**Class:** Final Year (Computer Science and Engineering)

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

**Course:** High Performance Computing Lab

**Practical No. 9**

PRN : 2019BTECS00070

Name : Prathmesh Killedar

## Problem Statement 1:

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

**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);

}

}

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;

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);

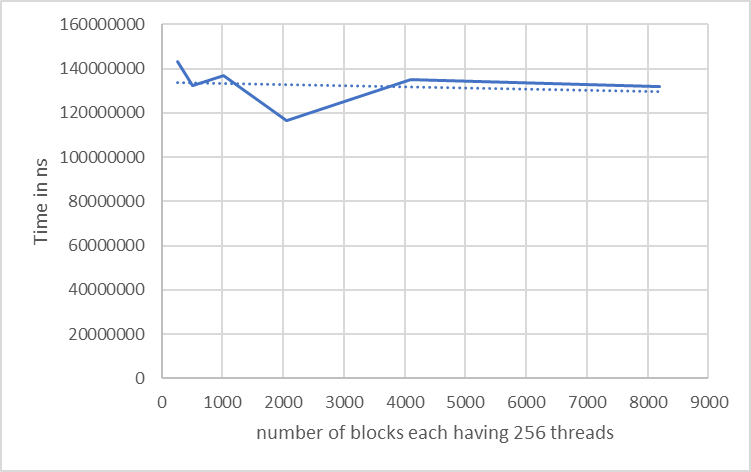
cudaFree(a);

cudaFree(b);

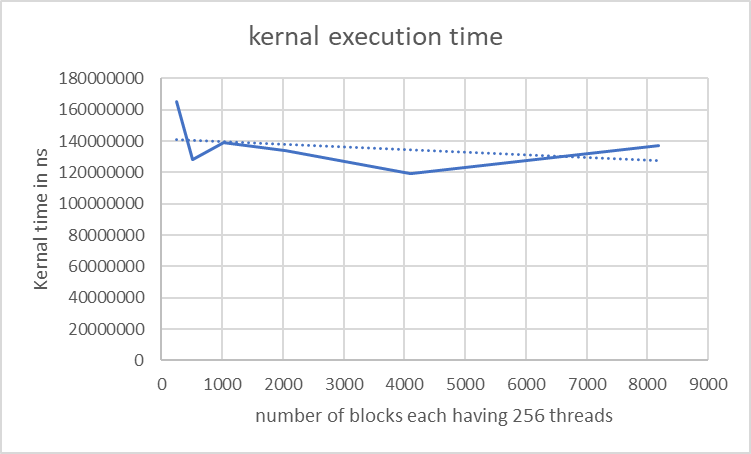
cudaFree(c);

}

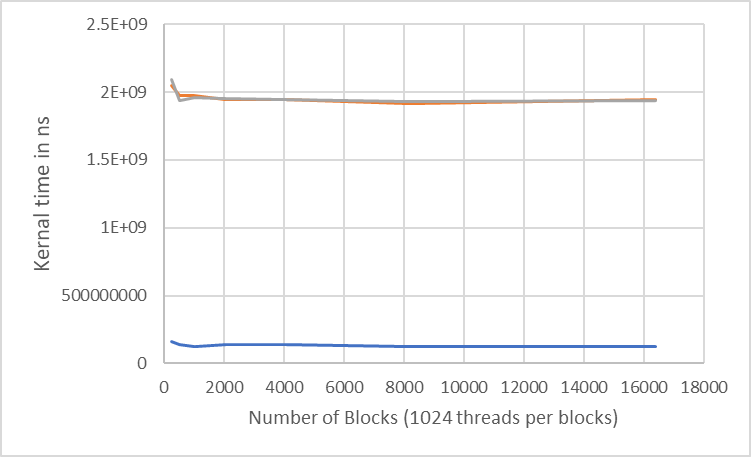
| **Blocks (thread constant 256)** | **kernal execution time** |
| --- | --- |
| **256** | **143037294** |
| **512** | **132251553** |
| **1024** | **137069194** |
| **2048** | **116664576** |
| **4096** | **134966460** |
| **8192** | **131768554** |

****

| **Blocks (thread constant 512)** | **kernal execution time** |
| --- | --- |
| **256** | **164921246** |
| **512** | **128164468** |
| **1024** | **139115093** |
| **2048** | **133940673** |
| **4096** | **118931852** |
| **8192** | **136805724** |

****

|  | **for 2<<24 data** | **for data INT\_MAX/sizeof(float)** | |
| --- | --- | --- | --- |
| **Blocks (thread constant 1024)** | **kernal execution time** | **kernal execution time** | **kernal execution time** |
| **256** | **160442261** | **2048682679** | **2091322925** |
| **512** | **141213279** | **1976524493** | **1941007817** |
| **1024** | **123889652** | **1975526236** | **1963358116** |
| **2048** | **137103827** | **1945817299** | **1949938430** |
| **4096** | **138071974** | **1942471645** | **1944636331** |
| **8192** | **124432532** | **1914443571** | **1929313649** |
| **16384** | **127385691** | **1944251296** | **1941889782** |

****

**Conclusion:**

By observing the above graph we have concluded that for the constant threads and increasing block number and for the constant blocks and increasing threads number the execution time is decreasing upto the certain point and after that due to communication overhead execution time is increasing.

## Problem Statement 2:

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

**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;

}

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];

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;

}

double avgTime = totalTime / (double)(nIters);

float billionsOfOpsPerSecond = 1e-9 \* nBodies \* nBodies / avgTime;

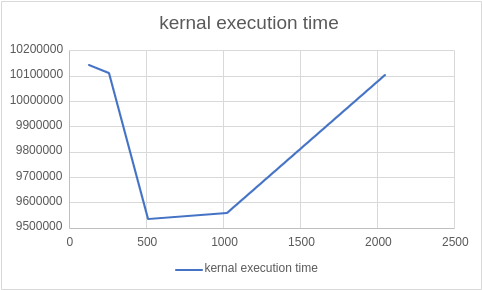
write\_values\_to\_file(solution\_values, buf, bytes);

printf("%0.3f Billion Interactions / second\n", billionsOfOpsPerSecond);

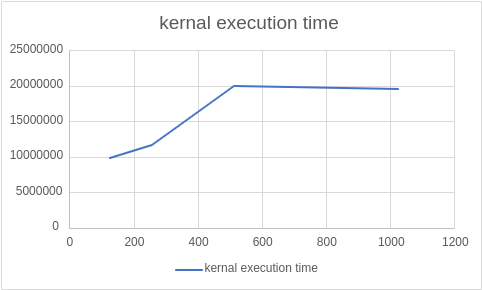
cudaFree(buf);

}

| **Blocks (thread constant 128)** | **kernal execution time** |
| --- | --- |
| 128 | 10140858 |
| 256 | 10110051 |
| 512 | 9533352 |
| 1024 | 9559230 |
| 2048 | 10101743 |



| **Threads(Block Constant 128)** | **kernal execution time** |
| --- | --- |
| 128 | 9780649 |
| 256 | 11664735 |
| 512 | 19929743 |
| 1024 | 19626103 |



**Conclusion:**

1. For constant number of blocks we have concluded that the execution time is increasing with the increasing number of threads per block
2. For constant number of threads we have concluded that the execution time is decreasing until a certain point and after that it is increasing due to communication overhead by increasing the number of block

**Github Link:** [**https://github.com/killedar27/HPC-assignments/tree/main/assignment9**](https://github.com/killedar27/HPC-assignments/tree/main/assignment9)