

SEM - VII - 2022-23

High Performance Computing Lab

Assignment - 9

Name: Pavan Krishnat Shinde

Title of practical:

Implementation of Vector-Vector addition & N-Body Simulator using CUDA C

Problem Statement 1:

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

```
#include <stdio.h>

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

```
__global__

void addVectorsInto(int *result, int *a, int *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(int target, int *array, int N)
{
    for(int i = 0; i < N; i++)
    {
        if(array[i] != target)
        {
            printf("FAIL: array[%d] - %d does not equal %d\n", i, array[i], target);
        }
    }
}
```

```
        exit(1);

    }

}

printf("SUCCESS! All values added correctly.\n");
}
```

```
int main()
{

    const int N = 2<<20;

    size_t size = N * sizeof(int);

    int *a;

    int *b;

    int *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 = (N + threadsPerBlock - 1) / threadsPerBlock;

addVectorsInto<<<numberOfBlocks, threadsPerBlock>>>(c, a, b, N);

cudaDeviceSynchronize();

checkElementsAre(7, c, N);

cudaFree(a);

cudaFree(b);

cudaFree(c);

}
```

```
nvcc -arch=sm_70 -o vectorAddition vectorAddition.cu -run
SUCCESS! All values added correctly.
```

```
nsys profile --stats=true vectorAddition
Warning: LBR backtrace method is not supported on this platform. DWARF backtrace method will be used.
Collecting data...
SUCCESS! All values added correctly.

The target application terminated with signal 11 (SIGSEGV)
Processing events...
Capturing symbol files...
Saving temporary "/tmp/nsys-report-8246-3bf4-060a-cbdd.qdstrm" file to disk...
Creating final output files...

Processing [=====100%]
Saved report file to "/tmp/nsys-report-8246-3bf4-060a-cbdd.qdrep"
Exporting 4923 events: [=====100%]

Exported successfully to
/tmp/nsys-report-8246-3bf4-060a-cbdd.sqlite

CUDA API Statistics:
-----
Time(%) Total Time (ns) Num Calls Average Minimum Maximum Name
-----
96.9 316,611,908 3 105,537,302.7 17,580 316,541,378 cudaMallocManaged
2.6 8,522,825 1 8,522,825.0 8,522,825 8,522,825 cudaDeviceSynchronize
```

```
Time(%) Total Time (ns) Num Calls Average Minimum Maximum Name
-----
96.9 316,611,908 3 105,537,302.7 17,580 316,541,378 cudaMallocManaged
2.6 8,522,825 1 8,522,825.0 8,522,825 8,522,825 cudaDeviceSynchronize
0.5 1,671,059 3 557,019.7 522,110 626,334 cudaFree
0.0 94,365 1 94,365.0 94,365 94,365 cudaLaunchKernel
```

```
CUDA Kernel Statistics:
-----
Time(%) Total Time (ns) Instances Average Minimum Maximum Name
-----
100.0 8,525,128 1 8,525,128.0 8,525,128 8,525,128 addVectorsInto(int*, int*, int*, int)
```

```
CUDA Memory Operation Statistics (by time):
-----
Time(%) Total Time (ns) Operations Average Minimum Maximum Operation
-----
80.5 3,000,112 324 9,259.6 2,655 85,375 [CUDA Unified Memory memcpy HtoD]
19.5 725,663 48 15,118.0 1,983 81,887 [CUDA Unified Memory memcpy DtoH]
```

```
CUDA Memory Operation Statistics (by size in KiB):
-----
Total Operations Average Minimum Maximum Operation
-----
8,192,000 48 170,667 4,000 1,020,000 [CUDA Unified Memory memcpy DtoH]
384,000 [CUDA Unified Memory memcpy HtoD]
```

Problem Statement 2:

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

```

#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 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;

    }

    p[i].vx += dt * Fx;

    p[i].vy += dt * Fy;

    p[i].vz += dt * Fz;
```

```

}

}

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

    int deviceId;

    int numberOfSMs;

    cudaGetDevice(&deviceId);

    cudaDeviceGetAttribute(&numberOfSMs, cudaDevAttrMultiProcessorCount, deviceId);

    // 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 nlters = 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.
     */

    size_t threadsPerBlock = 256;

    size_t numberOfBlocks = 32 * numberOfSMs;

    bodyForce<<<numberOfBlocks, threadsPerBlock>>>(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);

}
```

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

```
In [9]: !nsys profile --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...
9.977 Billion Interactions / second
Processing events...
Saving temporary "/tmp/nsys-report-ab84-acff-103f-6754.qdstrm" file to disk...
Creating final output files...
Processing [=====100%]
Saved report file to "/tmp/nsys-report-ab84-acff-103f-6754.qdrep"
Exporting 1219 events: [=====100%]
Exported successfully to
/tmp/nsys-report-ab84-acff-103f-6754.sqlite

CUDA API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
95.4	258379417	1	258379417.0	258379417	258379417	cudaMallocManaged
4.4	11920393	10	1192039.3	1141293	1429701	cudaDeviceSynchronize
0.1	345672	1	345672.0	345672	345672	cudaFree
0.1	215206	10	21520.6	12089	80893	cudaLaunchKernel

Execute the following cell to run and assess nbody:

```
In [11]: run_assessment()
```

Running nbody simulator with 4096 bodies

Application should run faster than 0.9s
Your application ran in: 0.2365s
Your application reports 13.167 Billion Interactions / second
Your results are correct

Running nbody simulator with 65536 bodies

Application should run faster than 1.3s
Your application ran in: 0.5209s
Your application reports 121.795 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!

Generate a Certificate

Github Link:

<https://github.com/pavanshinde7494/HPC-Assignment>