IT301 Assignment 5

NAME: SUYASH CHINTAWAR
ROLL NO.: 191IT109
TOPIC: LAB 5

Q1. Execute the program and compare the sequential and parallel executions.

SOLUTION:

Code:

```c
#include <stdio.h>
#include <sys/time.h>
#include <omp.h>
#include <stdlib.h>

int main(void){
    struct timeval TimeValue_Start;
    struct timezone TimeZone_Start;
    struct timeval TimeValue_Final;
    struct timezone TimeZone_Final;
    long time_start, time_end;
    double time_overhead;double pi,x;
    int i,N;
    pi=0.0;
    N=1000;
    gettimeofday(&TimeValue_Start, &TimeZone_Start);
    // #pragma omp parallel for private(x) reduction(+:pi) schedule(guided,5)
    for(i=0;i<=N;i++){
    x=(double)i/N;
    pi+=4/(1+x*x);
    }
    gettimeofday(&TimeValue_Final, &TimeZone_Final);
    time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv_usec;
    time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_usec;
    time_overhead = (time_end - time_start)/1000000.0;
    printf("\n\n\tTime in Seconds (T) : %lf\n",time_overhead);
    pi=pi/N;
    printf("\n \tPi is %f\n\n",pi);
}
```

Fig 1. Reference code for both sub parts

(a) Sequential Execution Result : (without uncommenting line 17)

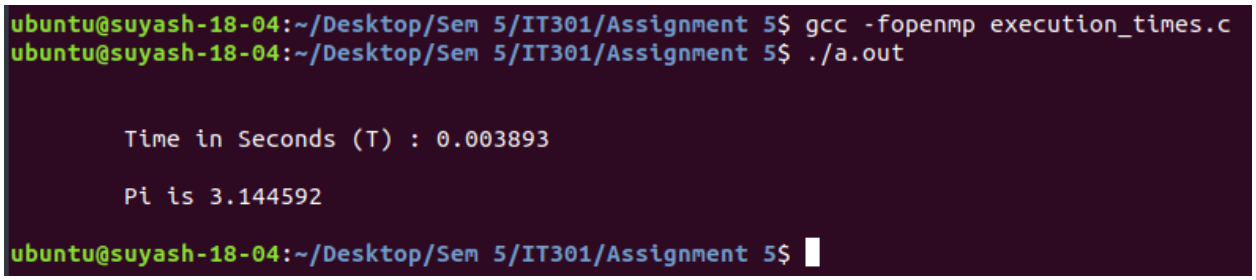
```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 5$ gcc -fopenmp execution_times.c
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 5$ ./a.out


        Time in Seconds (T) : 0.000041

        Pi is 3.144592

ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 5$
```

(b) Parallel Execution Result: (uncommenting line 17)



Observation: We can see that the execution time using parallel construct is taking more time than sequential execution. This shows that for a smaller number of iterations, openmp is not that useful. Rather, it consumes more time than sequential execution.

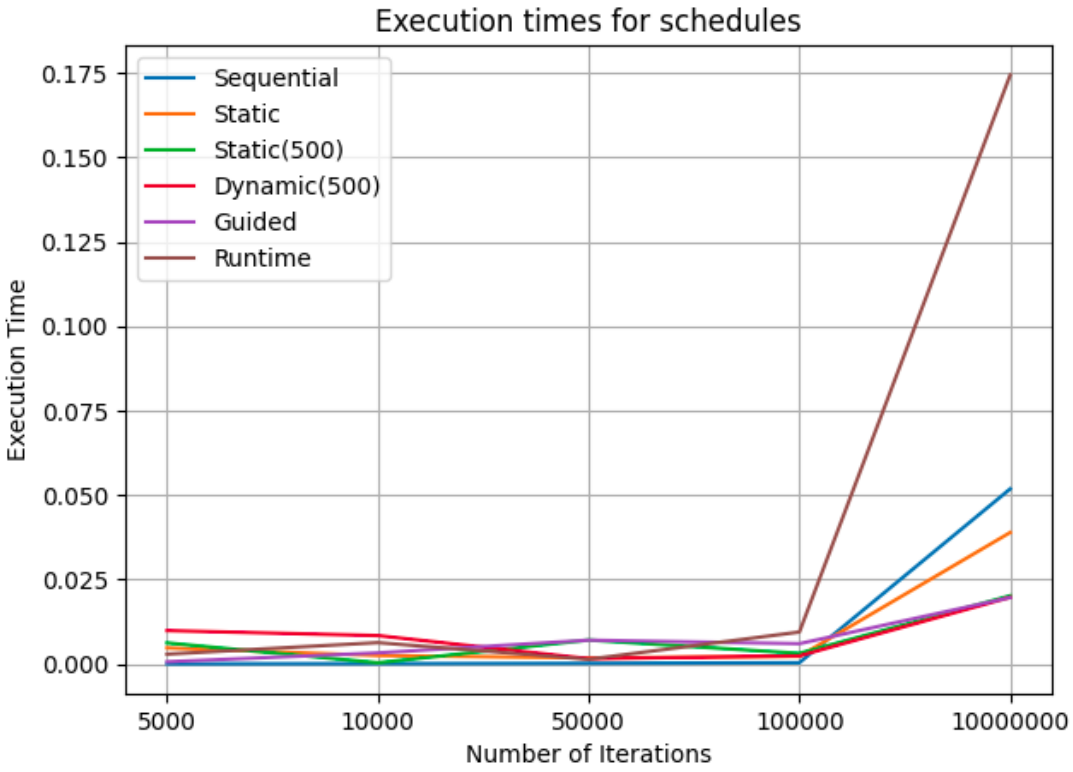
Q2. Write a sequential program to add elements of two arrays (c[i]=a[i]*b[i]. Convert the same program for parallel execution. Initialize array with random numbers. Consider an array size as 10k, 50k and 100k. Analyse the result for maximum number of threads and various schedule() function. Based on observation, perform analysis of the total execution time and explain the result by plotting the graph. [increase array size until parallel execution time is less than sequential execution.]

SOLUTION:

In the below table, columns have the number of iterations(or array size) and rows have the types of scheduling functions. Unit of time is in seconds.

schedule()	5k	10k	50k	100k	10^7
sequential	0.000057	0.000117	0.000212	0.000397	0.051843
static	0.004779	0.002529	0.001808	0.002296	0.038942
static,500	0.006256	0.000375	0.007104	0.003161	0.020212
dynamic,500	0.009899	0.008427	0.001576	0.002430	0.019770
guided	0.000672	0.003298	0.007063	0.006030	0.019681
runtime	0.002930	0.006334	0.001406	0.009476	0.174355

Plotting the above values in a graph, we get,



Average Values for each scheduling type,

Scheduling Type	Average Value
Sequential	0.010525
Static	0.010071
Static, 500	0.007421
Dynamic, 500	0.008420
Guided	0.007349
Runtime	0.038900

Observations: We can see from the table as well as the graph, that guided scheduling performs the best on large array sizes and sequential performs best on smaller arrays. Runtime scheduling gives us the maximum execution time on large arrays. Amongst static and static(500), we can see that static with chunksize 500 performs better on large arrays.
From the average values, we can see that guided gives us the best performance in terms of execution time and runtime the worst due to the high execution time in large arrays.

THANK YOU