**Final Year B. Tech., Sem VII 2022-23**

**High Performance Computing Lab**

**PRN: 2020BTECS00206**

**Full Name: SAYALI YOGESH DESAI**

**Batch: B4**

**Assignment No. 9**

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

#include <stdio.h>

/\*

\* Host function to initialize vector elements. This function

\* simply initializes each element to equal its index in the

\* vector.

\*/

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];

}

}

/\*

\* Host function to confirm values in `vector`. This function

\* assumes all values are the same `target` value.

\*/

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);

initWith(3, a, N);

initWith(4, b, N);

initWith(0, c, N);

size\_t threadsPerBlock;

size\_t numberOfBlocks;

/\*

\* nsys should register performance changes when execution configuration

\* is updated.

\*/

threadsPerBlock = 1;

numberOfBlocks = 1;

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);

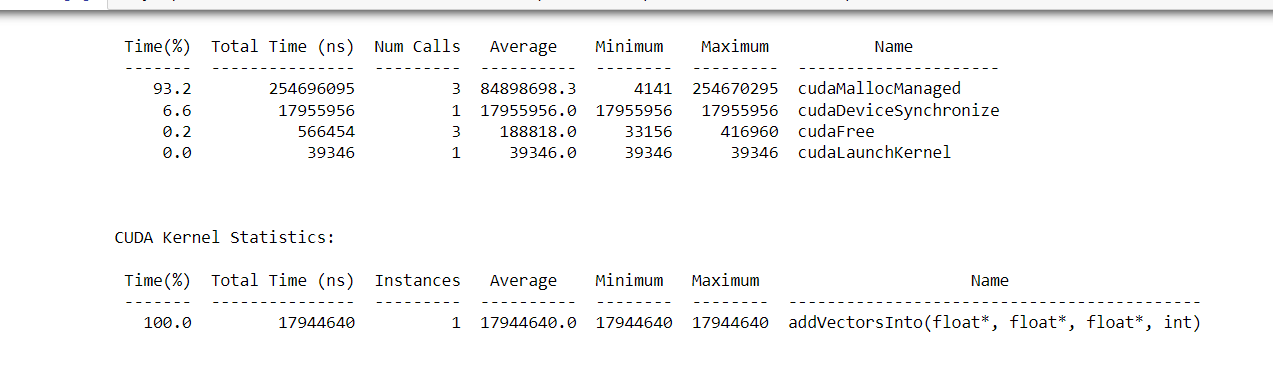
cudaFree(b);

cudaFree(c);

}

**N=2<<16**

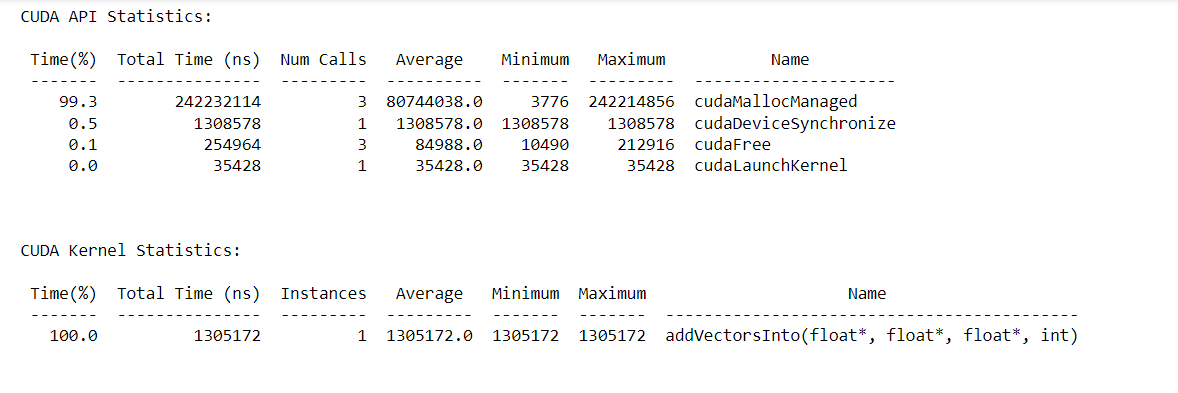
**Serial Execution Time: 17944640**



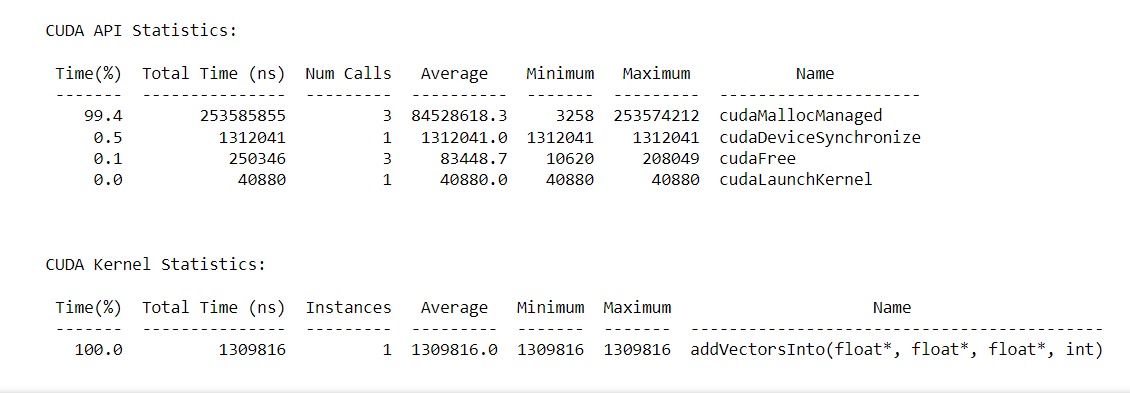
**Parallel Execution Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Blocks** | **Threads per Block** | **Time (in ns)** | **Speedup** |
| 8 | 256 | 1305172 | 13.7488 |
| 16 | 256 | 1309816 | 13.7001 |
| 32 | 256 | 1406457 | 12.7587 |
| 8 | 512 | 1567085 | 11.4509 |
| 16 | 512 | 1391095 | 12.8996 |
| 132 | 512 | 1384345 | 12.9625 |

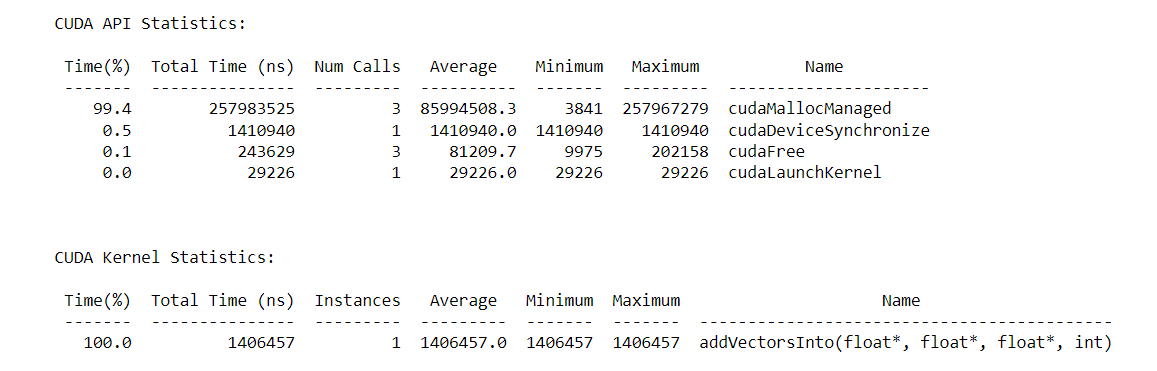
**Number of blocks: 8, Threads per block: 256, Time Required: 1305172**



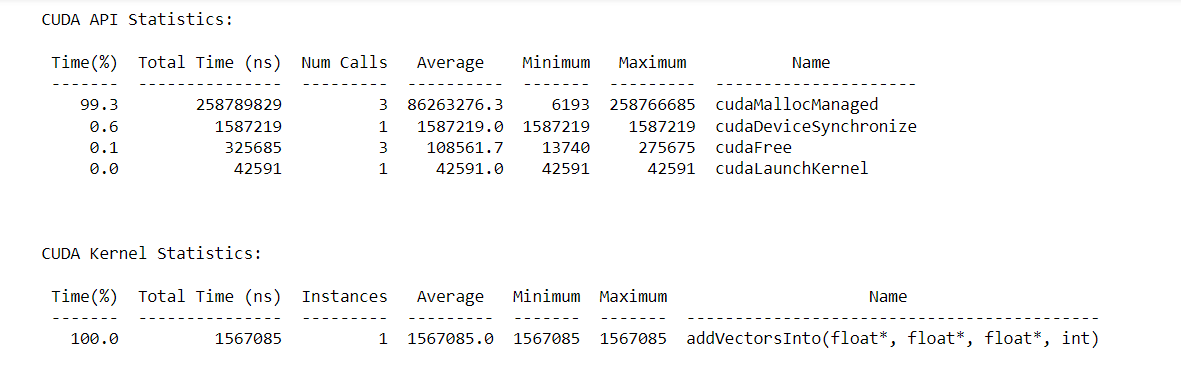
**Number of blocks: 16, Threads per block: 256, Time Required: 1309816**



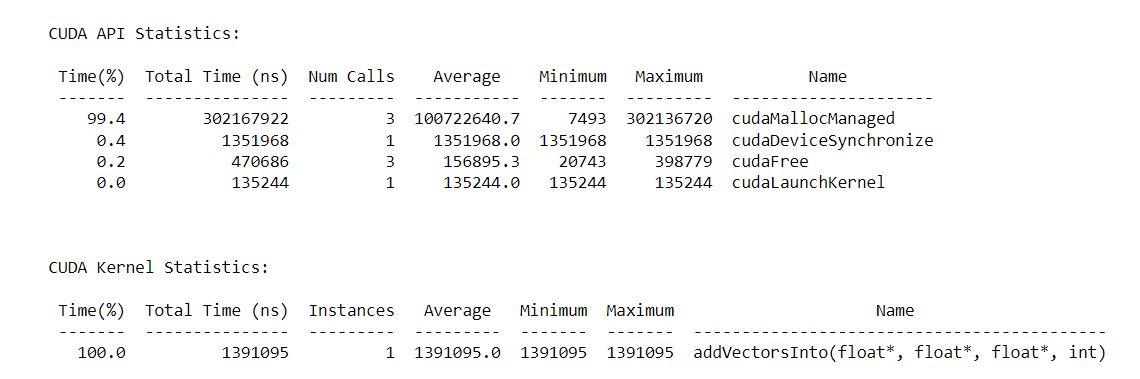
**Number of blocks: 32, Threads per block: 256, Time Required: 1406457**



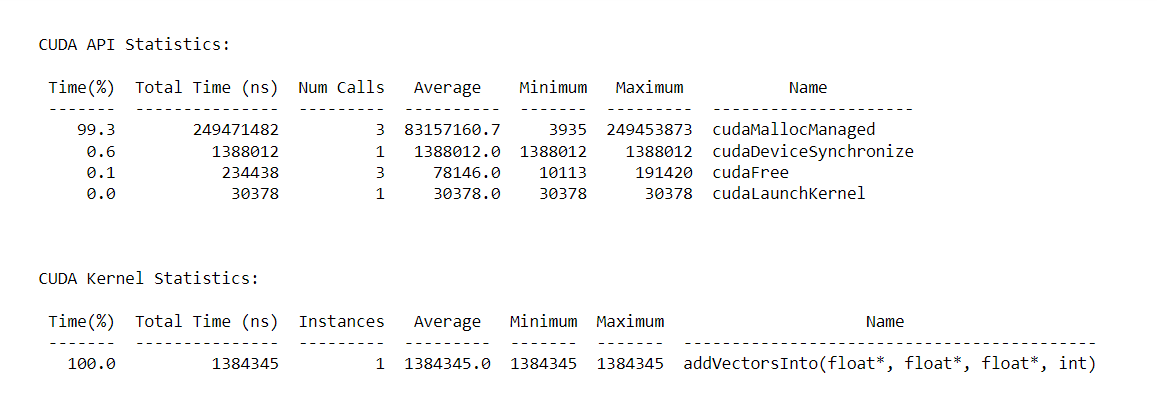
**Number of blocks: 8, Threads per block: 512, Time Required: 1567085**



**Number of blocks: 16, Threads per block: 512, Time Required: 1391095**

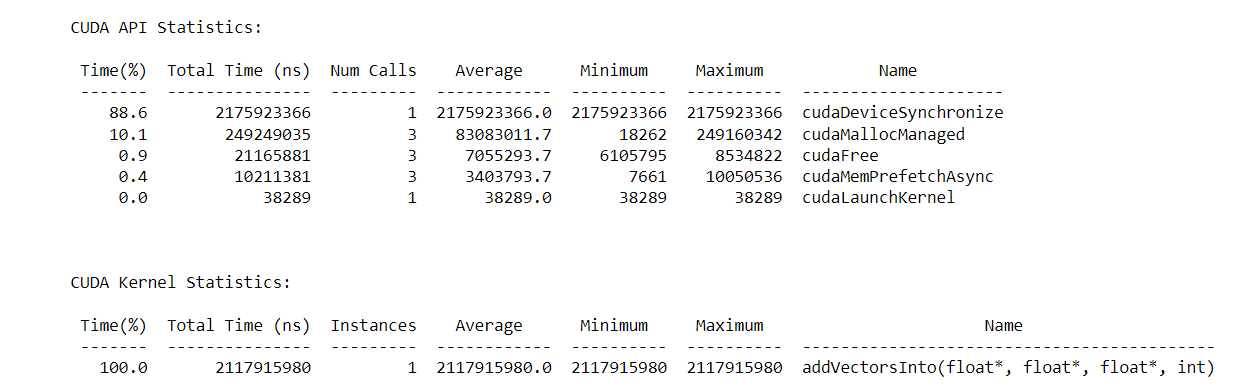


**Number of blocks: 32, Threads per block: 512, Time Required: 1384345**



**N= 2<<24**

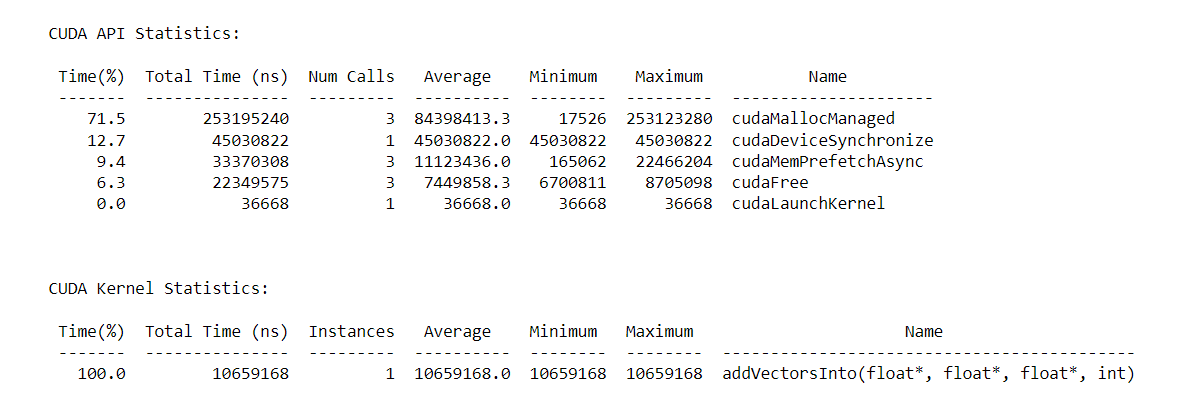
**Serial Execution Time: 2117915980**



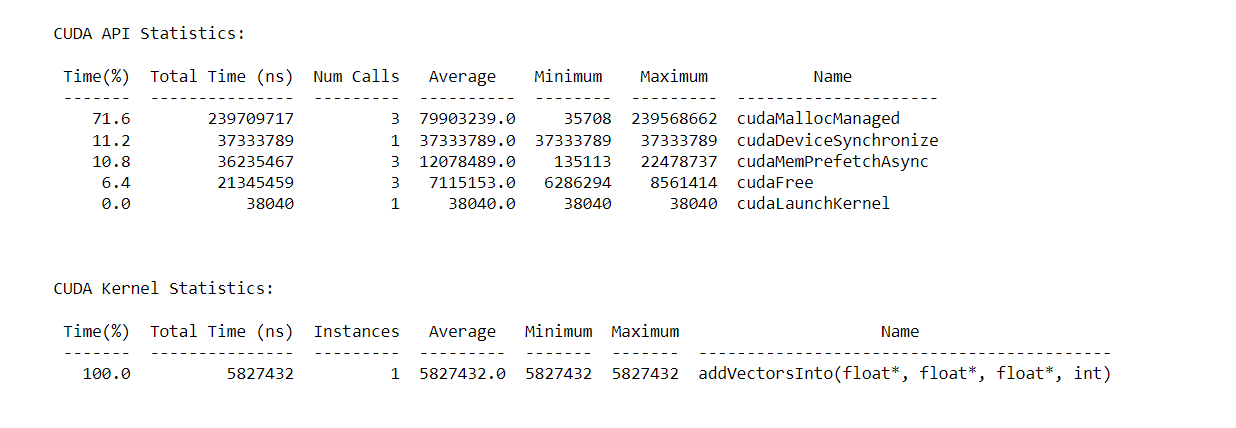
**Parallel Execution Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Blocks** | **Threads per Block** | **Time (in ns)** | **Speedup** |
| 8 | 256 | 10659168 | 198.6943 |
| 16 | 256 | 5827432 | 363.4389 |
| 32 | 256 | 3360477 | 630.2426 |
| 8 | 512 | 6078953 | 348.4014 |
| 16 | 512 | 3496636 | 605.7010 |
| 132 | 512 | 2148469 | 985.7791 |

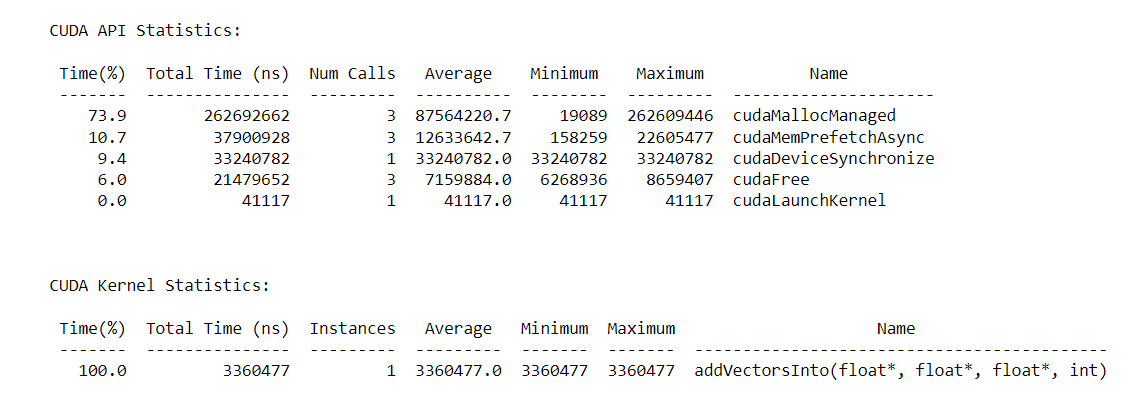
**Threads per block: 256, Number of blocks: 8, Time Required: 10659168**

****

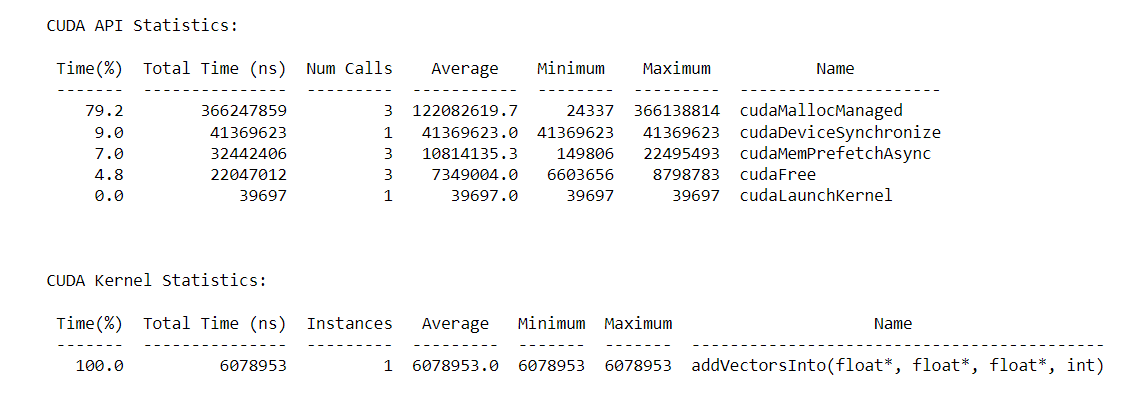
**Threads per block: 256, Number of blocks: 16, Time Required: 5827432**

****

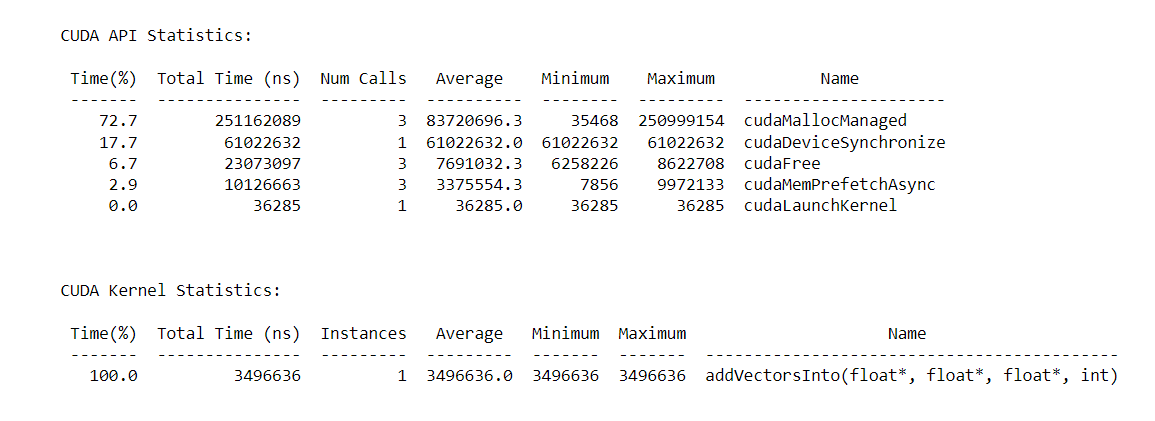
**Threads per block: 256, Number of blocks: 32, Time Required: 3360477**

****

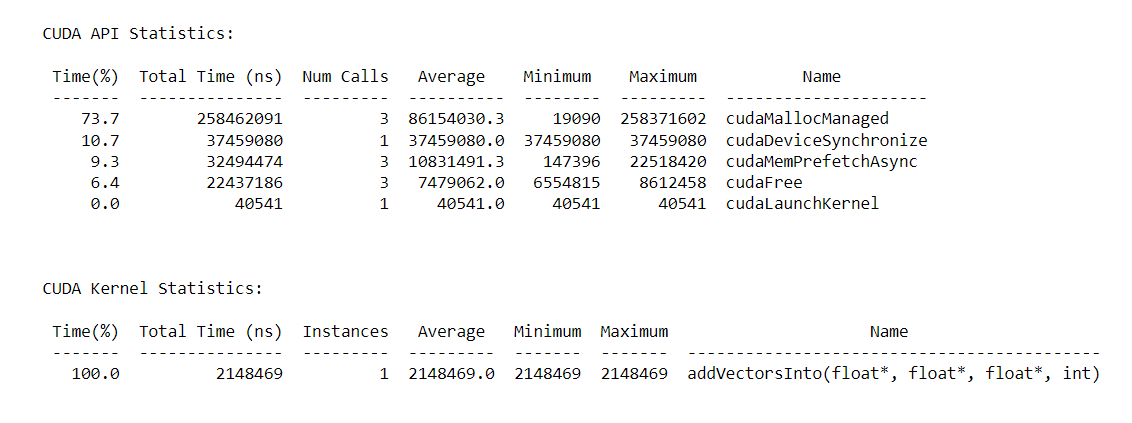
**Threads per block: 512, Number of blocks: 8, Time Required: 6078953**

****

**Threads per block: 512, Number of blocks: 16, Time Required: 3496636**

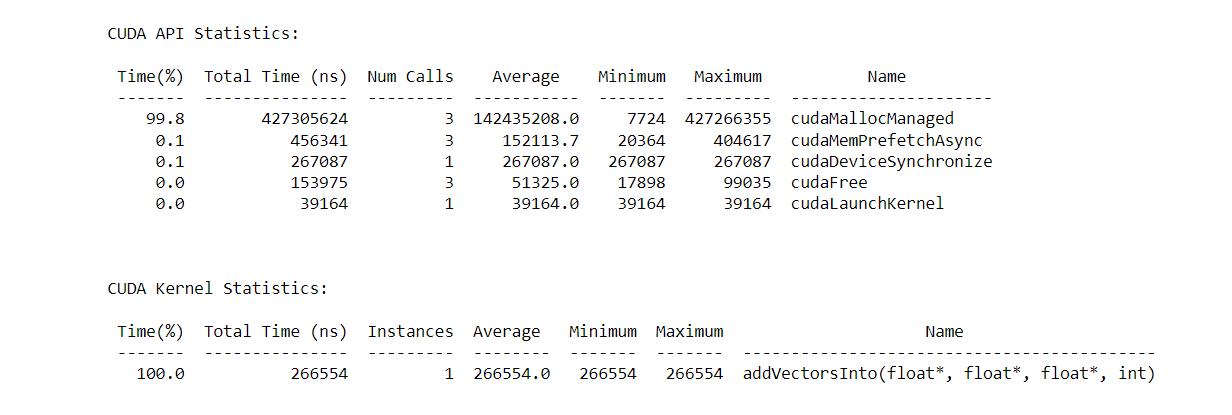
****

**Threads per block: 512, Number of blocks: 32, Time Required: 2148469**

****

**N= 2<<10**

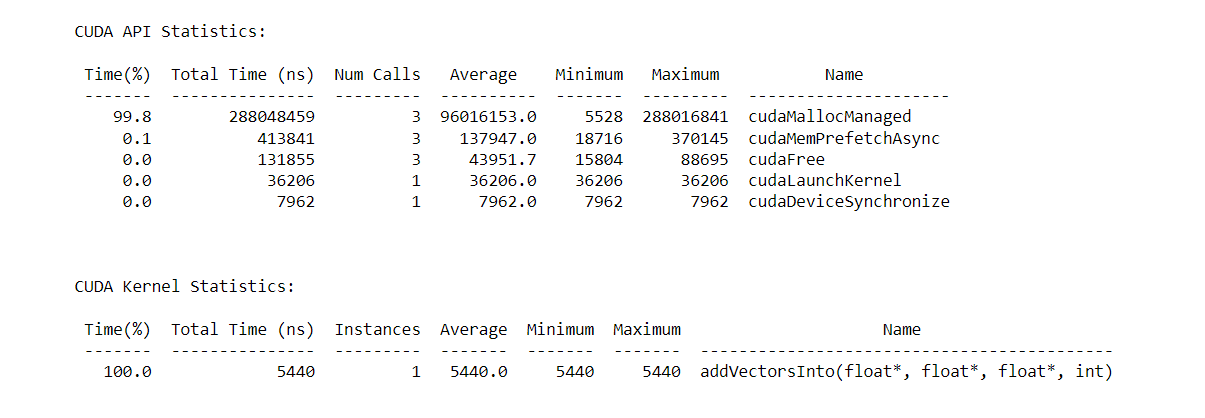
**Serial Execution Time: 266554**

****

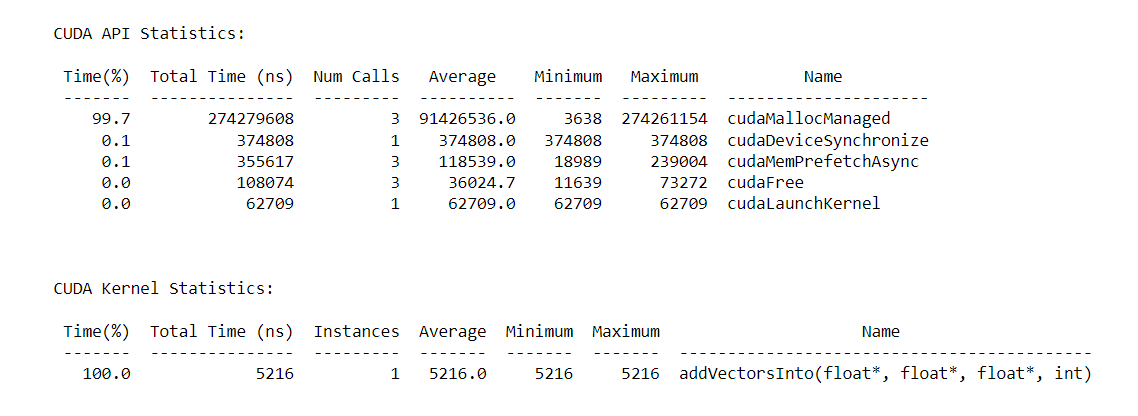
**Parallel Execution Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Blocks** | **Threads per Block** | **Time (in ns)** | **Speedup** |
| 8 | 256 | 5440 | 48.9988 |
| 16 | 256 | 5216 | 51.1031 |
| 32 | 256 | 5344 | 49.8791 |
| 8 | 512 | 5664 | 47.0610 |
| 16 | 512 | 5472 | 48.7123 |
| 132 | 512 | 5665 | 47.0527 |

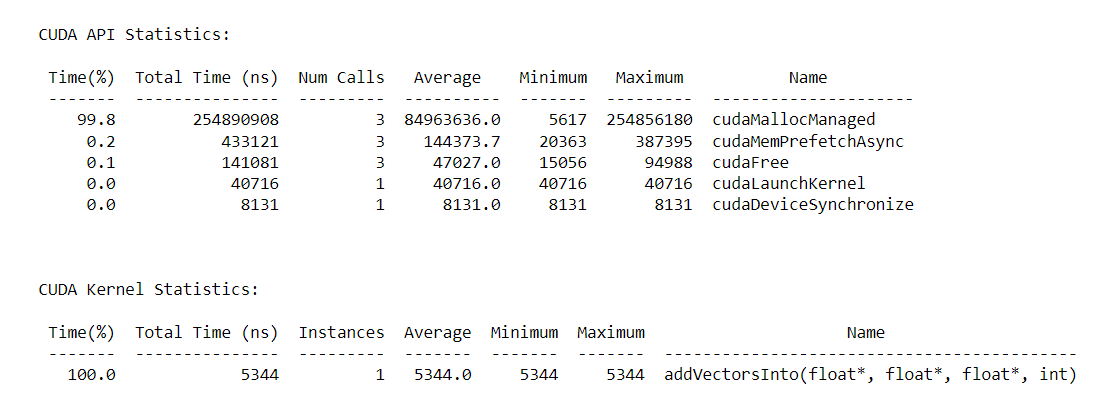
**Threads per block: 256, Number of blocks: 8, Time Required: 5440**

****

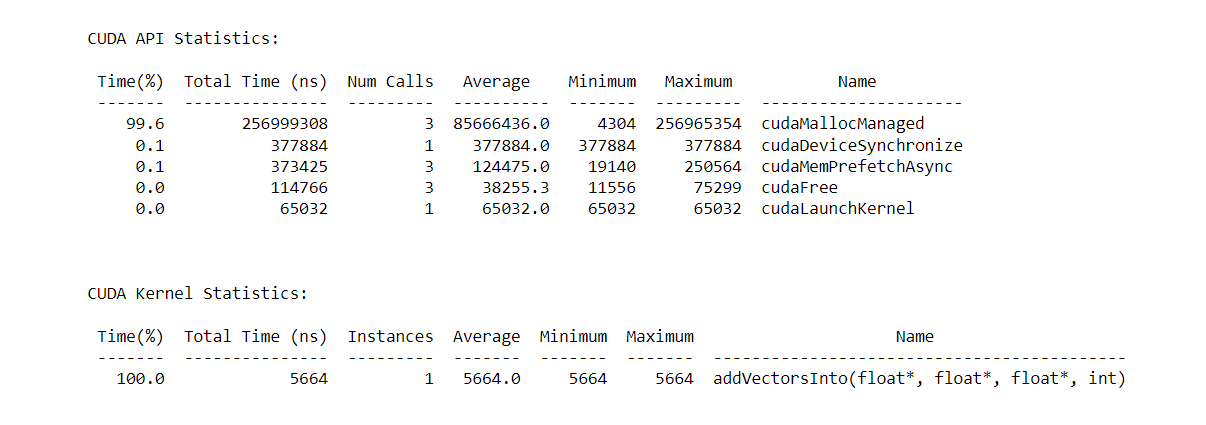
**Threads per block: 256, Number of blocks: 16, Time Required: 5216**

****

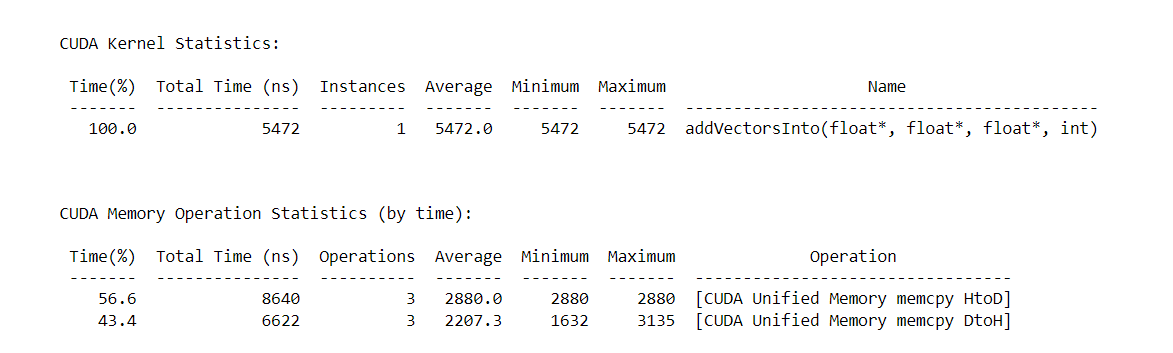
**Threads per block: 256, Number of blocks: 32, Time Required: 5344**

****

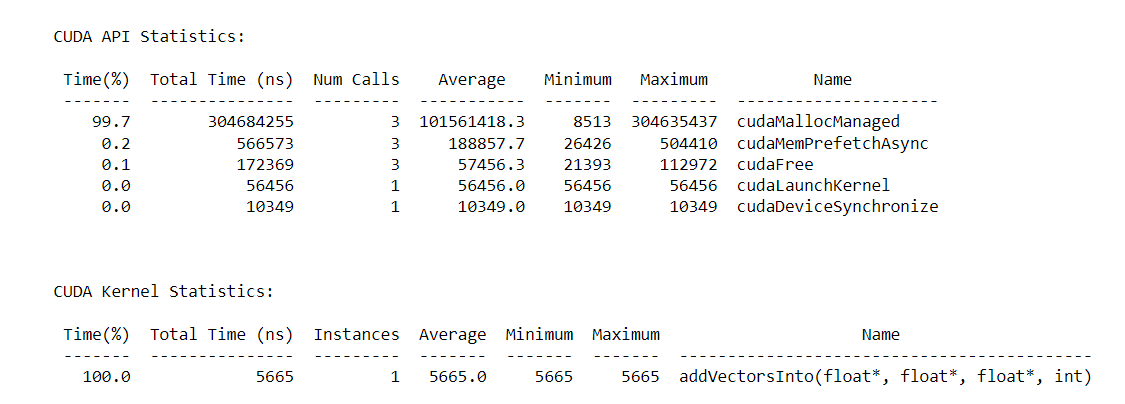
**Threads per block: 512, Number of blocks: 8, Time Required: 5664**

****

**Threads per block: 512, Number of blocks: 16, Time Required: 5472**

****

**Threads per block: 512, Number of blocks: 32, Time Required: 5665**

****

**Conclusion:**

From above graphs we can conclude that

1. For vector-vector addition problem for larger values of N time for parallel execution decreases exponentially as we increase number of threads
2. For vector-vector addition problem for smaller values of N time first decreases and then increases exponentially as we increase number of threads
3. Parallel execution is preferred for higher values of N
4. **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 tid = blockIdx.x \* blockDim.x + threadIdx.x;

if (tid < N) {

float Fx = 0, Fy = 0, Fz = 0;

for (int i = 0; i < N; i++) {

float dx = p[i].x - p[tid].x;

float dy = p[i].y - p[tid].y;

float dz = p[i].z - p[tid].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[tid].vx += dt\*Fx;

p[tid].vy += dt\*Fy;

p[tid].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;

//buf = (float \*)malloc(bytes);

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.

\*/

//int threads\_per\_block = 128;

//int number\_of\_blocks = (nBodies / threads\_per\_block);

int threads\_per\_block = 1;

int number\_of\_blocks = 1;

bodyForce <<< number\_of\_blocks, threads\_per\_block >>> ( p, dt, nBodies );

//bodyForce<<<>>>(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.

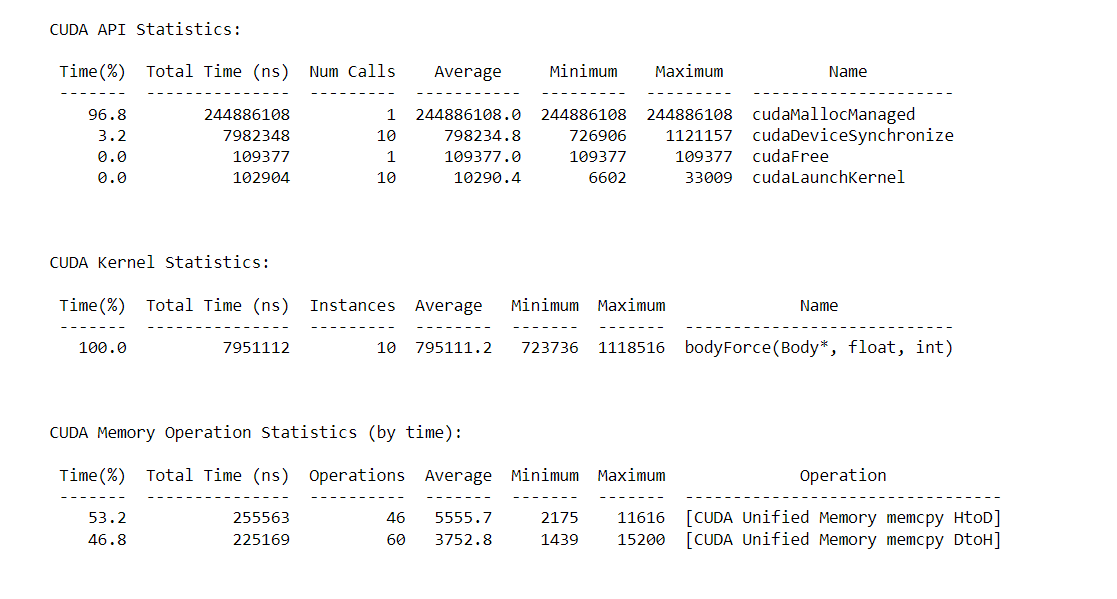
printf("%0.3f Billion Interactions / second\n", billionsOfOpsPerSecond);

cudaFree**(buf);**

**}**

**N=2<<11**

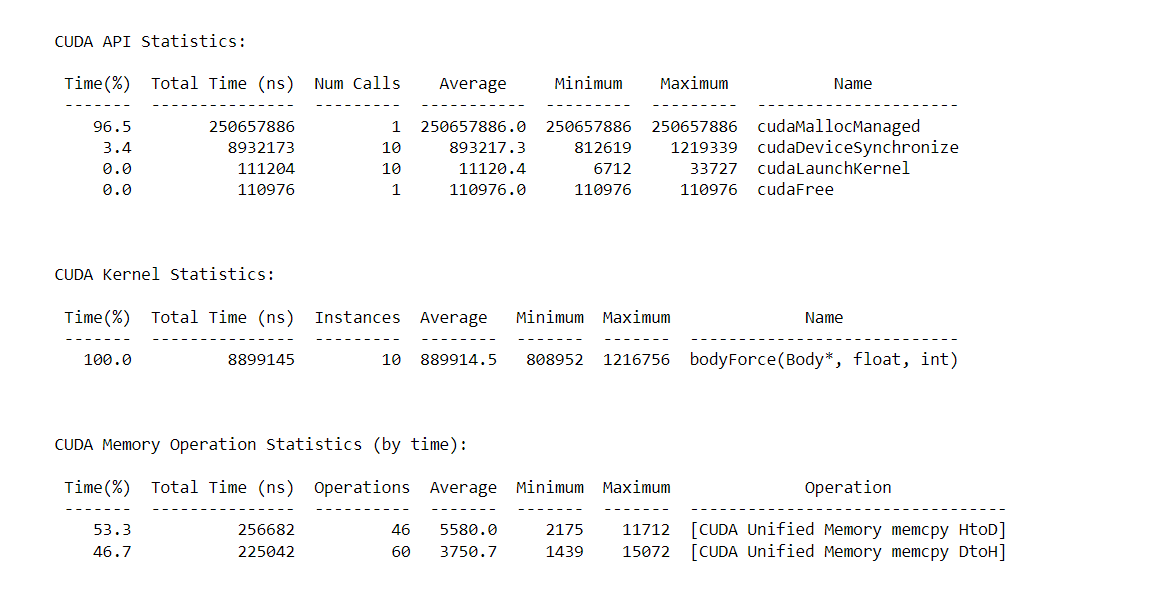
**Serial Time Execution: 7951112**



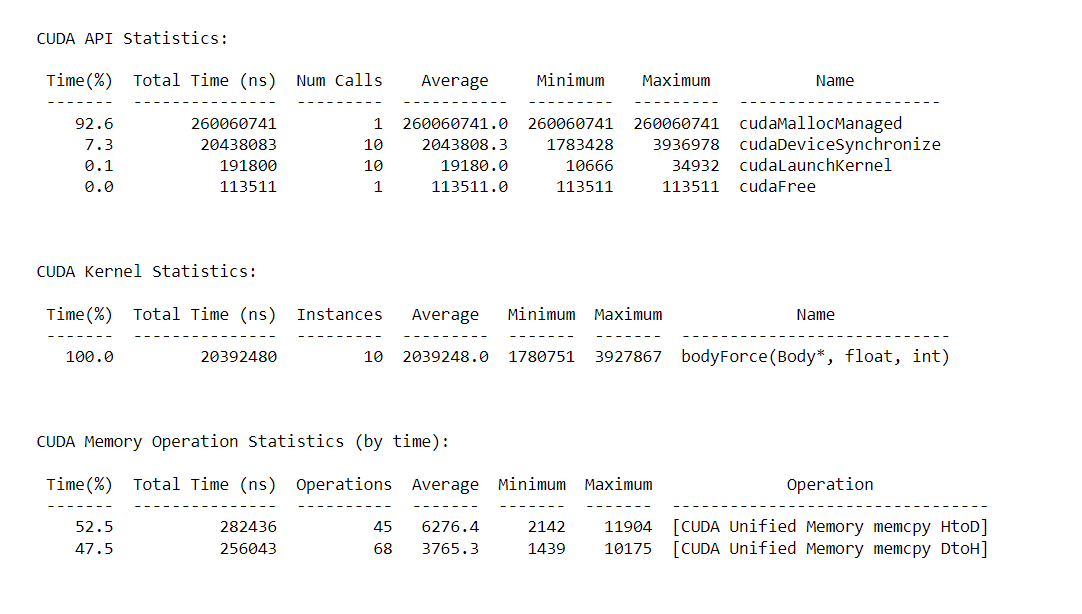
**Parallel Execution Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Blocks** | **Threads per Block** | **Time (in ns)** | **Speedup** |
| 2 | 128 | 8899145 | 0.8934 |
| 2 | 1024 | 20392480 | 0.3899 |
| 32 | 128 | 8958927 | 0.8875 |
| 32 | 1024 | 18692193 | 0.4253 |

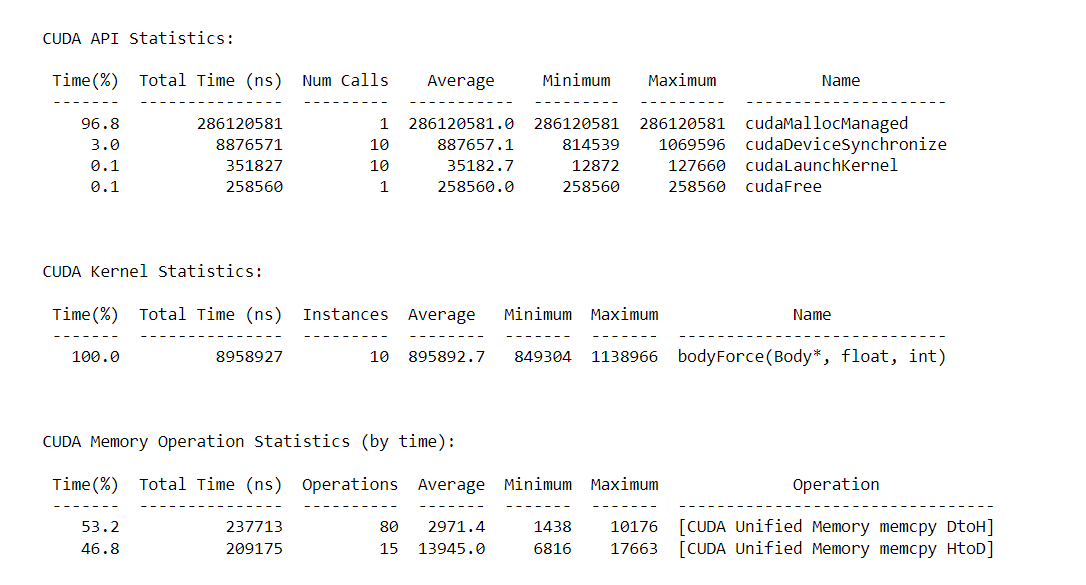
**Number of blocks: 2, Threads per block: 128, Execution Time: 8899145**



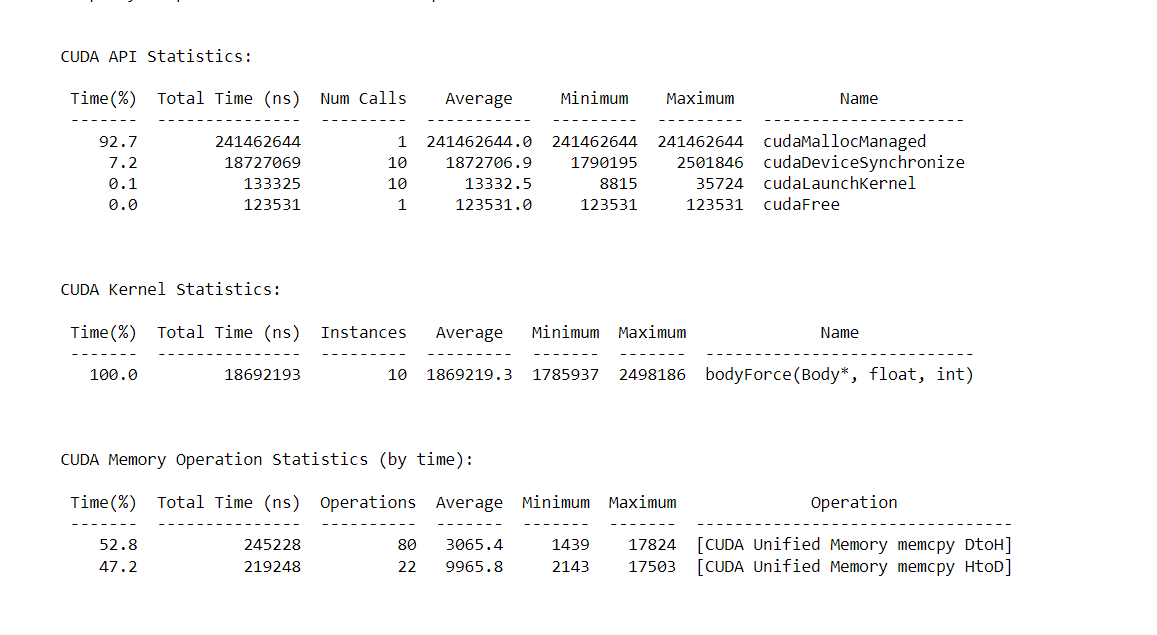
**Number of blocks: 2, Threads per block: 1024, Execution Time: 20392480**



**Number of blocks: 32, Threads per block: 128, Execution Time: 8958927**

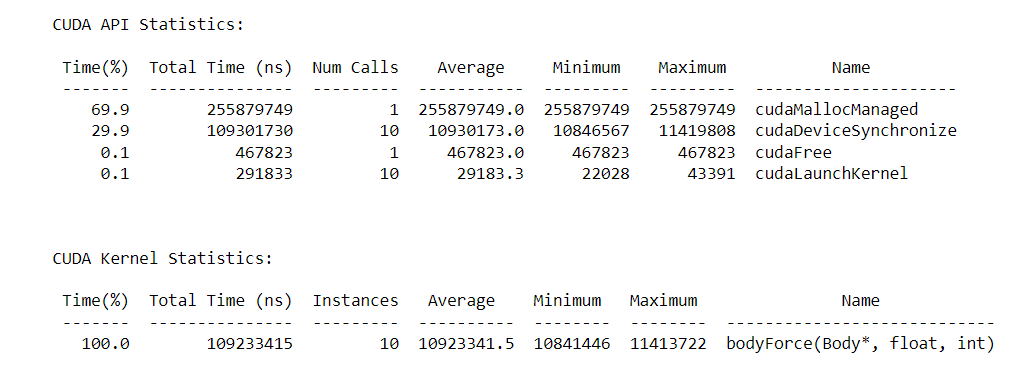


**Number of blocks: 32, Threads per block: 1024, Execution Time: 18692193**



**N=2<<15**

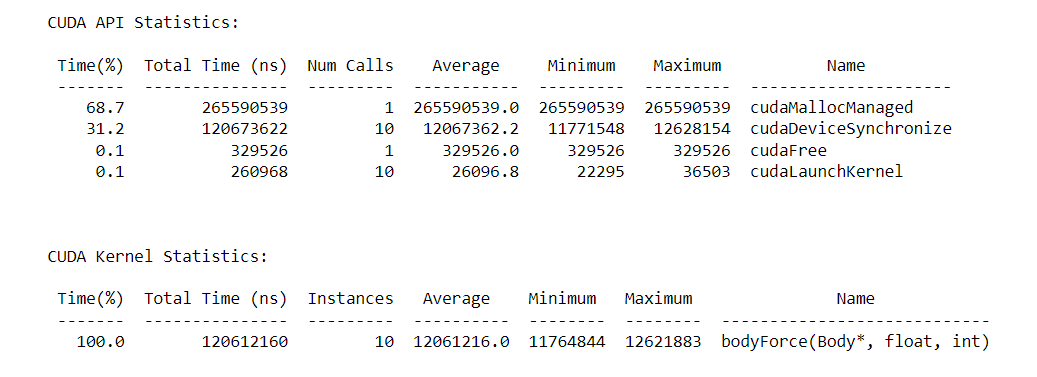
**Serial Execution Time: 109233415**

****

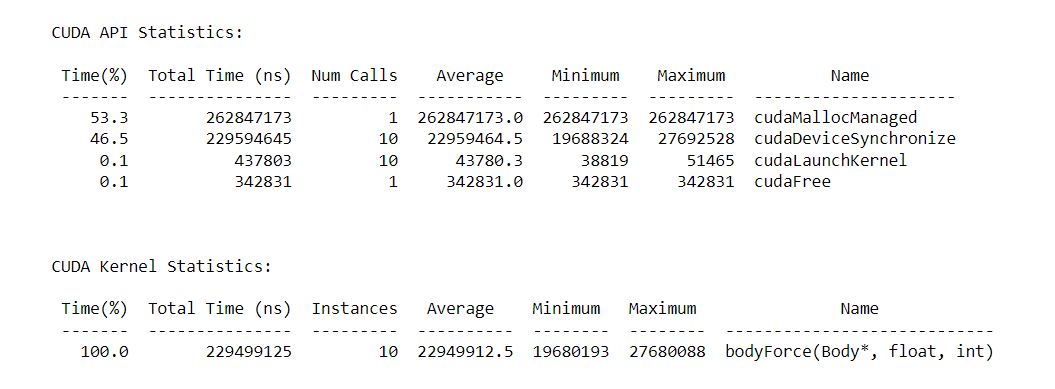
**Parallel Execution Time:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of Blocks** | **Threads per Block** | **Time (in ns)** | **Speedup** |
| 2 | 128 | 120612160 | 0.9056 |
| 2 | 1024 | 229499125 | 0.0888 |
| 32 | 128 | 116887945 | 0.9345 |
| 32 | 1024 | 264315658 | 0.4132 |

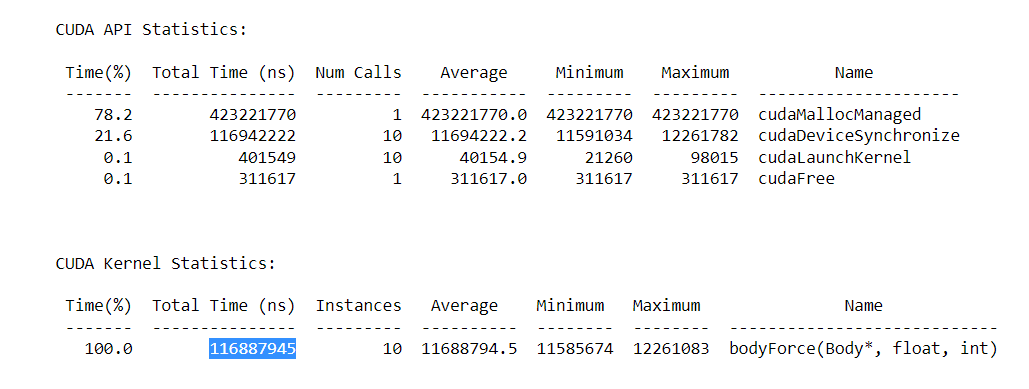
**Number of blocks: 2, Threads per block: 128, Execution Time: 120612160**

****

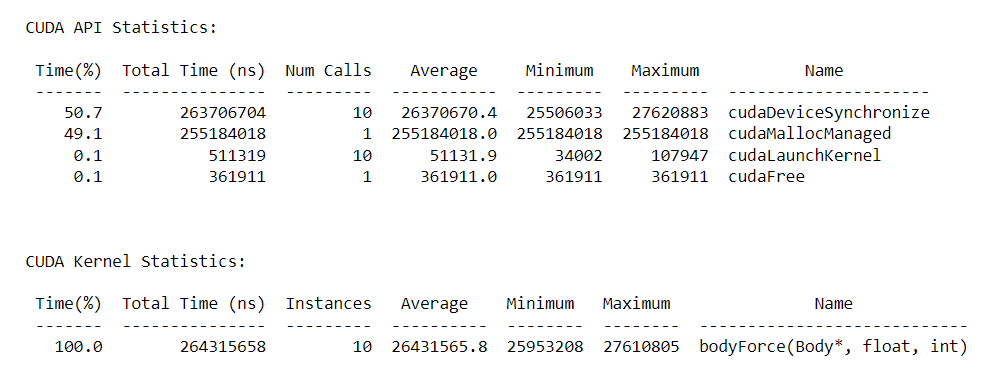
**Number of blocks: 2, Threads per block: 1024, Execution Time: 229499125**

****

**Number of blocks: 32, Threads per block: 128, Execution Time: 116887945**

****

**Number of blocks: 32, Threads per block: 1024, Execution Time: 264315658**

****

**Conclusion:**

1. From above graph we can conclude that
2. For nbody problem parallel execution time first decreases exponentially and then increases as we increase number of threads