# 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:

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

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

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

Listing 1: Code for Task 01

```
#!/bin/bash
_{2}|g++ -Wall -03 -ffast-math -mavx -o mc10d_1_b mc10d_1_b.cpp -lm
 g++ -Wall -03 -ffast-math -mavx -o mc10d_1_b_time mc10d_1_b_time.cpp -lm
 #g++ -Wall -o mc10d mc10d.cpp
6 file 1="dataSerial.dat"
 file_2="timeData.dat"
  if [ -f $file_1 ] ; then
     rm $file_1
11
12 fi
13
14 if [ -f $file_2 ]; then
      rm $file_2
15
16 fi
17
18
19 i=9
20 ./mc10d_1_b $((10**i)) >> dataSerial.dat
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
```

```
echo "Done! Check data file."
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 = 10000000000 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.

```
4 5080.24 5080.24 1.9919e-05

16 2471.4 2608.83 6.8764e-05

64 1507.09 964.315 8.1285e-05

256 1104.77 402.317 0.000108577

1024 1039.78 64.9883 0.00020541

4096 1028.26 11.5184 0.00057211

16384 1000.56 27.7005 0.00199103

65536 1021.12 20.5554 0.00740416

262144 1023.38 2.25632 0.0289531

1048576 1023.51 0.134259 0.115692

4194304 1022.96 0.547066 0.457918

16777216 1023.98 1.01816 1.94801

67108864 1024 0.0215758 7.89578

268435456 1024.19 0.185054 31.2206
```

Listing 3: Output sample for program

```
Using C++ timer: 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

```
/*************
  * 10D Monte Carlo
  ***************
 #include <iostream>
 #include <cstdlib>
 #include <ctime>
 #include <cmath>
 #include <omp.h>
 using namespace std;
11
 double integration fqn(unsigned long long int NumPoint, int dimention);
      calculation
 double error_per_step(double current, double previous);
                                                                             //Error
      calculation
double fqn(double *x, int y);
                                                                             // Lets
      take a simple fqn: f(x) = x in 10D.
                                                    // To check where is the point
double interval_map(double lowerLim, double upperLim, unsigned int *seed); //
     mapping the interval
```

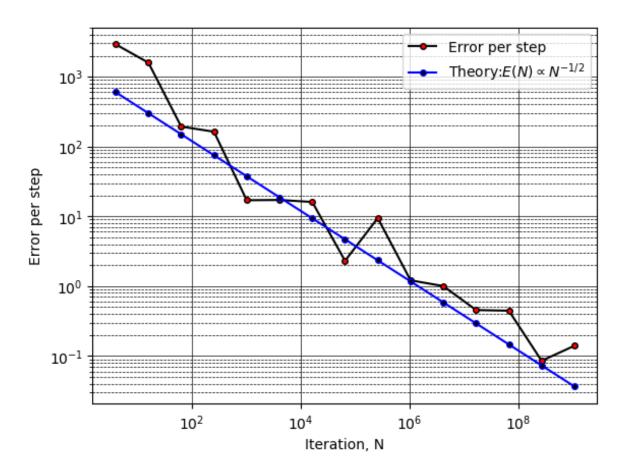


Figure 1: Step by step error (For every  $4^k$  step) and Theoretical Error for 10D MC integration for serial code. Error is decreasing, so we can say code is working properly.

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

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

```
f(x_1, x_2, \dots, x_{10}) = 1 + x_1 + x_2 + x_3 + \dots + x_{10} = 2**10 *
97
  double fqn(double *x, int dim)
     double value = 0.0;
     for (int i = 0; i < dim; ++i)
102
         value = value + x[i];
     }
105
     value = 1 + value;
     return value;
108
  double volume(double a, double b, int dim) //, double a2, double b2)
111
     int length = b - a;
112
     /********
      * V = L**dim
114
     ***********/
     return std::pow(length, dim);
117
118
  double error_per_step(double a, double b)
119
120
  {
     // const double pi = 3.141592653589793;
     // double absolute_error = fabs(piN - pi);
     // return absolute_error;
     return fabs(a - b);
124
 }
125
```

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

```
Thread 0 is working
Thread 1 is working
Thread 3 is working
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 1000000000 1023.88 4.93132 4
13 1000000000 1023.87 44.0962 4
```

Listing 6: Output file for Task 02, problem a (data Parallel\_2\_a.dat). NB: Output pattern: "N Intergration wall Time MaxNumThread"

### Problem b

Parallel code for output every  $4^k$  step.

#### Solution

```
/***********
  * 10D Monte Carlo
  ***************
  #include <iostream>
5 #include <cstdlib>
6 #include <ctime>
 | #include < cmath >
 #include <omp.h>
9 #include <chrono>
12 using namespace std;
13
double integration_fqn(unsigned long long int NumPoint, int dimention);
                                                                                // main
      calculation
double error_per_step(double current, double previous);
                                                                                //Error
      calculation
                                                                                // Lets
double fqn(double *x, int y);
      take a simple fqn: f(x) = x in 10D.
                                                     // To check where is the point
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
int main(int argc, char **argv)
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
      const int NThreads = 4;
                                                   // atoi(argv[2]);
26
      int dimention = 10;
                                                  // In this case 10D intigration
27
      omp_set_num_threads(NThreads);
28
      integration fqn(N, dimention);
29
      // cout << N << " " << piN << " " << error(piN) << endl;
30
      //cout << N << " " << intigral << endl;
      return 0;
32
  }
33
34
double integration fqn(unsigned long long int NumPoint, int dimention)
  {
36
      unsigned long long int N_attemps = NumPoint; //Total Number of points
37
                                                     //Approximate integration
      double I = 0.0;
38
      double a = -1.0;
                                                     //lower bound
39
      double b = 1.0;
                                                     // upper bound
40
      //int D = dimention;
41
      double *x;
42
      x = new double[dimention];
43
      double sum = 0.0;
44
      double sumFinal = 0.0;
45
      double V = volume(a, b, dimention);
46
      unsigned long long int check = 4;
47
      double previous = 0.0;
48
      unsigned int seed = 0;
49
      double timeStart = omp_get_wtime();
50
      // int counter = 0;
```

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

```
106
 }
  108
  * f(x1, x2..., x10) = 1 + x1 + x2 + x3 + ... + x10 = 2**10 *
  double fqn(double *x, int dim)
112
  {
113
     double value = 0.0;
114
     for (int i = 0; i < dim; ++i)
116
        value = value + x[i];
     }
     value = 1 + value;
     return value;
  }
  double volume(double a, double b, int dim) //, double a2, double b2)
123
124
     int length = b - a;
125
     /********
      * V = L**dim
127
     ***********/
128
     return std::pow(length, dim);
  }
130
  double error_per_step(double a, double b)
133
     // const double pi = 3.141592653589793;
     // double absolute_error = fabs(piN - pi);
     // return absolute error;
136
     return fabs(a - b);
138
```

Listing 7: Code for Task 02, problem b (Prallel 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
  64 1216.15 45.4809 0
  256 1218.69 2.53458 0
  1024 1056.92 161.764 0
  4096 1002.24 54.6813 0
  16384 1003.25 1.00902 0
  65536 1020.97 17.7242 0
  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
  67108864 1024.01 0.163583 0
13
  268435456 1024 0.0113546 0
15 1073741824 1024 0.000716433 0
  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:

```
#!/bin/bash
 g++ -Wall -03 -ffast-math -mavx -o mc10d_1_b_time mc10d_1_b_time.cpp -lm
 g++ -Wall -fopenmp -03 -ffast-math -mavx -o mc10d_parallel_optimized_2_c
     mc10d parallel 2 c.cpp -lm
  #g++ -Wall -o mc10d mc10d.cpp
  file_1="timeParallel.dat"
  file 2="timeSerial.dat"
  if [ -f $file_1 ] ; then
      rm $file_1
11 fi
12
 if [ -f $file_2 ] ; then
      rm $file_2
14
15 fi
16
_{17}|i=1
 while [[ i -le 11 ]]
18
19 do
      ./mc10d_1_b_time $((10**i)) >> timeSerial.dat
20
      ./mc10d_parallel_optimized_2_c $((10**i)) >> timeParallel.dat
      ((i = i + 1))
22
23 done
  echo "Done! Check data file."
```

Listing 9: Driver script for Task 02, problem c

# Output is given below:

```
For Serial code time for N = 10 is 2.905e-06s
For Serial code time for N = 100 is 1.2336e-05s
For Serial code time for N = 1000 is 0.000173149s
For Serial code time for N = 10000 is 0.00110168s
For Serial code time for N = 100000 is 0.0116284s
For Serial code time for N = 1000000 is 0.110736s
For Serial code time for N = 100000000 is 1.05517s
For Serial code time for N = 100000000 is 11.4688s
For Serial code time for N = 1000000000 is 106.943s
For Serial code time for N = 10000000000 is 1035.66s
For Serial code time for N = 100000000000 is 10349.9s
```

Listing 10: Times taken by Serial code

```
For parallel code: Time for N = 10 is 0.000249944s

For parallel code: Time for N = 100 is 0.000269048s

For parallel code: Time for N = 1000 is 0.000334549s

For parallel code: Time for N = 10000 is 0.00147503s

For parallel code: Time for N = 100000 is 0.0120788s

For parallel code: Time for N = 1000000 is 0.114125s

For parallel code: Time for N = 100000000 is 1.11547s

For parallel code: Time for N = 1000000000 is 11.3999s

For parallel code: Time for N = 10000000000 is 114.531s

For parallel code: Time for N = 100000000000 is 1138.43s

For parallel code: Time for N = 1000000000000 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.

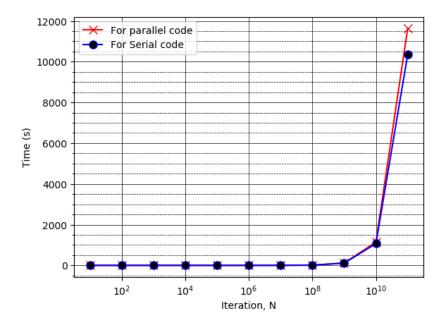


Figure 2: Time comparison between Serial and Parallel code.

### Problem d

Comparison between Serial and parallel code.

# Solution

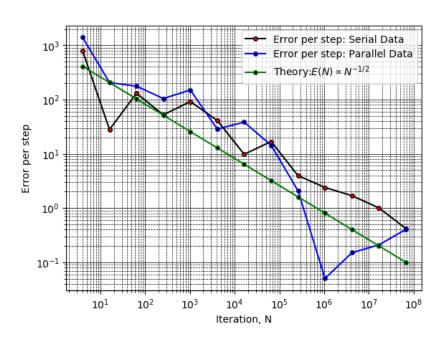


Figure 3: Step by step error (For every  $4^k$  step) and Theoretical Error for 10D MC integration for serial code and Parallel. Error is decreasing, so we can say code is working properly.

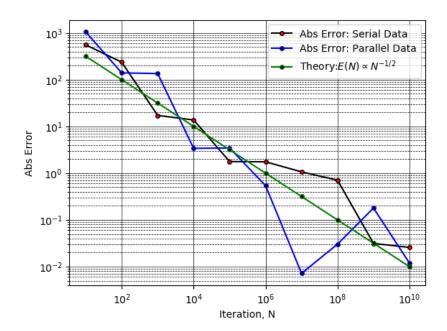


Figure 4: Absolute Error and Theoretical Error for 10D MC integration for serial code and Parallel. Error is decreasing, so we can say code is working properly.

# Task 3

### Problem a

Modification of OpenMP parallelized code to for Likelihood function.

# Solution

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

```
double integration_fqn(unsigned long long int NumPoint, int dimention)
  {
      unsigned long long int N_attemps = NumPoint; //Total Number of points
      double I = 0.0;
                                                      //Approximate integration
      double a = -0.5;
                                                      //lower bound
      double b = 0.5;
                                                      // upper bound
      //int D = dimention;
      double *x;
      x = new double[dimention];
      double sum = 0.0;
11
      double sumFinal = 0.0;
12
      double V = volume(a, b, dimention);
      //unsigned long long int check = 4;
14
      //double previous = 0.0;
15
      unsigned int seed = 0;
16
      double timeStart = omp_get_wtime();
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;</pre>
```

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

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

# By using the driver program:

```
((i = i + 1))
done
echo "Done! Check data file."
```

Listing 13: Driver program for Task 3, problem a

```
1 10 4.35824e-32 0.0331868 4

100 1.17318e-17 0.0401403 4

3 1000 1.39793e-16 0.0318567 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 3.12723e-11 870.941 4
```

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

# Problem b

Testing the equation.

#### Solution

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

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

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

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

Output is given below:

```
For x_i = 1: lnL = -0 L = exp(-0)
For x_i = 0: lnL = -9 L = exp(-9)
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:

```
#!/bin/bash
  ##-----
  #SBATCH -J 3_c_scale
                             #Job name
  #SBATCH -o 3_c_scale.o%j
                             #Name of stdout
 #SBATCH -e 3_c_scale.e%j
  #SBATCH -p development
 #SBATCH -N 1
 #SBATCH -n 1
9 #SBATCH -t 00:20:00
#SBATCH --mail-user=skhan2@umassd.edu
#SBATCH --mail-type=all
12 #
13 g++ -Wall -fopenmp -03 -ffast-math -mavx -o mc10d_parallel_optimized_3_c_scaling
     mc10d_parallel_3_c_scaling.cpp -lm
## Strong scaling test ##
15
file_1="3_c_dataStrong.dat"
file_2="3_c_dataWeak.dat"
18
19 if [ -f $file_1 ]; then
      rm $file_1
20
21 fi
22
23
24 if [ -f $file_2 ]; then
      rm $file_2
25
26 fi
27
```

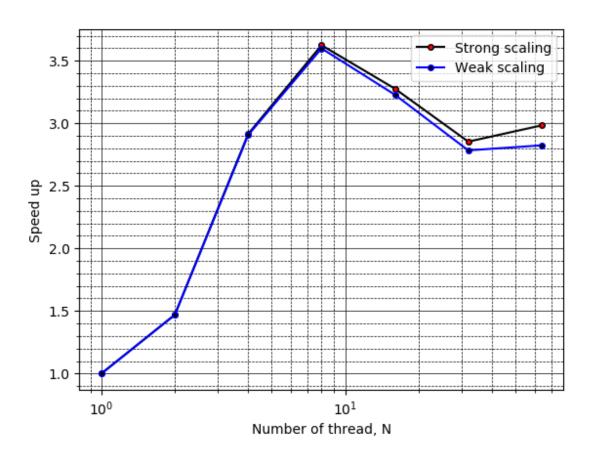


Figure 5: Strong and weak scaling test.

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

```
((N = 100000 * $th))
54
           echo "Number of thread in use: "$th
55
           echo "Number of sample point: "$N
56
           ./mc10d_parallel_optimized_3_c_scaling $N $th >> 3_c_dataWeak.dat
57
           ((i = i + 1))
58
      done
59
      i = 1
60
  done
61
62
  # wait
63
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:

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

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

```
double fqn_value = L(x, dimention);
89
90
  #pragma omp atomic
                    sum += fqn_value;
91
                }
92
  // #pragma omp atomic
93
                   sumFinal = sumFinal + sum;
  //
94
95 #pragma omp barrier
  #pragma omp master
96
97
                    double current = V * sum / (double)end_loop;
98
                    double err_per_step = fabs(current - previous);
99
                    previous = current;
                    cout << end loop << " " << current << " " << err per step << endl;
                    start loop = end loop + 1;
                    if (start_loop >= N)
                    {
104
                         continue_work = false;
                    }
106
                    else
                    {
108
                         if (end_loop > (N / 4))
                         {
                             end_loop = N;
111
                        }
112
                         else
                         {
114
                             end_loop = 4 * end_loop;
                        }
116
                    }
118
  #pragma omp barrier
119
           }
       //return 0;
122
  }
123
  int main(int argc, char **argv)
125
126
  {
       const unsigned long long int N = atol(argv[1]);
       const int NThreads = atoi(argv[2]);
128
       const int dimention = 10;
       omp set num threads (NThreads);
130
       double timeStart = omp_get_wtime();
131
       intigration(N, dimention);
       double timeEnd = omp_get_wtime();
133
       double wallTime = timeEnd - timeStart;
134
       cout << N << " " << wallTime << " " << omp_get_max_threads() << endl;</pre>
135
       // << "s" << endl;
136
       return 0;
137
138
```

Listing 18: Code for Task 3, problem d.

This code run on Stampede 2. The script is given below:

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

```
4 #SBATCH -0 3_d.0%j
                        #Name of stdout
  #SBATCH -e 3_d.e%j
  #SBATCH -p development
  #SBATCH -N 1
 #SBATCH -n 1
  #SBATCH -t 00:20:00
| #SBATCH --mail-user=skhan2@umassd.edu
11 #SBATCH --mail-type=all
12 #
13
## Strong scaling test ##
 g++ -Wall -fopenmp -03 -ffast-math -mavx -o mc10d_parallel_optimized_3_d_1
     mc10d_parallel_3_d_1.cpp -lm
  file_1="3_d_data.dat"
  #file_2="3_c_dataWeak.dat"
18
  if [ -f $file_1 ] ; then
19
      rm $file_1
20
  fi
21
23 N = 68719476736
_{24} th=64
25
  ./mc10d_parallel_optimized_3_d_1 $N $th >> 3_d_data.dat
27
28
  wait
29
  echo "Done! Check data file: 3_d_data.dat"
```

Listing 19: Code for Task 3, problem d.

# The output data is:

```
4 1.66418e-35 1.66418e-35
 16 7.11683e-26 7.11683e-26
 64 1.91695e-15 1.91695e-15
4 256 4.95094e-16 1.42185e-15
 1024 1.79868e-16 3.15226e-16
 4096 1.55177e-12 1.55159e-12
 16384 6.71555e-13 8.80218e-13
 65536 1.3279e-11 1.26074e-11
 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
 17179869184 1.89528e-12 5.68585e-12
 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.

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

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

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

```
4 1.74984e-22 3.1e-11
 16 4.37461e-23 3.1e-11
 64 8.18401e-19 3.1e-11
 256 2.19343e-19 3.1e-11
 1024 5.67672e-12 2.53233e-11
 4096 3.63613e-12 2.73639e-11
 16384 1.10303e-11 1.99697e-11
 65536 1.07404e-11 2.02596e-11
 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
 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.