**DEPARTMENT OF INFORMATION TECHNOLOGY, NITK SURATHKAL**

**Parallel Programming**

**LAB 5**

**28th Jan 2021**

**Note: Observe the results of each program, take the screenshot of the result and upload it in the Moodle.**

**Note:**

Q1: 2 marks : Vary the chunk size 1 mark, Vary the nuber of threads 1 mark

Q2: 2 marks: sequential time =1 mark parallel time =1 mark

Q3: Program writing 2 marks+ 5 marks for results in each row entry in table + 2 marks for graph plotting + 1 marks for explaining about your observation.

**parallel**

Forms a team of threads and starts parallel execution.

**#pragma omp parallel** *[clause[ [, ]clause] ...]*

*structured-block*

*clause*:

**if**(*scalar-expression*)

**num\_threads**(*integer-expression*)

**default**(**shared** | **none**)

**private**(*list*)

**firstprivate**(*list*)

**shared**(*list*)

**copyin**(*list*)

**reduction**(*reduction-identifier***:** *list*)

------------------------------------------------------------------------------------

**loop** Specifies that the iterations of associated loops will be

executed in parallel by threads in the team in the context

of their implicit tasks.

**#pragma omp for** *[clause[ [, ]clause] ...]*

*for-loops*

*clause*:

**private**(*list*)

**firstprivate**(*list*)

**lastprivate**(*list*)

**reduction**(*reduction-identifier***:** *list*)

**schedule**(*kind[***,** *chunk\_size]*) // this schedules iterations to threads

**collapse**(*n*)

**ordered**

**nowait**

*Refer OpenMP 4 recording and ppt slides in Moodle*

*Schedule kind***:**

• **static:** Iterations are divided into chunks of size *chunk\_size* and assigned to threads in the team in round-robin fashion in order of thread number.

• **dynamic:** Each thread executes a chunk of iterations then requests another chunk until none remain.

• **guided:** Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned.

• **auto:** The decision regarding scheduling is delegated to the compiler and/or runtime system.

• **runtime:** The schedule and chunk size are taken from the *run-sched-var* ICV.

**-------------------------------------------------------------------------------------------------------------**

**1.Write a C/C++ parallel program for adding corresponding elements of two arrays.**

/\*addarray.c\*/

#include<omp.h>

int main(){

int i,n,chunk;

int a[20],b[20],c[20];

n=20;

chunk=2;

/\*initializing array\*/

for(i=0;i<n;i++)

{ a[i]=i\*2;

b[i]=i\*3;

}

**#pragma omp parallel for default(shared) private(i) schedule(static,chunk)**

{

for(i=0;i<n;i++)

{

c[i]=a[i]+b[i];

printf(“Thread id= %d i=%d,c[%d]=%d\n”, omp\_get\_thread\_num(),i,i,c[i]);

}

}

**Check the output by varying**

1. **Chunk size**
2. **Number of threads**

**Note down the allotment of i range for each thread.**

**2. How to compare sequential and parallel program execution times. ?**

**Include following header files in the program.**

#include <sys/time.h>

#include <stdlib.h>

**//Declare following variables.**

struct timeval TimeValue\_Start;

struct timezone TimeZone\_Start;

struct timeval TimeValue\_Final;

struct timezone TimeZone\_Final;

long time\_start, time\_end;

double time\_overhead;

**Just before starting parallel region code , note down the time(start time)**

gettimeofday(&TimeValue\_Start, &TimeZone\_Start);

**After finishing parallel region, get end time.**

gettimeofday(&TimeValue\_Final, &TimeZone\_Final);

**Calculate the overhead time as follows:**

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",time\_overhead);

**Example : Run this program and compare execution time for sequential and parallel execution.**

#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);

}

**---------------------------------------------------------------------------------------------**

**3. Write a sequential program to find the smallest element in an array. Convert the same program for parallel execution. Initialise 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 |  |  |  |  |
| static |  |  |  |  |
| Static, chunksize |  |  |  |  |
| Dynamic, chunksize |  |  |  |  |
| Guided |  |  |  |  |
| runtime |  |  |  |  |

**Graph can be plotted with any tool like excel, matlab etc**