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:

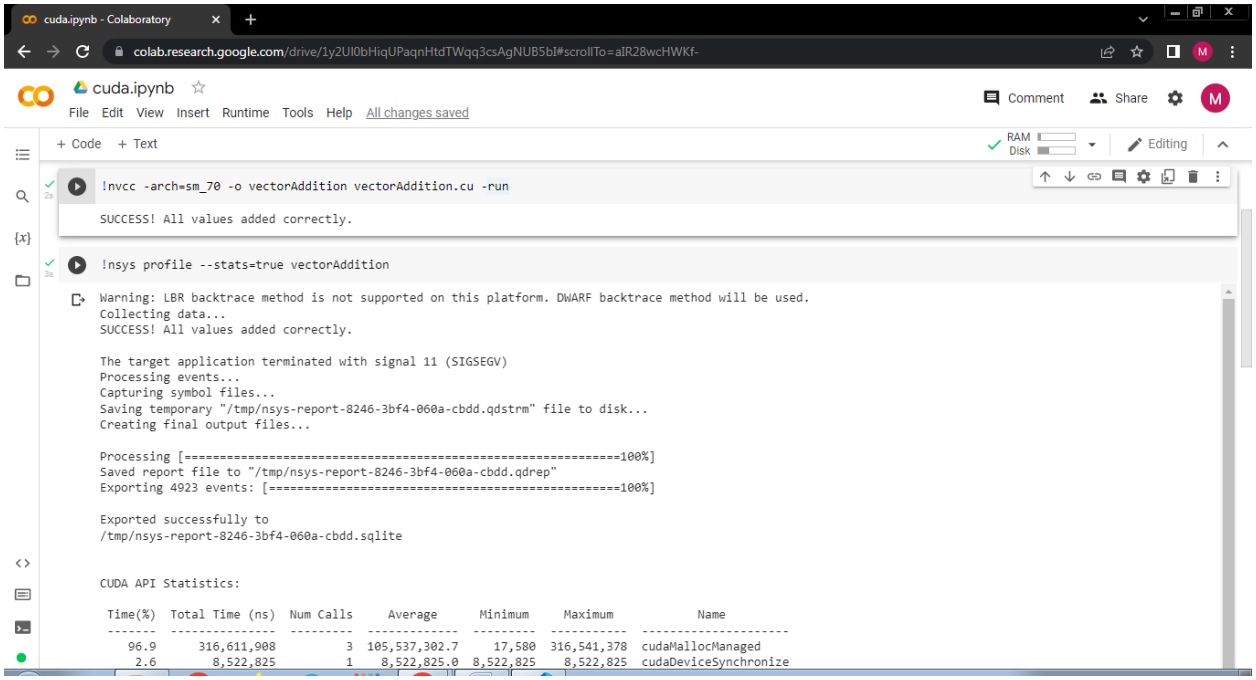
Implement Vector-Vector addition using CUDA C. State and justify the speedup using different size of threads and blocks.

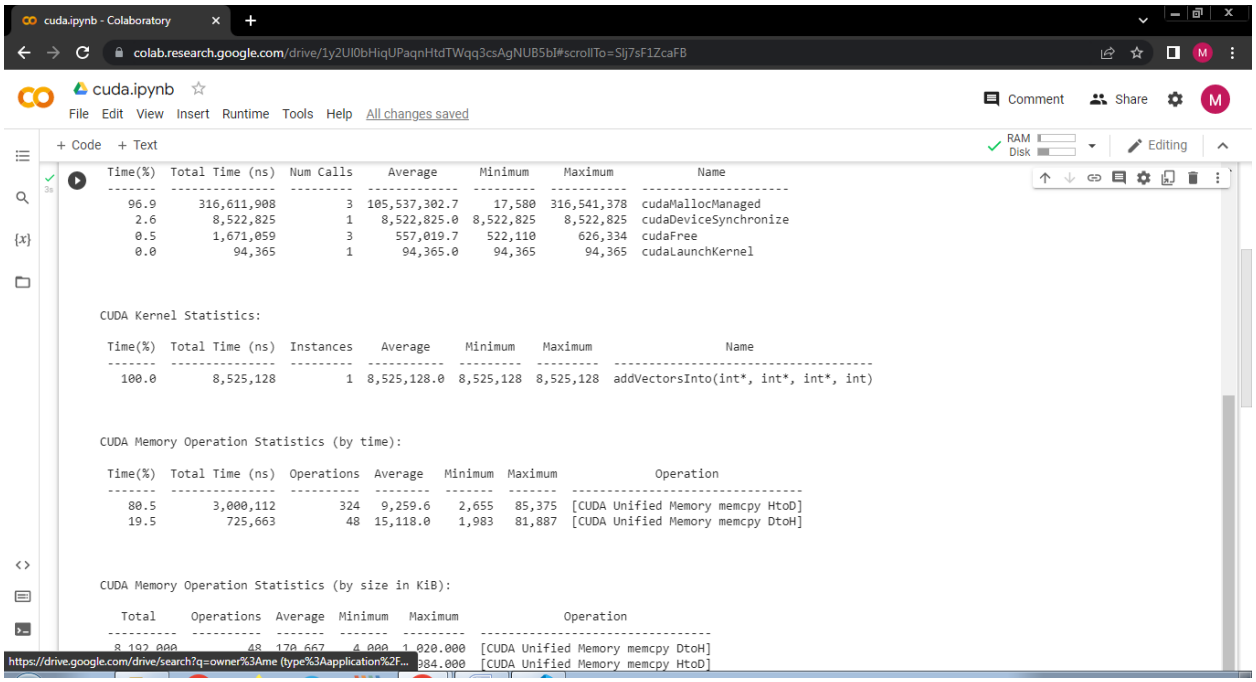
```
#include <stdio.h>



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);

}
```





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;
```

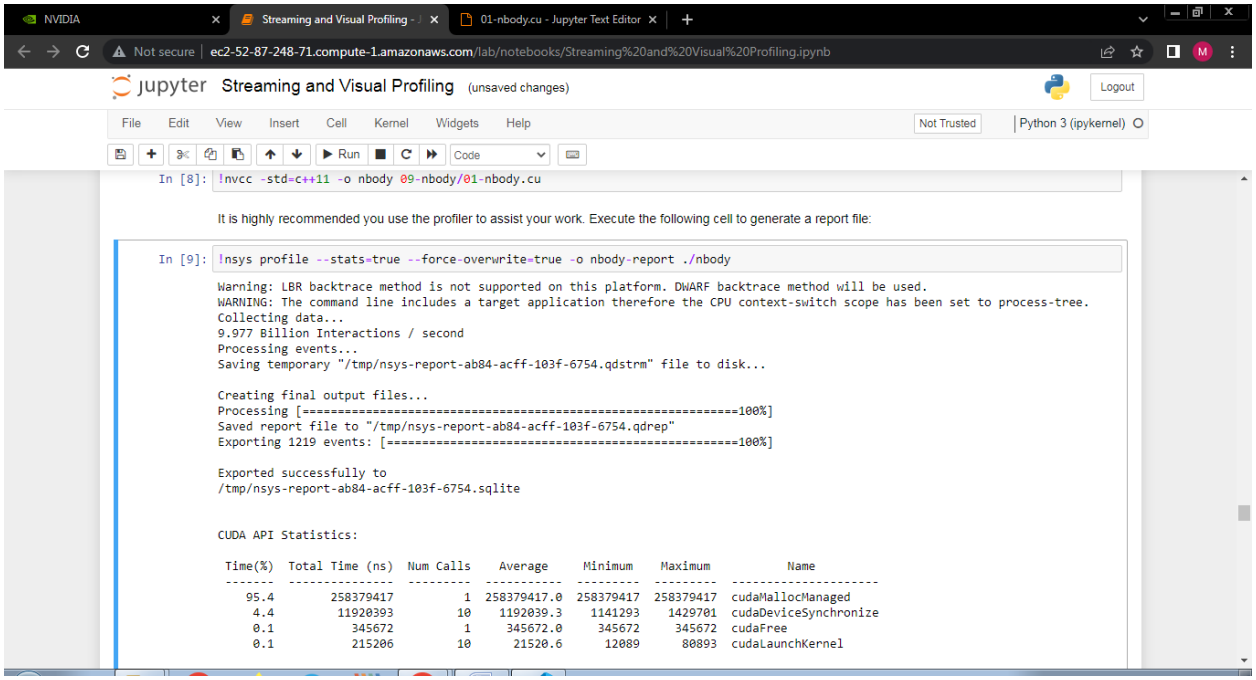
```
int stride = blockDim.x * gridDim.x;

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;
```

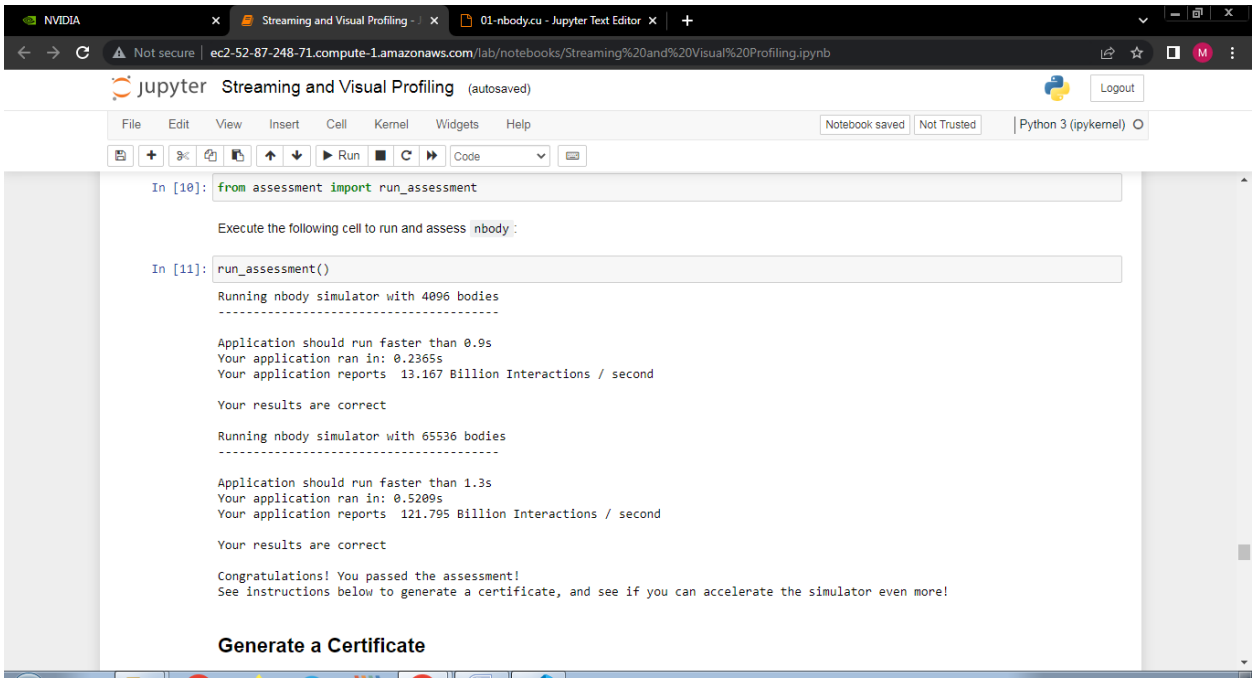
```
}

}

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 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.
   */
  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);

}
```





Github Link:

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