HIGH PERFORMANCE COMPUTING PRACTICE

EXPERIMENT - 7

CUDA:

1) Hello World Program

2) Vector Addition

3) Vector Multiplication

Use input as a larger double number (64-bit).

Run experiment for Threads = {1,2,4,8,16,32,64,128,256,500}.

Estimate the parallelization fraction.

Document the report and submit.

Bhaavanaa Thumu - CED17I021

Specifications

The size of all the vectors used for below two programs is 10,00,000.

All the elements of the vectors are produced randomly and are large double numbers (64-bit).

Formulae used

Speed-up = T(1)/T(P)

Parallel fraction, f = (1 - T(P)/T(1))/(1 - (1/P))

where,

T(1) - time taken for serial execution

T(P) - time taken for parallel execution

P - number of processes/threads/processors

Other functions used

cudaMalloc - allocates a block of memory

cudaMemcpy - copies ‘n’ characters from source to destination memory area.

1. Vector addition

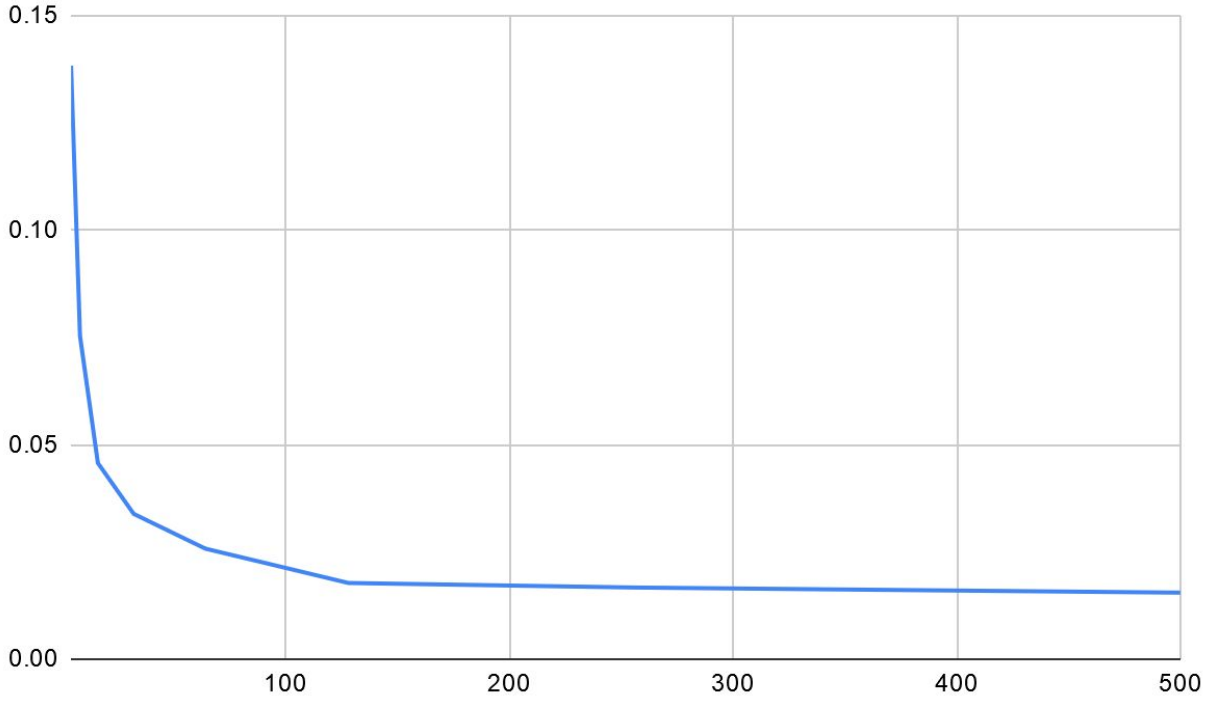
This program contains vectors of size 10,00,000.

Therefore, here, N = 10,00,000.

The number of processors initialized and the execution time in each case is as shown in the table.

No. of processors	*Execution time*	*Speed-up*	*Parallel fraction*
1	0.248387	1	#DIV/0!
2	0.247425	1.003888047	0.007745977044
4	0.138409	1.794587057	0.5903583252
8	0.075365	3.295787169	0.7960941135
16	0.045729	5.431717291	0.8702892395
32	0.033882	7.330942683	0.891449698
64	0.025777	9.635993327	0.9104481799
128	0.017763	13.98339244	0.9357975136
256	0.016700	14.87347305	0.9364241143
500	**0.015507**	**16.01773393**	**0.9394480926**

The plot below is the graph between no. of processors and execution time.



x-axis : no. of processors

y-axis : execution time

Hence, in this case, the optimal number of processors is 500. In this case-

Execution time = 0.015507

Speed-up = 16.01773393

Parallel fraction = 0.9394480926

Vector addition code

```
%%cu

#include<stdio.h>
#include<math.h>
#include <sys/time.h>

#define N 1000000

__global__ void vect_add(double *d_result, double
*d_a, double *d_b, int thread_count)
{
    int load = N/thread_count;

    if(threadIdx.x != thread_count-1)
    {
        for(int i=threadIdx.x*load;
i<(threadIdx.x+1)*load; i++)
        {
            d_result[i] = d_a[i] + d_b[i];
        }
    }
    else
    {
        for(int i=threadIdx.x*load; i<N; i++)
        {
            d_result[i] = d_a[i] + d_b[i];
```

```
        }
    }
}

int main()
{
    double *a, *b, *out, *d_a, *d_b, *d_result;
    struct timeval t1, t2;

    a = (double*)malloc(sizeof(double) * N);
    b = (double*)malloc(sizeof(double) * N);
    out = (double*)malloc(sizeof(double) * N);

    cudaMalloc((void**)&d_a, sizeof(double) * N);
    cudaMalloc((void**)&d_b, sizeof(double) * N);
    cudaMalloc((void**)&d_result, sizeof(double) * N);

    for(int i=0; i<N; i++)
    {
        a[i] = pow(2,15)+rand()+0.13246549884;
        b[i] = pow(2,15)+rand()+0.13246549884;
        // a[i] = i;
        // b[i] = i;
    }

    gettimeofday(&t1, 0);
```

```
    cudaMemcpy(d_a, a, sizeof(double) * N,
cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, sizeof(double) * N,
cudaMemcpyHostToDevice);

    int thread_count = 500;

    vect_add<<<1,thread_count>>>(d_result, d_a, d_b,
thread_count);

    cudaMemcpy(out, d_result, sizeof(double) * N,
cudaMemcpyDeviceToHost);

    gettimeofday(&t2, 0);

    /*
    for(int i=0; i<N; i++)
    {
        printf("%lf, ", out[i]);
    }
    printf("\n");
    */

    double time = (1000000.0*(t2.tv_sec-t1.tv_sec) +
t2.tv_usec-t1.tv_usec)/1000000.0;
    printf("time taken : %lf sec\n", time);
    return 0;
}
```

2. Vector multiplication

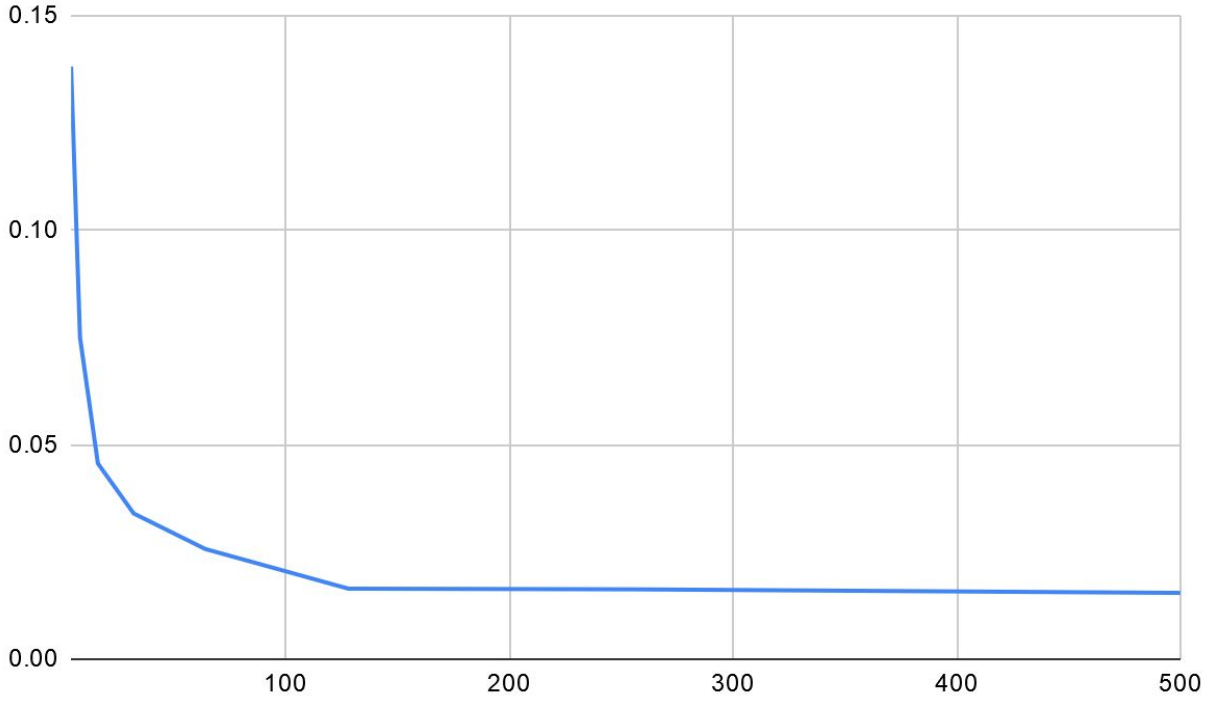
This program contains vectors of size 10,00,000.

Therefore, here, N = 10,00,000.

The number of processors initialized and the execution time in each case is as shown in the table.

No. of processors	*Execution time*	*Speed-up*	*Parallel fraction*
1	0.236728	1	#DIV/0!
2	0.229450	1.031719329	0.06148829036
4	0.138200	1.712937771	0.5549435076
8	0.075040	3.154690832	0.7805848303
16	0.045630	5.187990357	0.8610636117
32	0.033971	6.968531983	0.8841267125
64	0.025690	9.214791748	0.9056292856
128	0.016437	14.40214151	0.9378931719
256	0.016279	14.54192518	0.9348852095
500	**0.015450**	**15.32220065**	**0.9366084404**

The plot below is the graph between no. of processors and execution time.



x-axis : no. of processors

y-axis : execution time

Hence, in this case, the optimal number of processors is 500. In this case-

Execution time = 0.015450

Speed-up = 15.32220065

Parallel fraction = 0.9366084404

Vector multiplication code

```
%%cu

#include<stdio.h>
#include<math.h>
#include <sys/time.h>

#define N 1000000

__global__ void vect_mul(double *d_result, double
*d_a, double *d_b, int thread_count)
{
    int load = N/thread_count;

    if(threadIdx.x != thread_count-1)
    {
        for(int i=threadIdx.x*load;
i<(threadIdx.x+1)*load; i++)
        {
            d_result[i] = d_a[i] * d_b[i];
        }
    }
    else
    {
        for(int i=threadIdx.x*load; i<N; i++)
        {
            d_result[i] = d_a[i] * d_b[i];
```

```
        }
    }
}

int main()
{
    double *a, *b, *out, *d_a, *d_b, *d_result;
    struct timeval t1, t2;

    a = (double*)malloc(sizeof(double) * N);
    b = (double*)malloc(sizeof(double) * N);
    out = (double*)malloc(sizeof(double) * N);

    cudaMalloc((void**)&d_a, sizeof(double) * N);
    cudaMalloc((void**)&d_b, sizeof(double) * N);
    cudaMalloc((void**)&d_result, sizeof(double) * N);

    for(int i=0; i<N; i++)
    {
        a[i] = pow(2,15)+rand()+0.13246549884;
        b[i] = pow(2,15)+rand()+0.13246549884;
        // a[i] = i;
        // b[i] = i;
    }

    gettimeofday(&t1, 0);
```

```
    cudaMemcpy(d_a, a, sizeof(double) * N,
cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, sizeof(double) * N,
cudaMemcpyHostToDevice);

    int thread_count = 500;

    vect_mul<<<1,thread_count>>>(d_result, d_a, d_b,
thread_count);

    cudaMemcpy(out, d_result, sizeof(double) * N,
cudaMemcpyDeviceToHost);

    gettimeofday(&t2, 0);

    /*
    for(int i=0; i<N; i++)
    {
        printf("%lf, ", out[i]);
    }
    printf("\n");
    */

    double time = (1000000.0*(t2.tv_sec-t1.tv_sec) +
t2.tv_usec-t1.tv_usec)/1000000.0;
    printf("time taken : %lf sec\n", time);
    return 0;
}
```