

Home Work 03: EAS 520

Sayem Khan

Tuesday, Nov 07, 2019

Task 1

Problem a, b

Implementation of Monte Carlo integration algorithm in C, C++ or Fortran.

Solution

The problem is given below:

```
1  /*****
2   * 10D Monte Carlo
3   *****/
4  #include <iostream>
5  #include <cstdlib>
6  #include <ctime>
7  #include <cmath>
8  #include <chrono>
9
10
11 using namespace std;
12
13 double integration_fqn(unsigned long long int NumPoint, int dimation); // main
    calculation
14 double error_per_step(double current, double previous);                //Error
    calculation
15 double fqn(double *x, int y);                                          // Lets
    take a simple fqn: f(x) = x in 10D.                                // To check where is the point
16 double interval_map(double lowerLim, double upperLim);                // mapping
    the interval
17 double volume(double a, double b, int dim);                            //, double
    a2, double b2); // function for 2D vouleme calculation
18
19 int main(int argc, char **argv)
20 {
21     //double intigral = 0.0;
22     srand(time(NULL));
23     //Declearation and Initialization of the variables
24     unsigned long long int N = atoll(argv[1]); // Number of Random point
25     int dimation = 10;
26     //auto t_start = std::chrono::high_resolution_clock::now();
    // In this case 10D intigration
27     integration_fqn(N, dimation);
28     // auto t_end = std::chrono::high_resolution_clock::now();
29     //std::chrono::duration<double, std::milli> duration = (t_end - t_start);
30     // cout << N << " " << piN << " " << error(piN) << endl;
31     return 0;
32 }
```

```

33
34 double integration_fqn(unsigned long long int NumPoint, int dimention)
35 {
36     auto t_start = std::chrono::high_resolution_clock::now();
37     unsigned long long int N_attemps = NumPoint; //Total Number of points
38     //double I = 0.0; //Approximate integration
39     double a = -1.0; //lower bound
40     double b = 1.0; // upper bound
41     //int D = dimention;
42     double *x;
43     x = new double[dimention];
44     double sum = 0.0;
45     double V = volume(a, b, dimention);
46     unsigned long long int check = 4;
47     double previous = 0.0;
48
49     for (unsigned long long int i = 0; i < N_attemps; ++i)
50     {
51         for (int j = 0; j < dimention; ++j)
52         {
53             x[j] = interval_map(a, b);
54         }
55         sum = sum + fqn(x, dimention);
56         if (i == check)
57         {
58             double current = V * sum / double(i);
59             double err_per_step = error_per_step(current, previous);
60             previous = current;
61             check = 4 * check;
62             auto t_end = std::chrono::high_resolution_clock::now();
63             std::chrono::duration<double, std::milli> duration = (t_end - t_start)
;
64             cout << i << " " << current << " " << err_per_step << " " << duration.
count()/1000<<endl;
65             //cout << i << " " << current << " " << err_per_step << endl;
66
67         }
68     }
69
70     delete[] x;
71
72     return 0;
73 }
74
75 double interval_map(double lowerLim, double upperLim)
76 {
77     /*****
78     * Mapping the [0,1]=>[lowerLim, upperLim]
79     * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
80     * Solving this we get, f(x) = (upperLim-lowerLim)x + lowerLim
81     *****/
82     // double upperLim = b;
83     // double lowerLim = a;
84     double randomNumber = ((double)rand()) / ((double)RAND_MAX);
85     return ((upperLim - lowerLim) * randomNumber + lowerLim);
86 }
87

```

```

88 /*****
89  * *****f(x) = x in 10D: *****
90  * f(x1,x2...,x10) = 1 + x1 + x2 + x3 +...+ x10 = 2**10 *
91  *****/
92 double fqn(double *x, int dim)
93 {
94     double value = 0.0;
95     for (int i = 0; i < dim; ++i)
96     {
97         value = value + x[i];
98     }
99     value = 1 + value;
100    return value;
101 }
102
103 double volume(double a, double b, int dim) //, double a2, double b2)
104 {
105     int length = b - a;
106     /*****
107     * V = L**dim
108     *****/
109     return std::pow(length, dim);
110 }
111
112 double error_per_step(double a, double b)
113 {
114     // const double pi = 3.141592653589793;
115     // double absolute_error = fabs(piN - pi);
116     // return absolute_error;
117     return fabs(a - b);
118 }

```

Listing 1: Code for Task 01

```

1 #!/bin/bash
2 g++ -Wall -O3 -ffast-math -mavx -o mc10d_1_b mc10d_1_b.cpp -lm
3 g++ -Wall -O3 -ffast-math -mavx -o mc10d_1_b_time mc10d_1_b_time.cpp -lm
4 #g++ -Wall -o mc10d mc10d.cpp
5
6 file_1="dataSerial.dat"
7 file_2="timeData.dat"
8
9
10 if [ -f $file_1 ] ; then
11     rm $file_1
12 fi
13
14 if [ -f $file_2 ] ; then
15     rm $file_2
16 fi
17
18
19 i=9
20 ./mc10d_1_b $((10**i)) >> dataSerial.dat
21 echo "Using C++ timer: " >> timeData.dat
22 ./mc10d_1_b_time $((10**i)) >> timeData.dat
23 # time ./mc10d_1_b_time $((10**i)) >> timeData.dat
24

```

```

25
26 echo "Done! Check data file."
27 python3 plot.py

```

Listing 2: Driver script for Task 01

For the problem output is taken for every 4^k step. Time using C++ time for the Serial code time for $N = 1000000000$ is 108.965s. Also, same as the Linux timer that is **real 1m48.965s**. These result is documented in **dataSerial.dat** and **timedata.dat** files.

```

1 4 5080.24 5080.24 1.9919e-05
2 16 2471.4 2608.83 6.8764e-05
3 64 1507.09 964.315 8.1285e-05
4 256 1104.77 402.317 0.000108577
5 1024 1039.78 64.9883 0.00020541
6 4096 1028.26 11.5184 0.00057211
7 16384 1000.56 27.7005 0.00199103
8 65536 1021.12 20.5554 0.00740416
9 262144 1023.38 2.25632 0.0289531
10 1048576 1023.51 0.134259 0.115692
11 4194304 1022.96 0.547066 0.457918
12 16777216 1023.98 1.01816 1.94801
13 67108864 1024 0.0215758 7.89578
14 268435456 1024.19 0.185054 31.2206

```

Listing 3: Output sample for program

```

1 Using C++ timer:
2 For Serial code time for N = 1000000000 is 108.965s

```

Listing 4: Time, measured by C++ timer

Task 2

Problem a

Use OpenMP to parallelize the for-loop over the number of N random samples.

Solution

```

1 /*****
2  * 10D Monte Carlo
3  *****/
4 #include <iostream>
5 #include <cstdlib>
6 #include <ctime>
7 #include <cmath>
8 #include <omp.h>
9
10 using namespace std;
11
12 double integration_fqn(unsigned long long int NumPoint, int dimention); // main
    calculation
13 double error_per_step(double current, double previous); //Error
    calculation
14 double fqn(double *x, int y); // Lets
    take a simple fqn: f(x) = x in 10D. // To check where is the point
15 double interval_map(double lowerLim, double upperLim, unsigned int *seed); //
    mapping the interval

```

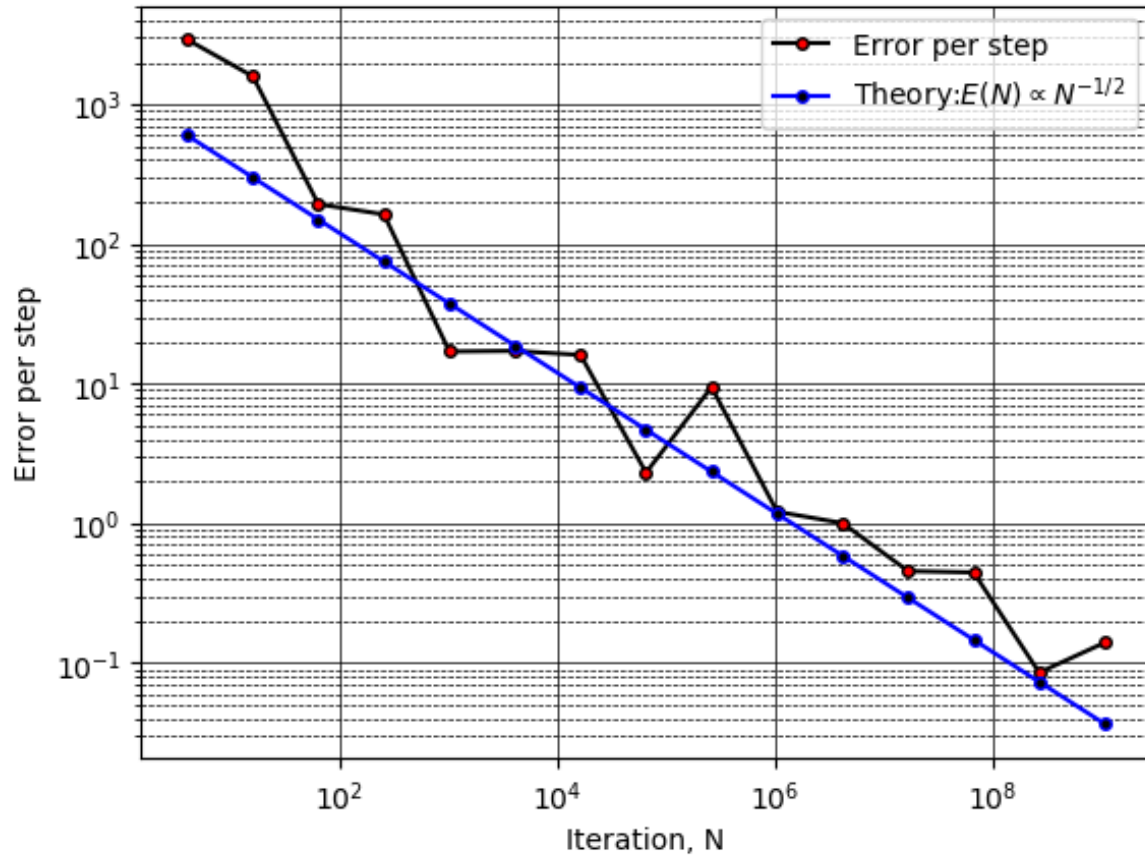


Figure 1: Step by step error (For every 4^k step) and Theoretical Error for 10D MC integration for serial code. Error is decreasing, so we can say code is working properly.

```

16 double volume(double a, double b, int dim); //,
    double a2, double b2); // function for 2D vouluem calculation
17
18 int main(int argc, char **argv)
19 {
20     //double intigral = 0.0;
21     //srand(time(NULL));
22     //Declearation and Initialization of the variables
23     unsigned long long int N = atoll(argv[1]); // Number of Random point
24     const int NThreads = 4; // atoi(argv[2]);
25     int dimation = 10; // In this case 10D intigration
26     omp_set_num_threads(NThreads);
27     integration_fqn(N, dimation);
28     // cout << N << " " << piN << " " << error(piN) << endl;
29     //cout << N << " " << intigral << endl;
30     return 0;
31 }
32
33 double integration_fqn(unsigned long long int NumPoint, int dimation)
34 {
35     unsigned long long int N_attemps = NumPoint; //Total Number of points
36     double I = 0.0; //Approximate integration
37     double a = -1.0; //lower bound
38     double b = 1.0; // upper bound

```

```

39 //int D = dimation;
40 double *x;
41 x = new double[dimation];
42 double sum = 0.0;
43 double sumFinal = 0.0;
44 double V = volume(a, b, dimation);
45 //unsigned long long int check = 4;
46 //double previous = 0.0;
47 unsigned int seed = 0;
48 double timeStart = omp_get_wtime();
49
50 #pragma omp parallel firstprivate(x, sum) private(seed)
51 {
52     const int thread_rank = omp_get_thread_num();
53     seed = time(NULL) * (int)(thread_rank + 1);
54     //cout << "Thread " << thread_rank << " reporting for work<< endl;
55     // printf("Thread %i is reporting for work \n", thread_rank);
56
57 #pragma omp for
58 for (unsigned long long int i = 0; i < N_attemps; ++i)
59 {
60
61     for (int j = 0; j < dimation; ++j)
62     {
63         x[j] = interval_map(a, b, &seed);
64     }
65     sum = sum + fqn(x, dimation);
66 }
67 //#pragma omp critical
68 #pragma omp atomic
69     sumFinal = sumFinal + sum;
70 }
71
72 double timeEnd = omp_get_wtime();
73 double wallTime = timeEnd - timeStart;
74 delete[] x;
75
76 I = V * sumFinal / (double)N_attemps;
77 cout << N_attemps << " " << I << " " << wallTime << " " << omp_get_max_threads
78 () << endl;
79
80 return 0;
81 }
82
83 double interval_map(double lowerLim, double upperLim, unsigned int *seed)
84 {
85     /*****
86      * Mapping the [0,1]=>[lowerLim, upperLim]
87      * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
88      * Solving this we get, f(x) = (upperLim-lowerLim)x + lowerLim
89      *****/
90     // double upperLim = b;
91     // double lowerLim = a;
92     double randomNumber = ((double)rand_r(seed)) / ((double)RAND_MAX);
93     return ((upperLim - lowerLim) * randomNumber + lowerLim);
94 }

```

```

95 /*****
96  * *****f(x) = x in 10D: *****
97  * f(x1,x2...,x10) = 1 + x1 + x2 + x3 +...+ x10 = 2**10 *
98  *****/
99 double fqn(double *x, int dim)
100 {
101     double value = 0.0;
102     for (int i = 0; i < dim; ++i)
103     {
104         value = value + x[i];
105     }
106     value = 1 + value;
107     return value;
108 }
109
110 double volume(double a, double b, int dim) //, double a2, double b2)
111 {
112     int length = b - a;
113     /*****
114     * V = L**dim
115     *****/
116     return std::pow(length, dim);
117 }
118
119 double error_per_step(double a, double b)
120 {
121     // const double pi = 3.141592653589793;
122     // double absolute_error = fabs(piN - pi);
123     // return absolute_error;
124     return fabs(a - b);
125 }

```

Listing 5: Code for Task 02, problem a (Parallel code for 10D integration)

```

1 Thread 0 is working
2 Thread 1 is working
3 Thread 3 is working
4 Thread 2 is working
5 10 608.408 0.0129856 4
6 100 1184.08 0.00257244 4
7 1000 1164.36 0.0162969 4
8 10000 1067.8 0.00229937 4
9 100000 1018.14 0.00479873 4
10 1000000 1021.96 0.0470217 4
11 10000000 1023.87 0.461724 4
12 100000000 1023.88 4.93132 4
13 1000000000 1023.87 44.0962 4

```

Listing 6: Output file for Task 02, problem a (dataParallel_2_a.dat). NB: Output pattern: "N Intergration wallTime MaxNumThread"

Problem b

Parallel code for output every 4^k step.

Solution

```

1  /*****
2   * 10D Monte Carlo
3   *****/
4  #include <iostream>
5  #include <cstdlib>
6  #include <ctime>
7  #include <cmath>
8  #include <omp.h>
9  #include <chrono>
10
11
12 using namespace std;
13
14 double integration_fqn(unsigned long long int NumPoint, int dimention); // main
    calculation
15 double error_per_step(double current, double previous); //Error
    calculation
16 double fqn(double *x, int y); // Lets
    take a simple fqn: f(x) = x in 10D. // To check where is the point
17 double interval_map(double lowerLim, double upperLim, unsigned int *seed); //
    mapping the interval
18 double volume(double a, double b, int dim); //,
    double a2, double b2); // function for 2D voulume calculation
19
20 int main(int argc, char **argv)
21 {
22     //double intigral = 0.0;
23     //srand(time(NULL));
24     //Declearation and Initialization of the variables
25     unsigned long long int N = atoll(argv[1]); // Number of Random point
26     const int NThreads = 4; // atoi(argv[2]);
27     int dimention = 10; // In this case 10D intigration
28     omp_set_num_threads(NThreads);
29     integration_fqn(N, dimention);
30     // cout << N << " " << piN << " " << error(piN) << endl;
31     //cout << N << " " << intigral << endl;
32     return 0;
33 }
34
35 double integration_fqn(unsigned long long int NumPoint, int dimention)
36 {
37     unsigned long long int N_attemps = NumPoint; //Total Number of points
38     double I = 0.0; //Approximate integration
39     double a = -1.0; //lower bound
40     double b = 1.0; // upper bound
41     //int D = dimention;
42     double *x;
43     x = new double[dimention];
44     double sum = 0.0;
45     double sumFinal = 0.0;
46     double V = volume(a, b, dimention);
47     unsigned long long int check = 4;
48     double previous = 0.0;
49     unsigned int seed = 0;
50     double timeStart = omp_get_wtime();
51     // int counter = 0;

```



```

52
53 #pragma omp parallel firstprivate(x, sum, check) private(seed)
54 {
55     const int thread_rank = omp_get_thread_num();
56     seed = time(NULL) * (int)(thread_rank + 1);
57 #pragma omp for
58     for (unsigned long long int i = 0; i < N_attempts; ++i)
59     {
60
61         for (int j = 0; j < dimention; ++j)
62         {
63             x[j] = interval_map(a, b, &seed);
64         }
65         sum = sum + fqn(x, dimention);
66 #pragma omp critical
67         {
68             sumFinal = sumFinal + sum;
69             if (i == check)
70             {
71                 double current = V * sum / double(i);
72                 double err_per_step = error_per_step(current, previous);
73                 previous = current;
74                 check = 4 * check;
75                 int th = omp_get_thread_num();
76                 double timeEnd = omp_get_wtime();
77                 double wallTime = timeEnd - timeStart;
78                 cout << i << " " << current << " " << err_per_step << " " <<
th << endl;
79                 //cout << i << " " << current << " " << err_per_step << " " <<
wallTime << endl;
80             }
81         }
82     }
83 }
84
85 // double timeEnd = omp_get_wtime();
86 // double wallTime = timeEnd - timeStart;
87 delete[] x;
88
89 // I = V * sumFinal / (double)N_attempts;
90 // cout << N_attempts << " " << I << " " << wallTime << " " <<
omp_get_max_threads() << endl;
91 // // //cout << counter << endl;
92 return 0;
93 }
94
95 double interval_map(double lowerLim, double upperLim, unsigned int *seed)
96 {
97     /*****
98      * Mapping the [0,1]=>[lowerLim, upperLim]
99      * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
100      * Solving this we get, f(x) = (upperLim-lowerLim)x + lowerLim
101      *****/
102     // double upperLim = b;
103     // double lowerLim = a;
104     double randomNumber = ((double)rand_r(seed)) / ((double)RAND_MAX);
105     return ((upperLim - lowerLim) * randomNumber + lowerLim);

```

```

106 }
107
108 /*****
109  * *****f(x) = x in 10D: *****
110  * f(x1,x2...,x10) = 1 + x1 + x2 + x3 +...+ x10 = 2**10 *
111  *****/
112 double fqn(double *x, int dim)
113 {
114     double value = 0.0;
115     for (int i = 0; i < dim; ++i)
116     {
117         value = value + x[i];
118     }
119     value = 1 + value;
120     return value;
121 }
122
123 double volume(double a, double b, int dim) //, double a2, double b2)
124 {
125     int length = b - a;
126     /*****
127     * V = L**dim
128     *****/
129     return std::pow(length, dim);
130 }
131
132 double error_per_step(double a, double b)
133 {
134     // const double pi = 3.141592653589793;
135     // double absolute_error = fabs(piN - pi);
136     // return absolute_error;
137     return fabs(a - b);
138 }

```

Listing 7: Code for Task 02, problem b (Parallel code for 10D integration with output for every 4^k step)

```

1 4 1897.73 1897.73 0
2 16 1170.67 727.059 0
3 64 1216.15 45.4809 0
4 256 1218.69 2.53458 0
5 1024 1056.92 161.764 0
6 4096 1002.24 54.6813 0
7 16384 1003.25 1.00902 0
8 65536 1020.97 17.7242 0
9 262144 1024.15 3.17808 0
10 1048576 1024.59 0.434212 0
11 4194304 1023.8 0.786908 0
12 16777216 1023.85 0.0477935 0
13 67108864 1024.01 0.163583 0
14 268435456 1024 0.0113546 0
15 1073741824 1024 0.000716433 0
16 4294967296 1024 0.00371232 0

```

Listing 8: Output file for Task 02, problem b (dataParallel_2_b.dat). Master thread did the final work.

Problem c

Time for both serial and parallel code.

Solution

Driver program is given below:

```

1 #!/bin/bash
2 g++ -Wall -O3 -ffast-math -mavx -o mc10d_1_b_time mc10d_1_b_time.cpp -lm
3 g++ -Wall -fopenmp -O3 -ffast-math -mavx -o mc10d_parallel_optimized_2_c
  mc10d_parallel_2_c.cpp -lm
4 #g++ -Wall -o mc10d mc10d.cpp
5
6 file_1="timeParallel.dat"
7 file_2="timeSerial.dat"
8
9 if [ -f $file_1 ] ; then
10     rm $file_1
11 fi
12
13 if [ -f $file_2 ] ; then
14     rm $file_2
15 fi
16
17 i=1
18 while [[ i -le 11 ]]
19 do
20     ./mc10d_1_b_time $((10*i)) >> timeSerial.dat
21     ./mc10d_parallel_optimized_2_c $((10*i)) >> timeParallel.dat
22     ((i = i + 1))
23 done
24 echo "Done! Check data file."

```

Listing 9: Driver script for Task 02, problem c

Output is given below:

```

1 For Serial code time for N = 10 is 2.905e-06s
2 For Serial code time for N = 100 is 1.2336e-05s
3 For Serial code time for N = 1000 is 0.000173149s
4 For Serial code time for N = 10000 is 0.00110168s
5 For Serial code time for N = 100000 is 0.0116284s
6 For Serial code time for N = 1000000 is 0.110736s
7 For Serial code time for N = 10000000 is 1.05517s
8 For Serial code time for N = 100000000 is 11.4688s
9 For Serial code time for N = 1000000000 is 106.943s
10 For Serial code time for N = 10000000000 is 1035.66s
11 For Serial code time for N = 100000000000 is 10349.9s

```

Listing 10: Times taken by Serial code

```

1 For parallel code: Time for N = 10 is 0.000249944s
2 For parallel code: Time for N = 100 is 0.000269048s
3 For parallel code: Time for N = 1000 is 0.000334549s
4 For parallel code: Time for N = 10000 is 0.00147503s
5 For parallel code: Time for N = 100000 is 0.0120788s
6 For parallel code: Time for N = 1000000 is 0.114125s
7 For parallel code: Time for N = 10000000 is 1.11547s
8 For parallel code: Time for N = 100000000 is 11.3999s
9 For parallel code: Time for N = 1000000000 is 114.531s
10 For parallel code: Time for N = 10000000000 is 1138.43s
11 For parallel code: Time for N = 100000000000 is 11627.8s

```

Listing 11: Times taken by Parallel code

Astonishingly, parallel code takes much time than Serial code. Since this program is run in my Laptop, it could be the effect of OS noise.

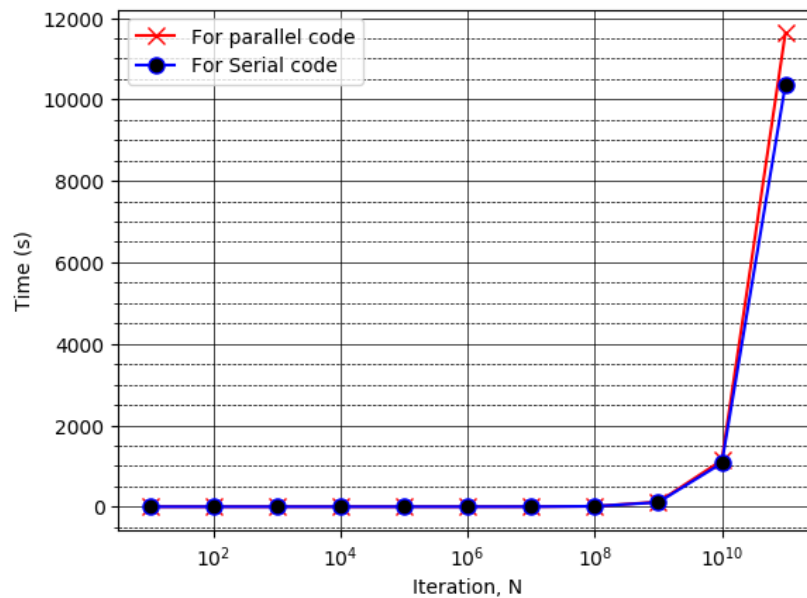


Figure 2: Time comparison between Serial and Parallel code.

Problem d

Comparison between Serial and parallel code.

Solution

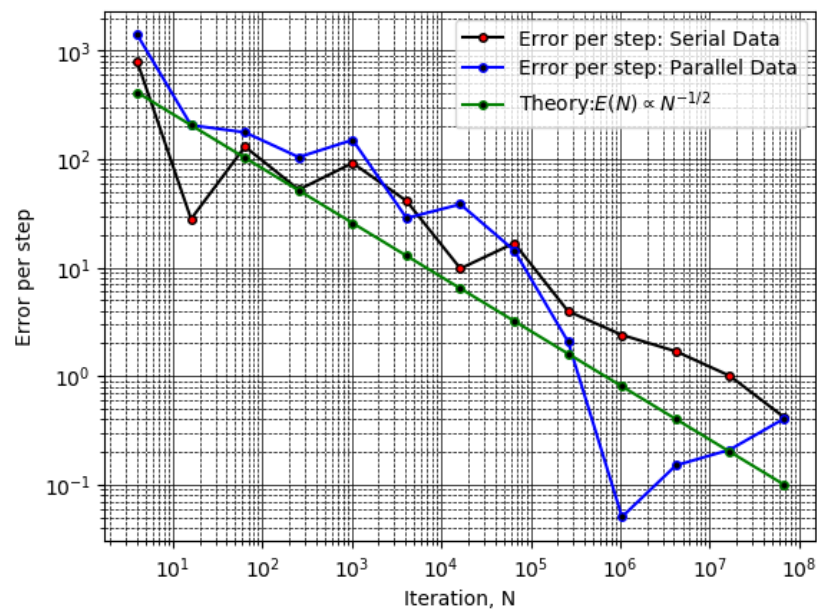


Figure 3: Step by step error (For every 4^k step) and Theoretical Error for 10D MC integration for serial code and Parallel. Error is decreasing, so we can say code is working properly.

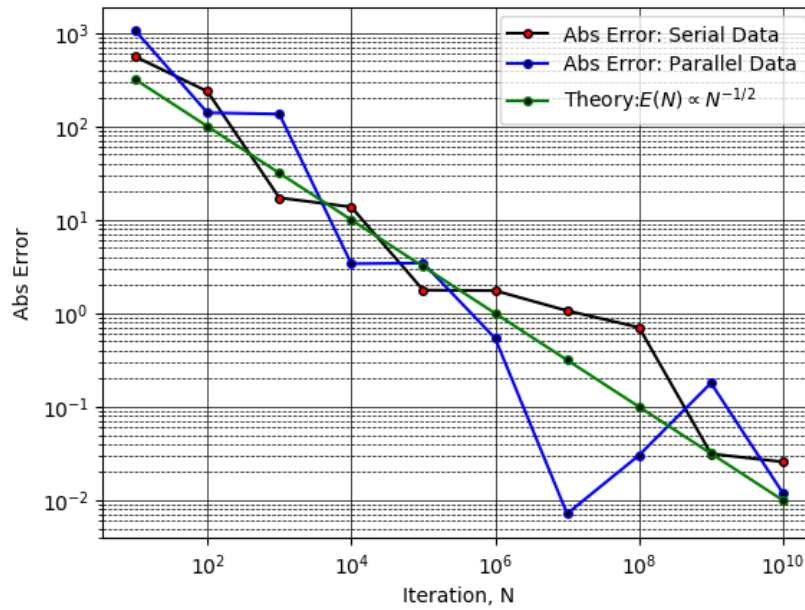


Figure 4: Absolute Error and Theoretical Error for 10D MC integration for serial code and Parallel. Error is decreasing, so we can say code is working properly.

Task 3

Problem a

Modification of OpenMP parallelized code to for Likelihood function.

Solution

Code snippet for Task 3, problem a is given below:

```

1  double integration_fqn(unsigned long long int NumPoint, int dimation)
2  {
3      unsigned long long int N_attempts = NumPoint; //Total Number of points
4      double I = 0.0; //Approximate integration
5      double a = -0.5; //lower bound
6      double b = 0.5; // upper bound
7      //int D = dimation;
8      double *x;
9      x = new double[dimation];
10     double sum = 0.0;
11     double sumFinal = 0.0;
12     double V = volume(a, b, dimation);
13     //unsigned long long int check = 4;
14     //double previous = 0.0;
15     unsigned int seed = 0;
16     double timeStart = omp_get_wtime();
17
18     #pragma omp parallel firstprivate(x, sum) private(seed)
19     {
20         //double sum = 0.0;
21         const int thread_rank = omp_get_thread_num();
22         seed = time(NULL) * (int)(thread_rank+1);
23         //cout << "Thread " << thread_rank << " reporting for work<< endl;
24     }

```

```

25         // printf("Thread %i is reporting for work \n", thread_rank);
26
27 #pragma omp for
28     for (unsigned long long int i = 0; i < N_attempts; ++i)
29     {
30
31         interval_map(a, b, dimention, &seed, x);
32         sum = sum + L(x, dimention);
33     }
34 #pragma omp atomic
35     sumFinal = sumFinal + sum;
36 }
37
38 double timeEnd = omp_get_wtime();
39 double wallTime = timeEnd - timeStart;
40 delete[] x;
41
42 I = V * sumFinal / (double)N_attempts;
43 //I = exp(I);
44 cout << N_attempts << " " << I << " " << wallTime << " " << omp_get_max_threads
45 () << endl;
46
47 return 0;
48 }
49
50 void interval_map(const double a, const double b, const int dim, unsigned int *
51 seed, double *x)
52 {
53     for (int i = 0; i < dim; ++i)
54     {
55         double random = ((double)rand_r(seed)) / ((double)RAND_MAX);
56         random = (b - a) * random + a;
57         x[i] = random;
58     }
59 }

```

Listing 12: Code snippet for Task 3, problem a.

By using the driver program:

```

1 #!/bin/bash
2 g++ -Wall -fopenmp -O3 -ffast-math -mavx -o mc10d_parallel_optimized_3_a
3     mc10d_parallel_3_a.cpp -lm
4
5 file_1="3_a_data.dat"
6
7 if [ -f $file_1 ] ; then
8     rm $file_1
9 fi
10
11 threadNum=4
12 i=1
13 while [[ i -le 10 ]]
14 do
15     ./mc10d_parallel_optimized_3_a $((10*i)) $threadNum >> 3_a_data.dat

```

```

16     ((i = i + 1))
17 done
18 echo "Done! Check data file."

```

Listing 13: Driver program for Task 3, problem a

```

1 10 4.35824e-32 0.0331868 4
2 100 1.17318e-17 0.0401403 4
3 1000 1.39793e-16 0.0318567 4
4 10000 7.75728e-13 0.0487708 4
5 100000 1.72123e-11 0.030293 4
6 1000000 1.07764e-11 0.133178 4
7 10000000 3.2812e-11 0.904849 4
8 100000000 3.19912e-11 8.36386 4
9 1000000000 3.09318e-11 84.5904 4
10 10000000000 3.12723e-11 870.941 4

```

Listing 14: Output of the program Task 3, problem a. Output pattern: N-Integration-wallTime-MaxNumThrea

Problem b

Testing the equation.

Solution

If we put $x_i = 1$ we should get output like $\ln L = 0$, if we put $x_i = 0$ we should get output like $\ln L = -9$.

```

1 int main(int argc, char **argv)
2 {
3
4     int dimation = 10; // In this case 10D intigration
5
6     double y = lnLtest(dimation);
7     cout << "For x_i = 2: lnL = " << y << " L = exp(" << y << ")" << endl;
8
9     return 0;
10 }
11
12 /*Bayesian Likelihood Function*/
13
14 double lnL(double *x, int dim)
15 {
16
17     double fqn = 0.0;
18
19     for (int i = 0; i < dim - 1; ++i) //sum 0 to 9
20     {
21         fqn = fqn + (1.0 - x[i]) * (1.0 - x[i]) + 100. * (x[i + 1] - x[i] * x[i])
22         * (x[i + 1] - x[i] * x[i]);
23     }
24
25     fqn = -fqn;
26
27     return fqn;
28 }
29
30 double lnLtest(int dimation)
31 {

```

```

31     double *x;
32     double sum = 0.0;
33     x = new double[dimension];
34     for (int j = 0; j < dimension; ++j)
35     {
36         x[j] = 2;
37     }
38     sum = sum + lnL(x, dimension);
39     delete[] x;
40
41     return sum;
42 }

```

Listing 15: Code snippet for Task 3, problem a.

Output is given below:

```

1 For x_i = 1: lnL = -0 L = exp(-0)
2 For x_i = 0: lnL = -9 L = exp(-9)
3 For x_i = 2: lnL = -3609 L = exp(-3609)

```

Listing 16: Testing the function.

Problem c

Scaling test.

Solution

Scaling test is done in Stampede2. Every data is taken 100 times to minimize the noise. Script is given below:

```

1 #!/bin/bash
2 ##-----
3 #SBATCH -J 3_c_scale           #Job name
4 #SBATCH -o 3_c_scale.o%j      #Name of stdout
5 #SBATCH -e 3_c_scale.e%j
6 #SBATCH -p development
7 #SBATCH -N 1
8 #SBATCH -n 1
9 #SBATCH -t 00:20:00
10 #SBATCH --mail-user=skhan2@umassd.edu
11 #SBATCH --mail-type=all
12 #
13 g++ -Wall -fopenmp -O3 -ffast-math -mavx -o mc10d_parallel_optimized_3_c_scaling
    mc10d_parallel_3_c_scaling.cpp -lm
14 ## Strong scaling test ##
15
16 file_1="3_c_dataStrong.dat"
17 file_2="3_c_dataWeak.dat"
18
19 if [ -f $file_1 ] ; then
20     rm $file_1
21 fi
22
23
24 if [ -f $file_2 ] ; then
25     rm $file_2
26 fi
27

```

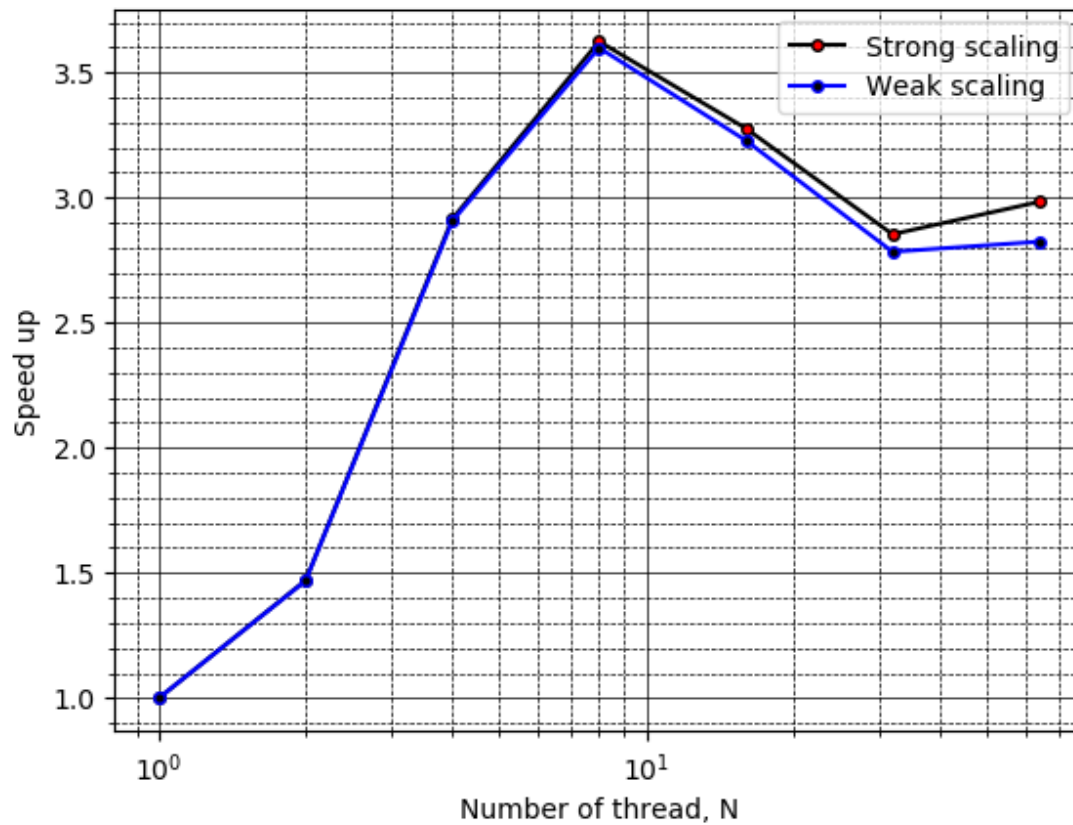



Figure 5: Strong and weak scaling test.

```

28
29 i=1
30 threadNum="1 2 4 8 16 32 64"
31 for th in $threadNum
32 do
33     echo "Number of thread in use: "$th
34     while [[ i -le 100 ]]
35     do
36         echo "Strong Scaling Test: "$i
37         ./mc10d_parallel_optimized_3_c_scaling $((10**6)) $th >> 3_c_dataStrong.dat
38         ((i = i + 1))
39     done
40     i=1
41 done
42
43
44 echo "____Weak Scaling test_____"
45 ## Weak scaling test ##
46 i=1
47 threadNum="1 2 4 8 16 32 64"
48
49 for th in $threadNum
50 do
51     while [[ i -le 100 ]]
52     do
53         echo "Weak Scaling Test: "$i

```

```

54      ((N = 100000 * $th))
55      echo "Number of thread in use: "$th
56      echo "Number of sample point: "$N
57      ./mc10d_parallel_optimized_3_c_scaling $N $th >> 3_c_dataWeak.dat
58      ((i = i + 1))
59  done
60      i=1
61 done
62
63 # wait
64 # python3 plot.py
65 # echo "Done! Check data files: 3_c_dataStrong.dat & 3_c_dataWeak.dat"

```

Listing 17: Testing the function.

Problem d

Output of the integration after each 4^k step.

Solution

The code is given below:

```

1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  #include <cmath>
5  #include <omp.h>
6  #include <chrono>
7
8  using namespace std;
9
10 void interval_map(const double a, const double b, const int dim, unsigned int *
    seed, double *x)
11 {
12
13     for (int i = 0; i < dim; ++i)
14     {
15         double random = ((double)rand_r(seed)) / ((double)RAND_MAX);
16         random = (b - a) * random + a;
17         x[i] = random;
18     }
19 }
20
21 double L(double *x, int dim)
22 {
23
24     double fqn = 0.0;
25
26     for (int i = 0; i < dim - 1; ++i) //0 to 9 (given)
27     {
28         fqn = fqn + (1.0 - x[i]) * (1.0 - x[i]) + 100.0 * (x[i + 1] - x[i] * x[i])
            * (x[i + 1] - x[i] * x[i]);
29     }
30
31     fqn = exp(-fqn);
32
33     return fqn;

```

```

34 }
35
36 double lnL(double *x, int dim)
37 {
38
39     double fqn = 0.0;
40
41     for (int i = 0; i < dim - 1; ++i) //0 to 9 (given)
42     {
43         fqn = fqn + (1.0 - x[i]) * (1.0 - x[i]) + 100.0 * (x[i + 1] - x[i] * x[i])
44         * (x[i + 1] - x[i] * x[i]);
45     }
46
47     fqn = -fqn;
48
49     return fqn;
50 }
51
52 double volume(double a, double b, int dim) //, double a2, double b2)
53 {
54     int length = b - a;
55     /*****
56      * V = L**dim
57      *****/
58     return std::pow(length, dim);
59 }
60
61 void intigration(unsigned long long int N, const int dimention)
62 {
63     const double a = -0.5;
64     const double b = 0.5;
65     const double V = volume(a, b, dimention);
66     unsigned long long int start_loop = 1;
67     unsigned long long int end_loop = 4;
68
69     bool continue_work = true;
70     double sumFinal = 0.0;
71
72     double x[dimention]; // array of random numbers
73     double sum = 0.0;
74
75     #pragma omp parallel firstprivate(x) shared(start_loop, end_loop, sum,
76     continue_work)
77     {
78         //double sum = 0.0;
79         double previous = 0.0;
80         const int thread_rank = omp_get_thread_num();
81         //printf("Thread %i reporting for work.\n", thread_rank);
82         //cout << "Thread " << thread_rank << " reporting for work" << endl;
83         unsigned int seed = time(NULL) * (int)(thread_rank + 1);
84         while (continue_work)
85         {
86             #pragma omp for //reduction(+ \
87             : sum)
88             for (unsigned int i = start_loop; i <= end_loop; ++i)
89             {
90                 interval_map(a, b, dimention, &seed, x);

```

```

89         double fqn_value = L(x, dimation);
90 #pragma omp atomic
91         sum += fqn_value;
92     }
93 // #pragma omp atomic
94 //     sumFinal = sumFinal + sum;
95 #pragma omp barrier
96 #pragma omp master
97     {
98         double current = V * sum / (double)end_loop;
99         double err_per_step = fabs(current - previous);
100        previous = current;
101        cout << end_loop << " " << current << " " << err_per_step << endl;
102        start_loop = end_loop + 1;
103        if (start_loop >= N)
104        {
105            continue_work = false;
106        }
107        else
108        {
109            if (end_loop > (N / 4))
110            {
111                end_loop = N;
112            }
113            else
114            {
115                end_loop = 4 * end_loop;
116            }
117        }
118    }
119 #pragma omp barrier
120 }
121 }
122 //return 0;
123 }
124
125 int main(int argc, char **argv)
126 {
127     const unsigned long long int N = atol(argv[1]);
128     const int NThreads = atoi(argv[2]);
129     const int dimation = 10;
130     omp_set_num_threads(NThreads);
131     double timeStart = omp_get_wtime();
132     intigration(N, dimation);
133     double timeEnd = omp_get_wtime();
134     double wallTime = timeEnd - timeStart;
135     cout << N << " " << wallTime << " " << omp_get_max_threads() << endl;
136     // << "s" << endl;
137     return 0;
138 }

```

Listing 18: Code for Task 3, problem d.

This code run on Stampede2. The script is given below:

```

1 #!/bin/bash
2 ##-----
3 #SBATCH -J 3_d          #Job name

```

```

4 #SBATCH -o 3_d.o%j      #Name of stdout
5 #SBATCH -e 3_d.e%j
6 #SBATCH -p development
7 #SBATCH -N 1
8 #SBATCH -n 1
9 #SBATCH -t 00:20:00
10 #SBATCH --mail-user=skhan2@umassd.edu
11 #SBATCH --mail-type=all
12 #
13
14 ## Strong scaling test ##
15 g++ -Wall -fopenmp -O3 -ffast-math -mavx -o mc10d_parallel_optimized_3_d_1
    mc10d_parallel_3_d_1.cpp -lm
16 file_1="3_d_data.dat"
17 #file_2="3_c_dataWeak.dat"
18
19 if [ -f $file_1 ] ; then
20     rm $file_1
21 fi
22
23 N=68719476736
24 th=64
25
26 ./mc10d_parallel_optimized_3_d_1 $N $th >> 3_d_data.dat
27
28 wait
29
30 echo "Done! Check data file: 3_d_data.dat"

```

Listing 19: Code for Task 3, problem d.

The output data is:

```

1 4 1.66418e-35 1.66418e-35
2 16 7.11683e-26 7.11683e-26
3 64 1.91695e-15 1.91695e-15
4 256 4.95094e-16 1.42185e-15
5 1024 1.79868e-16 3.15226e-16
6 4096 1.55177e-12 1.55159e-12
7 16384 6.71555e-13 8.80218e-13
8 65536 1.3279e-11 1.26074e-11
9 262144 2.43475e-11 1.10685e-11
10 1048576 1.83518e-11 5.99563e-12
11 4194304 2.20593e-11 3.70743e-12
12 16777216 2.3537e-11 1.47769e-12
13 67108864 3.10999e-11 7.56292e-12
14 268435456 3.11452e-11 4.53009e-14
15 1073741824 3.03245e-11 8.2065e-13
16 4294967296 7.58113e-12 2.27434e-11
17 17179869184 1.89528e-12 5.68585e-12
18 68719476736 4.73821e-13 1.42146e-12
19 68719476736 172.35 64

```

Listing 20: Output for Task 3, problem d. Format: N-Integration-ErrorPerStep. Line 19 indicates total N at final, total time taken (measured by omp timer) and total number of threads.

1-digit accuracy is achieved at $N = 268435456$.

The below code was run in my laptop (Intel Core i5-2.3GHz, 2-cores). It took more than 2 hours with $N = 68719476736$ and have not achieved any 1-digit accuracy.

```

1 double integration_fqn(unsigned long long int NumPoint, int dimention)
2 {
3     unsigned long long int N_attemps = NumPoint; //Total Number of points
4     //double I = 0.0; //Approximate integration
5     double a = -0.5; //lower bound
6     double b = 0.5; // upper bound
7     //int D = dimention;
8     double *x;
9     x = new double[dimention];
10    double sum = 0.0;
11    double sumFinal = 0.0;
12    double V = volume(a, b, dimention);
13    unsigned long long int check = 4;
14    double previous = 0.0;
15    unsigned int seed = 0;
16    double timeStart = omp_get_wtime();
17    // int counter = 0;
18
19    #pragma omp parallel firstprivate(x) private(seed)
20    {
21        const int thread_rank = omp_get_thread_num();
22        seed = time(NULL) * (int)(thread_rank + 1);
23        #pragma omp for reduction(+ \
24                                : sum)
25        for (unsigned long long int i = 0; i < N_attemps; ++i)
26        {
27
28            for (int j = 0; j < dimention; ++j)
29            {
30                x[j] = interval_map(a, b, &seed);
31            }
32            // #pragma omp atomic
33            sum += L(x, dimention);
34        #pragma omp atomic
35            sumFinal = sumFinal + sum;
36        // #pragma omp barrier
37        #pragma omp critical
38        {
39            if (i == check)
40            {
41                double current = V * sumFinal / double(i);
42                double err_per_step = error_per_step(current, previous);
43                double absErr = absError(current);
44                previous = current;
45                check = 4 * check;
46                int th = omp_get_thread_num();
47                // double timeEnd = omp_get_wtime();
48                // double wallTime = timeEnd - timeStart;
49                cout << i << " " << current << " " << absErr << endl;
50                // << " " << th << endl;
51                // cout << i << " " << current << " " << err_per_step << " " <<
52                wallTime << endl;
53            }
54        }
55    }

```

```

56
57     delete[] x;
58
59     double timeEnd = omp_get_wtime();
60     double wallTime = timeEnd - timeStart;
61     cout << "Total wallTime: " << wallTime << " "
62          << "s" << endl;
63
64     return 0;
65 }
66
67 double interval_map(double lowerLim, double upperLim, unsigned int *seed)
68 {
69     /*****
70      * Mapping the [0,1]=>[lowerLim, upperLim]
71      * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
72      * Solving this we get, f(x) = (upperLim-lowerLim)x + lowerLim
73      *****/
74     // double upperLim = b;
75     // double lowerLim = a;
76     double randomNumber = ((double)rand_r(seed)) / ((double)RAND_MAX);
77     return ((upperLim - lowerLim) * randomNumber + lowerLim);
78 }

```

Listing 21: Code for Task 3, problem d (An alternative approach).

```

1  4  1.74984e-22  3.1e-11
2  16 4.37461e-23  3.1e-11
3  64 8.18401e-19  3.1e-11
4  256 2.19343e-19 3.1e-11
5  1024 5.67672e-12 2.53233e-11
6  4096 3.63613e-12 2.73639e-11
7  16384 1.10303e-11 1.99697e-11
8  65536 1.07404e-11 2.02596e-11
9  262144 2.92014e-11 1.79863e-12
10 1048576 2.0324e-11 1.0676e-11
11 4194304 4.1365e-11 1.0365e-11
12 16777216 3.33074e-11 2.30745e-12
13 67108864 3.01826e-11 8.1741e-13
14 268435456 3.03172e-11 6.82809e-13
15 1073741824 3.03789e-11 6.21054e-13
16 4294967296 3.07488e-11 2.51177e-13
17 17179869184 3.07958e-11 2.04214e-13
18 Total wallTime: 8288.4 s

```

Listing 22: Output for Task 3, problem d (An alternative approach).

I can not figure out the bug! N.B: I did not upload this code in bitbucket.