High Performance Computing (CS5013)

Lab - 01

Student Details

Name: R.Abinav

Roll Number: ME23B1004

Solutions

Question 1

Code

```c
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

#define n 1000000

#define r1 1
#define r2 2
#define r3 4
#define r4 6
#define r5 8
#define r6 10
#define r7 12
#define r8 16
#define r9 20
#define r10 24
#define r11 32
#define r12 64

int main(void){
   long double *a;
   long double *b;

   //using heap mem as stack is giving overflow
   a = (long double*)malloc(n * sizeof(long double));
   b = (long double*)malloc(n * sizeof(long double));
```

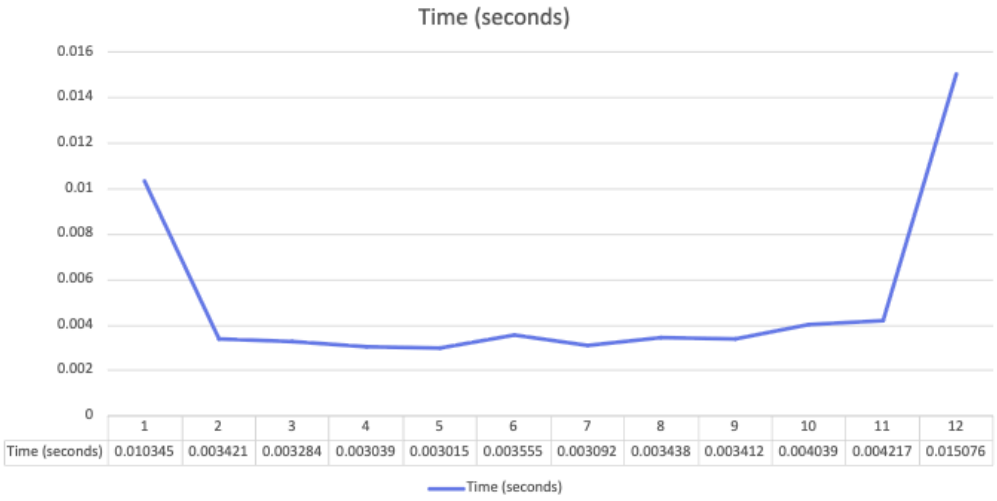
```
    //init of a and b
    for(int i=0; i<n; i++){
        a[i] = ((long double)i * i) * 100000.00L;
        b[i] = ((long double)i * i * i) * 999999.99L;
    }

    long double *c;
    c = (long double*)malloc(n * sizeof(long double));

    int threads[] = {r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12};
    int num_tests = sizeof(threads)/sizeof(threads[0]);

    for(int i=0; i<num_tests; i++){
        omp_set_num_threads(threads[i]);

        double s_time = omp_get_wtime();
        #pragma omp parallel
        {
            #pragma omp for
            for(int i=0; i<n; i++)
            {
                c[i] = a[i] + b[i];
            }
        }
        double e_time = omp_get_wtime();

        double exec_time = e_time - s_time;
        printf("%d thread(s) exec time: %f\n", threads[i], exec_time);
    }
    return 0;
}
```

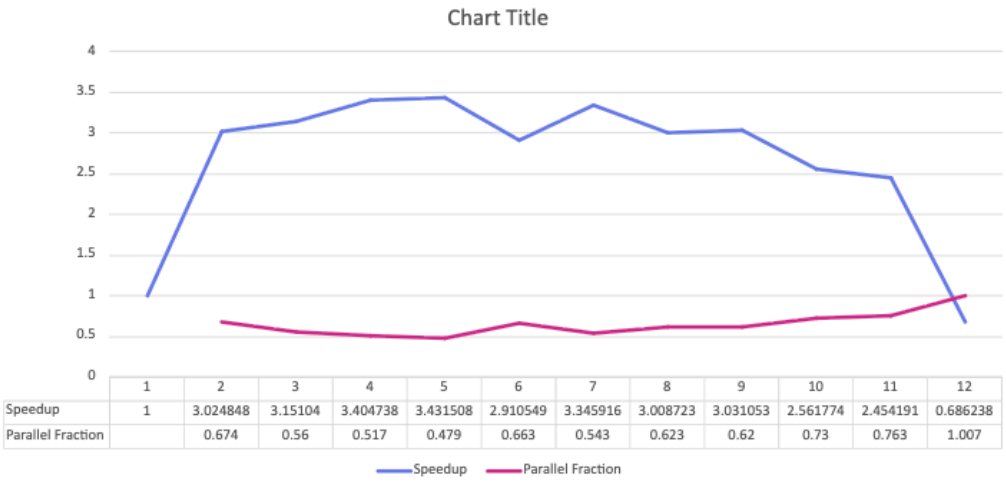
Output

```
root@b5a21ddaff0a:/tmp# ./vector_add
1 thread(s) exec time: 0.010345
2 thread(s) exec time: 0.003421
4 thread(s) exec time: 0.003284
6 thread(s) exec time: 0.003039
8 thread(s) exec time: 0.003015
10 thread(s) exec time: 0.003555
12 thread(s) exec time: 0.003092
16 thread(s) exec time: 0.003438
20 thread(s) exec time: 0.003412
24 thread(s) exec time: 0.004039
32 thread(s) exec time: 0.004217
64 thread(s) exec time: 0.015076
```

Analysis

Threads	Time (seconds)	Speedup	Parallel Fraction
1	0.010345	1	
2	0.003421	3.024848	0.674
4	0.003284	3.15104	0.56
6	0.003039	3.404738	0.517
8	0.003015	3.431508	0.479
10	0.003555	2.910549	0.663
12	0.003092	3.345916	0.543
16	0.003438	3.008723	0.623
20	0.003412	3.031053	0.62
24	0.004039	2.561774	0.73
32	0.004217	2.454191	0.763
64	0.015076	0.686238	1.007





Question 2

Code

```c
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

#define n 1000000

#define r1 1
#define r2 2
#define r3 4
#define r4 6
#define r5 8
#define r6 10
#define r7 12
#define r8 16
#define r9 20
#define r10 24
#define r11 32
#define r12 64

int main(void){
   long double *a;
   long double *b;

   //using heap mem as stack is giving overflow
   a = (long double*)malloc(n * sizeof(long double));
   b = (long double*)malloc(n * sizeof(long double));

   //init of a and b
   for(int i=0; i<n; i++){
       a[i] = ((long double)i * i) * 100000.00L;
       b[i] = ((long double)i * i * i) * 999999.99L;
   }

   long double *c;
   c = (long double*)malloc(n * sizeof(long double));

   int threads[] = {r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12};
   int num_tests = sizeof(threads)/sizeof(threads[0]);
```

```
    for(int i=0; i<num_tests; i++){
        omp_set_num_threads(threads[i]);

        double s_time = omp_get_wtime();
        #pragma omp parallel
        {
            #pragma omp for
            for(int i=0; i<n; i++)
            {
                c[i] = a[i] * b[i];
            }
        }
        double e_time = omp_get_wtime();

        double exec_time = e_time - s_time;
        printf("%d thread(s) exec time: %f\n", threads[i], exec_time);
    }
    return 0;
}
```

Output

```
root@b5a21ddaff0a:/tmp# ./vector_mul
1 thread(s) exec time: 0.045727
2 thread(s) exec time: 0.007909
4 thread(s) exec time: 0.007731
6 thread(s) exec time: 0.007660
8 thread(s) exec time: 0.007517
10 thread(s) exec time: 0.007561
12 thread(s) exec time: 0.007356
16 thread(s) exec time: 0.007831
20 thread(s) exec time: 0.007971
24 thread(s) exec time: 0.009543
32 thread(s) exec time: 0.016318
64 thread(s) exec time: 0.030382
```

Analysis

Threads	Time (seconds)	Speedup	Parallel Fraction
1	0.045727	1	
2	0.007909	5.781995	0.413
4	0.007731	5.91423	0.308
6	0.00766	5.969972	0.247
8	0.007517	6.083747	0.201
10	0.007561	6.048776	0.185
12	0.007356	6.215821	0.163
16	0.007831	5.840376	0.199
20	0.007971	5.736899	0.214
24	0.009543	4.790612	0.317
32	0.016318	2.802128	0.61
64	0.030382	1.505299	0.853