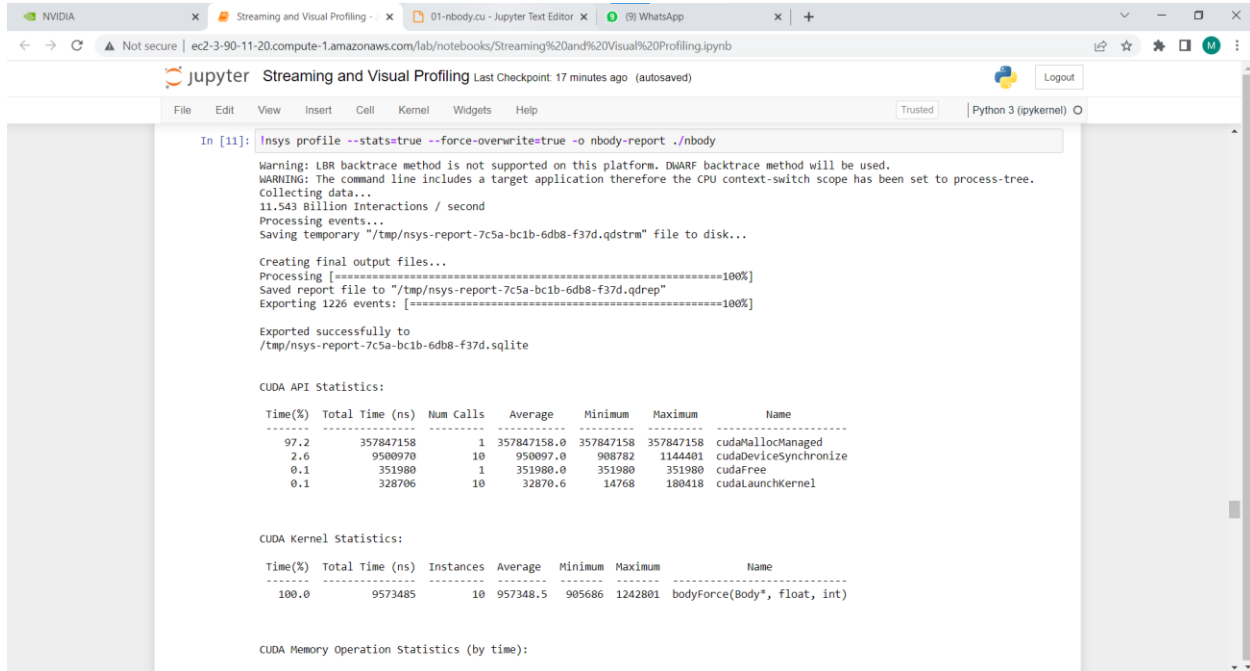
CUDA API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
96.9	321523780	1	321523780.0	321523780	321523780	cudaMallocManaged
2.9	9611346	10	961134.6	913082	1234746	cudaDeviceSynchronize
0.1	320312	1	320312.0	320312	320312	cudaFree
0.1	235514	10	23551.4	14231	75105	cudaLaunchKernel

Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

Number of thread per block – 32 with block size 1024



```
In [11]: !nbody --stats=true --force-override=true -o nbody-report ./nbody

Warning: LBR backtrace method is not supported on this platform. DWARF backtrace method will be used.
WARNING: The command line includes a target application therefore the CPU context-switch scope has been set to process-tree.
Collecting data...
11.543 Billion Interactions / second
Processing events...
Saving temporary "/tmp/nbody-report-7c5a-bc1b-6db8-f37d.qdstrm" file to disk...

Creating final output files...
Processing [=====100%]
Saved report file to "/tmp/nbody-report-7c5a-bc1b-6db8-f37d.qdrep"
Exporting 1226 events: [=====100%]

Exported successfully to
/tmp/nbody-report-7c5a-bc1b-6db8-f37d.sqlite

CUDA API Statistics:

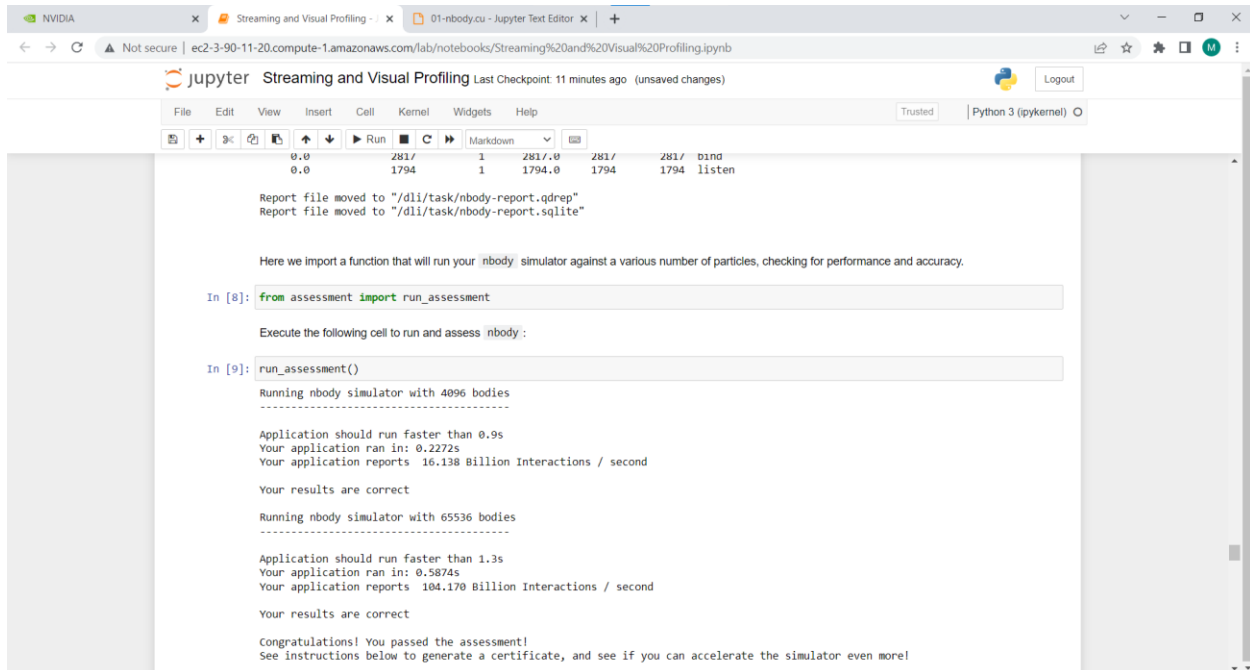
Time(%) Total Time (ns) Num Calls Average Minimum Maximum Name
-----
97.2 357847158 1 357847158.0 357847158 357847158 cudaMallocManaged
2.6 9500970 10 950097.0 908782 1144401 cudaDeviceSynchronize
0.1 351980 1 351980.0 351980 351980 cudaFree
0.1 328706 10 32870.6 14768 180418 cudaLaunchKernel

CUDA Kernel Statistics:

Time(%) Total Time (ns) Instances Average Minimum Maximum Name
-----
100.0 9573485 10 957348.5 905686 1242801 bodyForce(Body*, float, int)

CUDA Memory Operation Statistics (by time):
```

Number of thread per block – 32 with block size 512



```
In [8]: from assessment import run_assessment

Execute the following cell to run and assess nbody:

In [9]: run_assessment()

Running nbody simulator with 4096 bodies
-----
Application should run faster than 0.9s
Your application ran in: 0.2272s
Your application reports 16.138 Billion Interactions / second

Your results are correct

Running nbody simulator with 65536 bodies
-----
Application should run faster than 1.3s
Your application ran in: 0.5874s
Your application reports 104.170 Billion Interactions / second

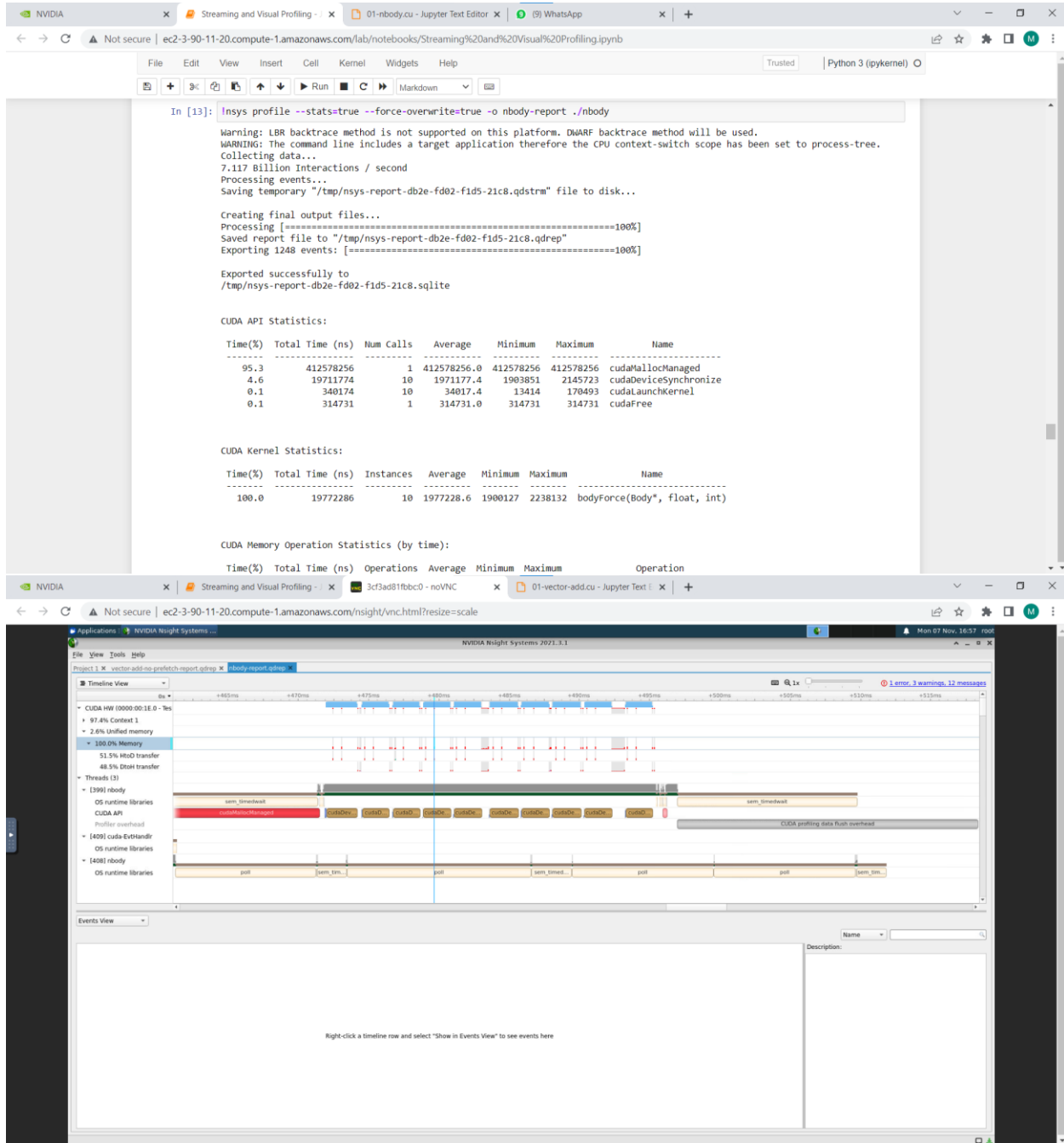
Your results are correct

Congratulations! You passed the assessment!
See instructions below to generate a certificate, and see if you can accelerate the simulator even more!
```


Walchand College of Engineering, Sangli

Department of Computer Science and Engineering

Number of thread per block – 4 with block size 1024



Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Information #:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "timer.h"
#include "files.h"

#define SOFTENING 1e-9f

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

__global__ void bodyForce(Body *p, float dt, int n)
{
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    int stride = blockDim.x * gridDim.x;

    for(int i = index; 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;
        }
    }
```

Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

```
        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
    y 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];
```

Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

```
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;

buf = (float *)malloc(bytes);
cudaMallocManaged(&buf, bytes)

Body *p = (Body*)buf;

read_values_from_file(initialized_values, buf, bytes);

double totalTime = 0.0;

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

    bodyForce<<<1024,32>>>(p, dt, nBodies); // compute
interbody forces
    cudaDeviceSynchronize();

    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;
```

Walchand College of Engineering, Sangli
Department of Computer Science and Engineering
}

```
double avgTime = totalTime / (double)(nltrs);  
float billionsOfOpsPerSecond = 1e-9 * nBodies * nBodies / avgTime;  
write_values_to_file(solution_values, buf, bytes);  
  
printf("%0.3f Billion Interactions / second\n",  
billionsOfOpsPerSecond);  
  
cudaFree(buf);  
}
```

Github Link:

<https://github.com/Kunalkadam179/HPC-Assignment/tree/main/Assignment%20-%209>