Name : Harshal Kodgire
PRN : 2019BTECS00029
Batch : B1
Sub : HPC Lab

Assignment - 9

Q1) Implement Vector-Vector addition using CUDA C. State and justify the speedup using different sizes of threads and blocks.

Code :

```
#include <stdio.h>

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

/*
 * Device kernel stores into `result` the sum of each
 * same-indexed value of `a` and `b`.
 */

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

int main()
{
  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);

      addVectorsInto<<<8,32>>>(c,a,b,N);
  cudaDeviceSynchronize();


  cudaFree(a);
  cudaFree(b);
  cudaFree(c);
}
```

For <<<1,1>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average      Minimum     Maximum           Name
 -------  ---------------  ---------  ------------  ----------  ----------  --------------------
    89.0       2197177162          1  2197177162.0  2197177162  2197177162  cudaDeviceSynchronize
    10.6        261585252          3    87195084.0       23727   261490522  cudaMallocManaged
     0.5         11147162          3     3715720.7     3650969     3813264  cudaFree
     0.0            39971          1       39971.0       39971       39971  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances    Average      Minimum     Maximum                      Name
 -------  ---------------  ---------  ------------  ----------  ----------  ----------------------------------------
   100.0       2197163231          1  2197163231.0  2197163231  2197163231  addVectorsInto(float*, float*, float*, int)
```

For <<<2,32>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum           Name
 -------  ---------------  ---------  -----------  ---------  ---------  --------------------
    55.2        305749237          1  305749237.0  305749237  305749237  cudaDeviceSynchronize
    42.6        235994217          3   78664739.0      16791  235930900  cudaMallocManaged
     2.3         12597439          3    4199146.3    4141923    4274340  cudaFree
     0.0            28649          1      28649.0      28649      28649  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances    Average     Minimum    Maximum                      Name
 -------  ---------------  ---------  -----------  ---------  ---------  ----------------------------------------
   100.0        305733663          1  305733663.0  305733663  305733663  addVectorsInto(float*, float*, float*, int)
```

For <<<4,64>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum           Name
 -------  ---------------  ---------  -----------  ---------  ---------  --------------------
    64.7        242791066          3   80930355.3      17296  242726305  cudaMallocManaged
    30.5        114271587          1  114271587.0  114271587  114271587  cudaDeviceSynchronize
     4.8         17923683          3    5974561.0    5936920    6045652  cudaFree
     0.0            33458          1      33458.0      33458      33458  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances    Average     Minimum    Maximum                      Name
 -------  ---------------  ---------  -----------  ---------  ---------  ----------------------------------------
   100.0        114256989          1  114256989.0  114256989  114256989  addVectorsInto(float*, float*, float*, int)
```

Q2) Implement N-Body Simulator using CUDA C. State and justify the speedup using different sizes of threads and blocks.

Code :

```
#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 i = blockIdx.x * blockDim.x + threadIdx.x;
  //int stride = gridDim.x * blockDim.x;
  if( i < n){



        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.
   */
    size_t threads_per_block = 256;
    size_t number_of_blocks = (nBodies + threads_per_block - 1) /
threads_per_block;

    bodyForce<<<number_of_blocks, threads_per_block>>>(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);
}
```

In [16]:
```
run_assessment()
```

```
Running nbody simulator with 4096 bodies
-----------------------------------------

Application should run faster than 0.9s
Your application ran in: 0.2437s
Your application reports  15.895 Billion Interactions / second

Your results are correct

Running nbody simulator with 65536 bodies
-----------------------------------------

Application should run faster than 1.3s
Your application ran in: 0.5703s
Your application reports  110.704 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!
```