NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL

DEPARTMENT OF INFORMATION TECHNOLOGY

**IT 301 Parallel Computing LAB 3**

26th August 2020

Faculty: Dr. Geetha V and Mrs. Tanmayee

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

Execute following programs and put screen shots of the output. Write analysis of the result before uploading in IRIS as a single pdf file. for programming exercises, write the code and also put screenshot of the results.

**1. Program 1**

**Execute following code and observe the working of threadprivate directive and copyin clause:**

#include<stdio.h>

#include<omp.h>

int tid,x;

#pragma omp threadprivate(x,tid)

void main()

{

x=10;

#pragma omp parallel num\_threads(4) copyin(x)

{

tid=omp\_get\_thread\_num();

#pragma omp master

{

printf("Parallel Region 1 \n");

x=x+1;

}

#pragma omp barrier

if(tid==1)

x=x+2;

printf("Thread % d Value of x is %d\n",tid,x);

}//#pragma omp barrier

#pragma omp parallel num\_threads(4)

{

#pragma omp master

{

printf("Parallel Region 2 \n");

}

#pragma omp barrier

printf("Thread %d Value of x is %d\n",tid,x);

}

printf("Value of x in Main Region is %d\n",x);

}

**DO the following:**

**1. Remove copyin clause and check the output.**

**22. Remove copyin clause and initialize x globally.**

**Note the observation about threadprivate directive and copyin clause.**

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

**2. Program 2**

**Learn the concept of firstprivate() and threadprivate()**

int main (void) {

int x=10, y=20,a[10],b[10],c[10],i;

//int count=0;

for(i=0;i<10;i++)

b[i]=c[i]=i;

printf("1. count=%d\n",count);

#pragma omp parallel num\_threads(2) copyin(count)

{

#pragma omp for schedule(static,5) firstprivate(x)

for(i=0;i<10;i++)

{

int tid1=omp\_get\_thread\_num();

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

count++;

x++;

printf("tid=%d,a[%d]=%d, count=%d x=%d\n",tid1,i,a[i],count,x);

}

#pragma omp barrier

printf("2. before copyprivate count=%d x=%d tid=%d\n",count,x,omp\_get\_thread\_num());

#pragma omp single copyprivate(count)

{

count=count+20;

}

printf("3. after copyprivate count=%d x=%d tid=%d\n",count,x,omp\_get\_thread\_num());

#pragma omp for schedule(static,5) firstprivate(x)

for(i=0;i<10;i++)

{

int tid1=omp\_get\_thread\_num();

a[i]=b[i]\*c[i];

count++;

x++;

printf("tid=%d,a[%d]=%d, count=%d, x=%d\n",tid1,i,a[i],count,x);

}

}

#pragma omp barrier

printf("4. count=%d x=%d\n",count,x);

printf("\n");

return 0;

}

**Analyse the results for variable count and x.**

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

**3. Program to understand the concept of collapse()**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

int main (void) {

int i,j;

#pragma omp parallel

{

#pragma omp for schedule(static,3) private(i,j)

for(i=0;i<6;i++)

for(j=0;j<5;j++)

{

int tid2=omp\_get\_thread\_num();

printf("tid=%d, i=%d j=%d\n",omp\_get\_thread\_num(),i,j);

}

}

return 0;

}

**Consider three for loops and check the result with no collapse(), collapse(2) and collapse(3).**

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

**4. 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**

#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.**

**I used number of threads = 100.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | Total execution for number of iterations 1000K |
| Sequential execution | 0.000045 | 0.000017 | 0.000087 | 0.000175 | 0.002312 |
| static | 0.009789 | 0.000006 | 0.000016 | 0.000372 | 0.000839 |
| Static, 10 | 0.005215 | 0.000969 | 0.000018 | 0.000033 | 0.003233 |
| Dynamic, 10 | 0.009690 | 0.000022 | 0.000260 | 0.000200 | 0.002036 |
| Guided | 0.002575 | 0.000011 | 0.000024 | 0.000039 | 0.000442 |
| runtime | 0.005931 | 0.000206 | 0.001019 | 0.006609 | 0.020606 |