NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 5

31st August 2021

Faculty: Dr. Geetha V

NAME: BHUVANESWAR DHARMASIVAM, ROLL NO:191IT107

1. How to compare sequential and parallel program execution times?

```
#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)
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);
}
```

SERIAL EXECUTION

```
bhuvan@bhuvan-N550JK:~/Desktop$ gcc -o time_compare_serial -fopenmp time_compare_serial.c
bhuvan@bhuvan-N550JK:~/Desktop$ ./time_compare_serial


        Time in Seconds (T) : 0.000050

        Pi is 3.144592
```

PARALLEL EXECUTION

```
bhuvan@bhuvan-N550JK:~/Desktop$ gcc -o time_compare -fopenmp time_compare.c
bhuvan@bhuvan-N550JK:~/Desktop$ ./time_compare


        Time in Seconds (T) : 0.000375

        Pi is 3.144592
```

OBSERVATION:

Here serial execution is faster because the no of iterations is low. But serial will take longer time if iterations are high

---

2. Write a sequential program to add elements of two arrays (c[i]=a[i]*b[i]. Convert the same program for parallel execution.

CODE

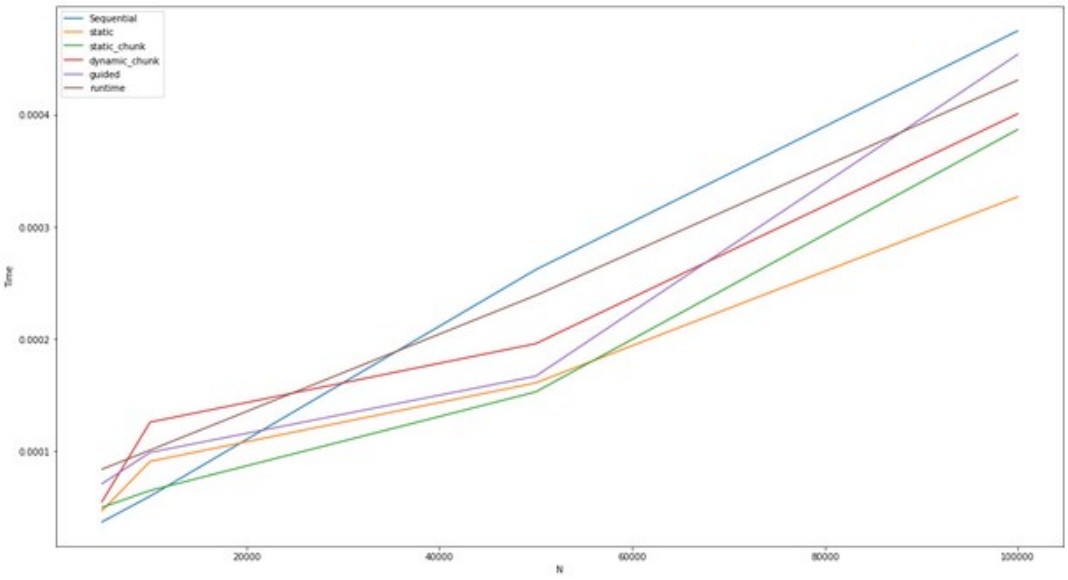
```c
#include<stdio.h>
#include<omp.h>
#include<sys/time.h>
#include<stdlib.h>
#define N 100000
int main()
{
int a[N];
int b[N];
int c[N];
struct timeval TimeValue_Start;
struct timezone TimeZone_Start;
struct timeval TimeValue_Final;
struct timezone TimeZone_Final;
long time_start, time_end;
double time_overhead;
for(int i = 0;i < N; i++) {
a[i] =rand() % N;
b[i] =rand() % N;
c[i] =0;
}


gettimeofday(&TimeValue_Start, &TimeZone_Start);
#pragma omp parallel shared(a,b,c) num_threads(8)
{
#pragma omp for schedule(guided)
for(int i = 0; i < N; i++) c[i] = a[i] + b[i];
}
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\t\t Time in Seconds (T) : %lf\n\n", time_overhead);
}
```

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.]

Schedule()	**Total Execution time for number of iterations 5K**	**Total execution for number of iterations 10K**	**Total execution for number of iterations 50K**	**Total execution for number of iterations 100K**
Sequential execution	0.000037	0.000060	0.000262	0.000475
static	0.000047	0.000091	0.000161	0.000327
Static, chunksize	0.000050	0.000065	0.000153	0.000387
Dynamic, chunksize	0.000055	0.000126	0.000196	0.000401
Guided	0.000071	0.000099	0.000167	0.000454
Runtime	0.000084	0.000101	0.000239	0.000431

GRAPH PLOT(python library - matplotlib)

```
In [27]: plt.figure(figsize=(20,11))
         plt.plot(n,seq,label="Sequential")
         plt.plot(n,static,label="static")
         plt.plot(n,static_chunk,label="static_chunk")
         plt.plot(n,dynamic_chunk,label="dynamic_chunk")
         plt.plot(n,guided,label="guided")
         plt.plot(n,runtime,label="runtime")
         plt.xlabel("N")
         plt.ylabel("Time")
         plt.legend()
         plt.show()
```



OBSERVATIONS:

Sequential takes most time for more iterations. Guided is faster for low no of iteration compared to others. Static is always fixed and it’s performance is worst compared to others overall. Static(chunksize)is very fast for low no of iterations but slow a little for more iterations. Dynamic(chunksize) initially is slow compared to other but become fast for higher iterations when compared to others.