PADC

Lab - 3

K. Manoj Kumar

2022BCS0184

Q1. Write an OpenMP program with C++ that estimates the value of pi (π) using a following function and apply rectangle rule.

Case 1: Value of x is the starting point in every rectangle. Calculate the leftsum of each rectangle on the curve and do sum of all. Observe the error value with actual pi (π). Case 2: Value of x is the end point in every rectangle. Calculate the rightsum of each rectangle on the curve and do sum of all. Observe the error value with actual pi (π). Case 3: Value of x is the middle point in every rectangle. Calculate the midsum of each rectangle on the curve and do sum of all. Observe the error value with actual pi (π).

Program :

```
#define _USE_MATH_DEFINES
#include<bits/stdc++.h>
#include<omp.h>

using namespace std;

double estimatePi_serial(int n) {
    double width = 1.0 / n;
    double area = 0.0;

    for (int i = 0; i < n; i++) {
        double mid = (i + 0.5) * width;
```

```
        double height = 4.0 / (1 + (mid * mid));
        area += height * width;
    }

    return area;
}

double estimatePi_serial_leftEndPoint(int n) {
    double width = 1.0 / n;
    double area = 0.0;

    for (int i = 0; i < n; i++) {
        double x = i * width;
        double height = 4.0 / (1 + (x * x));
        area += height * width;
    }

    return area;
}
double estimatePi_serial_rightEndPoint(int n) {
    double width = 1.0 / n;
    double area = 0.0;

    for (int i = 0; i < n; i++) {
        double x = (i + 1) * width;
        double height = 4.0 / (1 + (x * x));
```

```
        area += height * width;
    }

    return area;
}
double estimatePi_parallel_atomic(int n) {
    double width = 1.0 / n;
    double area = 0.0;

    #pragma omp parallel
    {
        double localArea = 0.0;

        #pragma omp for
        for (int i = 0; i < n; i++) {
            double mid = (i + 0.5) * width;
            double height = 4.0 / (1 + (mid * mid));
            localArea += height * width;
        }



        #pragma omp atomic
        area += localArea;
```

```
    }

    return area;
}
double estimatePi_parallel_critical(int n) {
    double width = 1.0 / n;
    double area = 0.0;

    #pragma omp parallel
    {
        double localArea = 0.0;

        #pragma omp for
        for (int i = 0; i < n; i++) {
            double mid = (i + 0.5) * width;
            double height = 4.0 / (1 + (mid * mid));
            localArea += height * width;
        }

        #pragma omp critical
        {
            area += localArea;
          cout << "Thread " << omp_get_thread_num() << " calculated localArea: "
<< localArea << endl;

        }
```

```
    }

    return area;
}
double estimatePi_parallel_reduction(int n) {
    double width = 1.0 / n;
    double area = 0.0;

    #pragma omp parallel for reduction(+ : area)
    for (int i = 0; i < n; i++) {
        double mid = (i + 0.5) * width;
        double height = 4.0 / (1 + (mid * mid));
        area += height * width;
    }

    return area;
}

void run(int n) {
    cout << "############ " << "For n :" << n << " ###################\n";
    omp_set_num_threads(5);
    // double PI = 3.141414141;
    double s = omp_get_wtime();
```

```
    double area = estimatePi_serial(n);
    double e = omp_get_wtime();
    cout << "Value of pi (serial): " << area << endl;
    cout << "Execution time (serial): " << (e - s) << " micro seconds" << endl;
    double area_left = estimatePi_serial_leftEndPoint(n);
    cout << "value of pi with left most point :" << area_left << endl;
    double right_area = estimatePi_serial_rightEndPoint(n);
    cout << "Pi value(right end point) :" << right_area << endl;
    cout << "error value with mid point :" << abs( M_PI - area) << endl;
    cout << "error value with left :" <<abs(area_left - M_PI) << endl;
    cout << "error value with right :" << abs(M_PI - right_area) << endl;



    s = omp_get_wtime();
    area = estimatePi_parallel_atomic(n);
    e = omp_get_wtime();
    cout << "Value of pi (parallel with atomic): " << area << endl;
    cout << "Execution time (parallel with atomic): " << (e - s) << " micro
seconds" << endl;
    s = omp_get_wtime();
    area = estimatePi_parallel_critical(n);
    e = omp_get_wtime();
    cout << "Value of pi (parallel with critical): " << area << endl;
    cout << "Execution time (parallel with critical): " << (e - s) << " micro
seconds" << endl;
    s = omp_get_wtime();
    // auto st = std::chrono::high_resolution_clock::now();
```

```
    area = estimatePi_parallel_reduction(n);
    // auto end = std::chrono::high_resolution_clock::now();
    e = omp_get_wtime();
    // std::chrono::duration<double> d = end - st;
    cout << "Value of pi (parallel with reduction): " << area << endl;
    cout << "Execution time (parallel with reduction): " << e - s << " micro seconds" << endl;
    cout << "<----------------------------------------------------------------->\n";
    cout << "\n";
}

int main() {
    vector<int> n = {10, 50, 100, 500, 1000};
    for(int num : n) {
        run(num);
    }
    return 0;
}
```

Output :

```
############ For n :10 ##################
Value of pi (serial): 3.14243
Execution time (serial): 0 micro seconds
value of pi with left most point :3.23993
Pi value(right end point) :3.03993
error value with mid point :0.000833331
```

error value with left :0.0983333

error value with right :0.101667

Value of pi (parallel with atomic): 3.14243

Execution time (parallel with atomic): 0.00400019 micro seconds

Thread 1 calculated localArea: 0.732818

Thread 2 calculated localArea: 0.639742

Thread 3 calculated localArea: 0.537195

Thread 4 calculated localArea: 0.44247

Thread 0 calculated localArea: 0.790201

Value of pi (parallel with critical): 3.14243

Execution time (parallel with critical): 0.00499988 micro seconds

Value of pi (parallel with reduction): 3.14243

Execution time (parallel with reduction): 0 micro seconds

<-->

############ For n :50 ##################

Value of pi (serial): 3.14163

Execution time (serial): 0 micro seconds

value of pi with left most point :3.16153

Pi value(right end point) :3.12153

error value with mid point :3.33333e-05

error value with left :0.0199333

error value with right :0.0200667

Value of pi (parallel with atomic): 3.14163

Execution time (parallel with atomic): 0 micro seconds

Thread 1 calculated localArea: 0.732458

Thread 2 calculated localArea: 0.639656

Thread 3 calculated localArea: 0.537282

Thread 4 calculated localArea: 0.442623

Thread 0 calculated localArea: 0.789607

Value of pi (parallel with critical): 3.14163

Execution time (parallel with critical): 0.00300002 micro seconds

Value of pi (parallel with reduction): 3.14163

Execution time (parallel with reduction): 0 micro seconds

<---------->

############ For n :100 ##################

Value of pi (serial): 3.1416

Execution time (serial): 0 micro seconds

value of pi with left most point :3.15158

Pi value(right end point) :3.13158

error value with mid point :8.33333e-06

error value with left :0.00998333

error value with right :0.0100167

Value of pi (parallel with atomic): 3.1416

Execution time (parallel with atomic): 0.000999928 micro seconds

Thread 4 calculated localArea: 0.442627

Thread 1 calculated localArea: 0.732447

Thread 2 calculated localArea: 0.639653

Thread 3 calculated localArea: 0.537285

Thread 0 calculated localArea: 0.789588

Value of pi (parallel with critical): 3.1416

Execution time (parallel with critical): 0.00100017 micro seconds

Value of pi (parallel with reduction): 3.1416

Execution time (parallel with reduction): 0 micro seconds

<-->

############ For n :500 ##################

Value of pi (serial): 3.14159

Execution time (serial): 0 micro seconds

value of pi with left most point :3.14359

Pi value(right end point) :3.13959

error value with mid point :3.33333e-07

error value with left :0.00199933

error value with right :0.00200067

Value of pi (parallel with atomic): 3.14159

Execution time (parallel with atomic): 0 micro seconds

Thread 2 calculated localArea: 0.639653

Thread 4 calculated localArea: 0.442629

Thread 3 calculated localArea: 0.537286

Thread 1 calculated localArea: 0.732443

Thread 0 calculated localArea: 0.789582

Value of pi (parallel with critical): 3.14159

Execution time (parallel with critical): 0.00200009 micro seconds

Value of pi (parallel with reduction): 3.14159

Execution time (parallel with reduction): 0 micro seconds

<-->

############ For n :1000 ##################

Value of pi (serial): 3.14159

Execution time (serial): 0 micro seconds

value of pi with left most point :3.14259

Pi value(right end point) :3.14059

error value with mid point :8.33333e-08

error value with left :0.000999833

error value with right :0.00100017

Value of pi (parallel with atomic): 3.14159

Execution time (parallel with atomic): 0 micro seconds

Thread 2 calculated localArea: 0.639653

Thread 4 calculated localArea: 0.442629

Thread 3 calculated localArea: 0.537286

Thread 1 calculated localArea: 0.732443

Thread 0 calculated localArea: 0.789582

Value of pi (parallel with critical): 3.14159

Execution time (parallel with critical): 0.000999928 micro seconds

Value of pi (parallel with reduction): 3.14159

Execution time (parallel with reduction): 0 micro seconds

<---------->

Q2. Write an openMP program with C++ that estimates the value of pi (π) using MonteCarlo simulation. a) Write the serial version program to estimate the value of pi (π). Test the result with classical integration value. Calculate the execution time by using the library function omp_get_wtime(). Write the parallel version program to estimate the same. Test the result with classical integration value and by (a). It includes number of threads involved and the area calculated by which thread number. Calculate the execution time by using

the library function omp_get_wtime(). b) Identify the line of statement which leads the race condition. Race condition occurs when the multiple threads accessing a shared variable. If it exists how will you handle this problem? Use appropriate OpenMP directives/clauses such as critical, atomic, reduction and find the solution. Test the result with classical integration value and by (a) and (b). Calculate the execution time for critical, atomic, reduction clauses by using the library function omp_get_wtime().

Program :

```
//q2
#include<bits/stdc++.h>
#include<omp.h>
// #define points 100000
using namespace std;

double monteCarlo_serial(int points) {
   int circlePoints = 0;
   srand(time(0));

   for(int i = 0;i < (points) ;i++) {
      double randX = (double)rand() / RAND_MAX;
      double randY = (double)rand() / RAND_MAX;
      if((randX * randX) + (randY * randY) <= 1.0) circlePoints++;

   }
      return 4.0 * circlePoints / points;
}
double monteCarlo_parallel(int points) {
```

```
    int circlePoints = 0;
    srand(time(0));
    #pragma omp parallel reduction(+ : circlePoints)
    {
        #pragma omp for
        for(int i = 0;i < (points) ;i++) {
            double randX = (double)rand() / RAND_MAX;
            double randY = (double)rand() / RAND_MAX;
            if((randX * randX) + (randY * randY) <= 1.0)
                circlePoints++;


        }
    }
        return 4.0 * circlePoints / points;
}
double monteCarlo_parallel_atomic(int points) {
    int circlePoints = 0;
    srand(time(0));
    #pragma omp parallel
    {
        #pragma omp for
        for(int i = 0;i < (points) ;i++) {
            double randX = (double)rand() / RAND_MAX;
            double randY = (double)rand() / RAND_MAX;
            if((randX * randX) + (randY * randY) <= 1.0)
            #pragma omp atomic
```

```
            circlePoints++;

        }
    }
        return 4.0 * circlePoints / points;
}
double monteCarlo_parallel_critical(int points) {
    int circlePoints = 0;
    srand(time(0));
    #pragma omp parallel
    {
        #pragma omp for
        for(int i = 0;i < (points) ;i++) {
            double randX = (double)rand() / RAND_MAX;
            double randY = (double)rand() / RAND_MAX;
            if((randX * randX) + (randY * randY) <= 1.0)
            #pragma omp critical
            circlePoints++;



        }
    }
        return 4.0 * circlePoints / points;
}

void run(int n) {
```

```
   cout << "for n :" << n << "------------------->\n";
 double s = omp_get_wtime();
   // cin >> side ;
   double result = monteCarlo_serial(n);
   double e = omp_get_wtime();
   cout << "execution time(serial) : " << e - s <<" seconds" << endl;
   cout << "Serial value: " << result << endl;
   s = omp_get_wtime();
   result = monteCarlo_parallel(n);
   e = omp_get_wtime();
   cout << "pi value for parallel(reduction):" << result << " Execution
time(reduction) :" << (e - s) * 1.0e6 << " seconds\n";
   // cout << "s\n";
   s = omp_get_wtime();
   result = monteCarlo_parallel_atomic(n);
   e = omp_get_wtime();
   cout << "pi value:(atomic) " << result << " Execution time:" << (e - s)* 1.0e9
<< endl;
   s = omp_get_wtime();
   result = monteCarlo_parallel_critical(n);
   e = omp_get_wtime();
   cout << "Pi value:(critical) " << result << " Execution time:" << (e - s)* 1.0e9
<< endl;
   cout << "<-------------------------------------------------------------->\n";

}
```

```
int main() {
    omp_set_num_threads(5);
    vector<int> n = {150,500,10000};
    for(int i : n) {
        run(i);
    }

}
```

Output :

```
for n :150------------------->
execution time(serial) : 0 seconds
Serial value: 3.30667
pi value for parallel(reduction):3.06667 Execution time(reduction) :3000.02 seconds
pi value:(atomic) 3.28 Execution time:0
Pi value:(critical) 3.17333 Execution time:0
<--------------------------------------------------------------------->
for n :500------------------->
execution time(serial) : 0 seconds
Serial value: 3.168
pi value for parallel(reduction):3.12 Execution time(reduction) :0 seconds
pi value:(atomic) 3.152 Execution time:999928
Pi value:(critical) 3.28 Execution time:1.00017e+06
<--------------------------------------------------------------------->
for n :10000------------------->
execution time(serial) : 0 seconds
Serial value: 3.1444
pi value for parallel(reduction):3.074 Execution time(reduction) :999.928 seconds
pi value:(atomic) 3.1396 Execution time:1.00017e+06
Pi value:(critical) 3.1316 Execution time:999928
<--------------------------------------------------------------------->
```