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;

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)
{
    //double intigral = 0.0;
    srand(time(NULL));
    //Declearation and Initialization of the variables
    unsigned long long int N = atoll(argv[1]); // Number of Random point
    int dimention = 10;
    //auto t_start = std::chrono::high_resolution_clock::now();  // In this case 10D intigration
    integration_fqn(N, dimention);
    // auto t_end = std::chrono::high_resolution_clock::now();
    //std::chrono::duration<double, std::milli> duration = (t_end - t_start);
    // cout << N << " " << piN << " " << error(piN) << endl;
    return 0;
}
```

```

double integration_fqn(unsigned long long int NumPoint, int dimention)
{
    auto t_start = std::chrono::high_resolution_clock::now();
    unsigned long long int N_attemps = NumPoint; //Total Number of points
    //double I = 0.0;                //Approximate integration
    double a = -1.0; //lower bound
    double b = 1.0;  // upper bound
    //int D = dimention;
    double *x;
    x = new double[dimention];
    double sum = 0.0;
    double V = volume(a, b, dimention);
    unsigned long long int check = 4;
    double previous = 0.0;

    for (unsigned long long int i = 0; i < N_attemps; ++i)
    {
        for (int j = 0; j < dimention; ++j)
        {
            x[j] = interval_map(a, b);
        }
        sum = sum + fqn(x, dimention);
        if (i == check)
        {
            double current = V * sum / double(i);
            double err_per_step = error_per_step(current, previous);
            previous = current;
            check = 4 * check;
            auto t_end = std::chrono::high_resolution_clock::now();
            std::chrono::duration<double, std::milli> duration = (t_end - t_start);
            cout << i << " " << current << " " << err_per_step << " " << duration.count()/1000<<endl;
            //cout << i << " " << current << " " << err_per_step << endl;

        }
    }

    delete[] x;

    return 0;
}

double interval_map(double lowerLim, double upperLim)
{
    /***********************************************************
     * Mapping the [0,1]=>[lowerLim, upperLim]
     * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
     * Solving this we get, f(x) = (upperLim-lowerLim)x + lowerLim
    ***********************************************************/
    // double upperLim = b;
    // double lowerLim = a;
    double randomNumber = ((double)rand()) / ((double)RAND_MAX);
    return ((upperLim - lowerLim) * randomNumber + lowerLim);
}

```

```
/*********************************************************
 * ****************f(x) = x in 10D: **********************
 * f(x1,x2....,x10) = 1 + x1 + x2 + x3 +...+ x10 = 2**10 *
**********************************************************/
double fqn(double *x, int dim)
{
    double value = 0.0;
    for (int i = 0; i < dim; ++i)
    {
        value = value + x[i];
    }
    value = 1 + value;
    return value;
}

double volume(double a, double b, int dim) //, double a2, double b2)
{
    int length = b - a;
    /*************
     * V = L**dim
    *************/
    return std::pow(length, dim);
}

double error_per_step(double a, double b)
{
    // const double pi = 3.141592653589793;
    // double absolute_error = fabs(piN - pi);
    // return absolute_error;
    return fabs(a - b);
}
```

Listing 1: Code for Task 01

```
#!/bin/bash
g++ -Wall -O3 -ffast-math -mavx -o mc10d_1_b mc10d_1_b.cpp -lm
g++ -Wall -O3 -ffast-math -mavx -o mc10d_1_b_time mc10d_1_b_time.cpp -lm
#g++ -Wall -o mc10d mc10d.cpp

file_1="dataSerial.dat"
file_2="timeData.dat"


if [ -f $file_1 ] ; then
    rm $file_1
fi

if [ -f $file_2 ] ; then
    rm $file_2
fi


i=9
./mc10d_1_b $((10**i)) >> dataSerial.dat
echo "Using C++ timer: " >> timeData.dat
./mc10d_1_b_time $((10**i)) >> timeData.dat
# time ./mc10d_1_b_time $((10**i)) >> timeData.dat

```

```

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 $N = 1000000000$ 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.

```
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;

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, unsigned int *seed); // mapping the interval
```

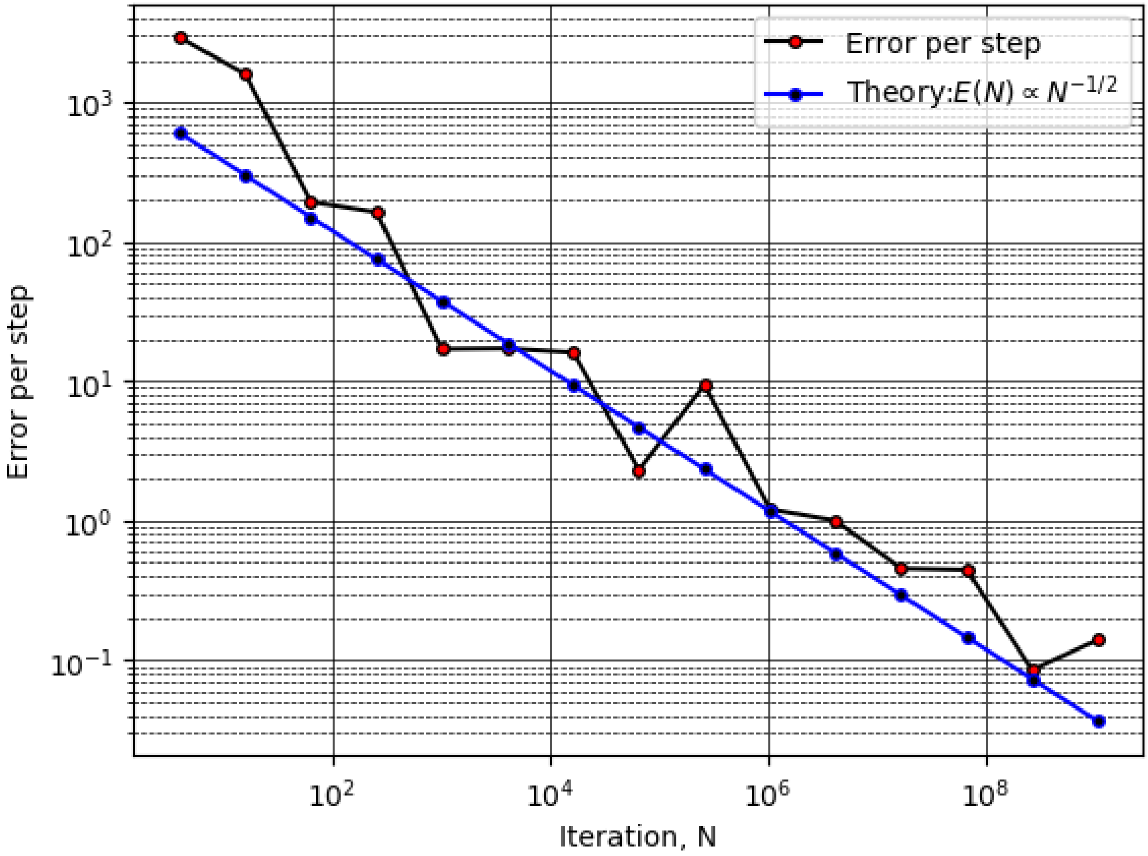

Figure 1: Step by step error (For every 4^k step) and Theoretical Error for 10D MC integration for serial code. Error is decreasing, so we can say code is working properly.

```
double volume(double a, double b, int dim);                              //, double a2, double b2); // function for 2D voulume calculation

int main(int argc, char **argv)
{
    //double intigral = 0.0;
    //srand(time(NULL));
    //Declearation and Initialization of the variables
    unsigned long long int N = atoll(argv[1]); // Number of Random point
    const int NThreads = 4;                    // atoi(argv[2]);
    int dimention = 10;                        // In this case 10D intigration
    omp_set_num_threads(NThreads);
    integration_fqn(N, dimention);
    // cout << N << " " << piN << " " << error(piN) << endl;
    //cout << N << " " << intigral << endl;
    return 0;
}

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 = -1.0;                             //lower bound
    double b = 1.0;                              // upper bound
    //int D = dimention;
    double *x;
    x = new double[dimention];
    double sum = 0.0;
    double sumFinal = 0.0;
    double V = volume(a, b, dimention);
    //unsigned long long int check = 4;
    //double previous = 0.0;
    unsigned int seed = 0;
    double timeStart = omp_get_wtime();

#pragma omp parallel firstprivate(x, sum) private(seed)
    {
        const int thread_rank = omp_get_thread_num();
        seed = time(NULL) * (int)(thread_rank + 1);
        //cout << "Thread " << thread_rank << " reporting for work<< endl;
        // printf("Thread %i is reporting for work \n", thread_rank);

#pragma omp for
        for (unsigned long long int i = 0; i < N_attemps; ++i)
        {

            for (int j = 0; j < dimention; ++j)
            {
                x[j] = interval_map(a, b, &seed);
            }
            sum = sum + fqn(x, dimention);
        }
//#pragma omp critical
#pragma omp atomic
        sumFinal = sumFinal + sum;
    }

    double timeEnd = omp_get_wtime();
    double wallTime = timeEnd - timeStart;
    delete[] x;

    I = V * sumFinal / (double)N_attemps;
    cout << N_attemps << " " << I << " " << wallTime << " " << omp_get_max_threads() << endl;

    return 0;
}

double interval_map(double lowerLim, double upperLim, unsigned int *seed)
{
    /**************************************************************
     * Mapping the [0,1]=>[lowerLim, upperLim]
     * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
     * 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);
    return ((upperLim - lowerLim) * randomNumber + lowerLim);
}

```

```
/*********************************************************
 * ****************f(x) = x in 10D: **********************
 * f(x1,x2....,x10) = 1 + x1 + x2 + x3 +...+ x10 = 2**10 *
**********************************************************/
double fqn(double *x, int dim)
{
    double value = 0.0;
    for (int i = 0; i < dim; ++i)
    {
        value = value + x[i];
    }
    value = 1 + value;
    return value;
}

double volume(double a, double b, int dim) //, double a2, double b2)
{
    int length = b - a;
    /*************
     * V = L**dim
    *************/
    return std::pow(length, dim);
}

double error_per_step(double a, double b)
{
    // const double pi = 3.141592653589793;
    // double absolute_error = fabs(piN - pi);
    // return absolute_error;
    return fabs(a - b);
}
```

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
10 608.408 0.0129856 4
100 1184.08 0.00257244 4
1000 1164.36 0.0162969 4
10000 1067.8 0.00229937 4
100000 1018.14 0.00479873 4
1000000 1021.96 0.0470217 4
10000000 1023.87 0.461724 4
100000000 1023.88 4.93132 4
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

```cpp
/*******************
 * 10D Monte Carlo
*******************/
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <omp.h>
#include <chrono>


using namespace std;

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, unsigned int *seed); // 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)
{
    //double intigral = 0.0;
    //srand(time(NULL));
    //Declearation and Initialization of the variables
    unsigned long long int N = atoll(argv[1]); // Number of Random point
    const int NThreads = 4;                    // atoi(argv[2]);
    int dimention = 10;                        // In this case 10D intigration
    omp_set_num_threads(NThreads);
    integration_fqn(N, dimention);
    // cout << N << " " << piN << " " << error(piN) << endl;
    //cout << N << " " << intigral << endl;
    return 0;
}

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 = -1.0;                             //lower bound
    double b = 1.0;                              // upper bound
    //int D = dimention;
    double *x;
    x = new double[dimention];
    double sum = 0.0;
    double sumFinal = 0.0;
    double V = volume(a, b, dimention);
    unsigned long long int check = 4;
    double previous = 0.0;
    unsigned int seed = 0;
    double timeStart = omp_get_wtime();
    //  int counter = 0;
```

```

#pragma omp parallel firstprivate(x, sum, check) private(seed)
    {
        const int thread_rank = omp_get_thread_num();
        seed = time(NULL) * (int)(thread_rank + 1);
#pragma omp for
        for (unsigned long long int i = 0; i < N_attemps; ++i)
        {

            for (int j = 0; j < dimention; ++j)
            {
                x[j] = interval_map(a, b, &seed);
            }
            sum = sum + fqn(x, dimention);
#pragma omp critical
            {
                sumFinal = sumFinal + sum;
                if (i == check)
                {
                    double current = V * sum / double(i);
                    double err_per_step = error_per_step(current, previous);
                    previous = current;
                    check = 4 * check;
                    int th = omp_get_thread_num();
                    double timeEnd = omp_get_wtime();
                    double wallTime = timeEnd - timeStart;
                    cout << i << " " << current << " " << err_per_step << " " << th << endl;
                    //cout << i << " " << current << " " << err_per_step << " " << wallTime << endl;
                }
            }
        }
    }

    // double timeEnd = omp_get_wtime();
    // double wallTime = timeEnd - timeStart;
    delete[] x;

    // I = V * sumFinal / (double)N_attemps;
    // cout << N_attemps << " " << I << " " << wallTime << " " << omp_get_max_threads() << endl;
    // // //cout << counter << endl;
    return 0;
}

double interval_map(double lowerLim, double upperLim, unsigned int *seed)
{
    /**************************************************************
     * Mapping the [0,1]=>[lowerLim, upperLim]
     * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
     * 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);
    return ((upperLim - lowerLim) * randomNumber + lowerLim);
```

```
}

/*********************************************************
 * ****************f(x) = x in 10D: **********************
 * f(x1,x2....,x10) = 1 + x1 + x2 + x3 +...+ x10 = 2**10 *
**********************************************************/
double fqn(double *x, int dim)
{
    double value = 0.0;
    for (int i = 0; i < dim; ++i)
    {
        value = value + x[i];
    }
    value = 1 + value;
    return value;
}

double volume(double a, double b, int dim) //, double a2, double b2)
{
    int length = b - a;
    /*************
     * V = L**dim
    **************/
    return std::pow(length, dim);
}

double error_per_step(double a, double b)
{
    // const double pi = 3.141592653589793;
    // double absolute_error = fabs(piN - pi);
    // return absolute_error;
    return fabs(a - b);
}
```

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

```
4 1897.73 1897.73 0
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
1048576 1024.59 0.434212 0
4194304 1023.8 0.786908 0
16777216 1023.85 0.0477935 0
67108864 1024.01 0.163583 0
268435456 1024 0.0113546 0
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:

```bash
#!/bin/bash
g++ -Wall -O3 -ffast-math -mavx -o mc10d_1_b_time mc10d_1_b_time.cpp -lm
g++ -Wall -fopenmp -O3 -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
fi

if [ -f $file_2 ] ; then
    rm $file_2
fi

i=1
while [[ i -le 11 ]]
do
    ./mc10d_1_b_time $((10**i)) >> timeSerial.dat
    ./mc10d_parallel_optimized_2_c $((10**i)) >> timeParallel.dat
    ((i = i + 1))
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 = 10000000 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 = 10000000 is 1.11547s
For parallel code: Time for N = 100000000 is 11.3999s
For parallel code: Time for N = 1000000000 is 114.531s
For parallel code: Time for N = 10000000000 is 1138.43s
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.

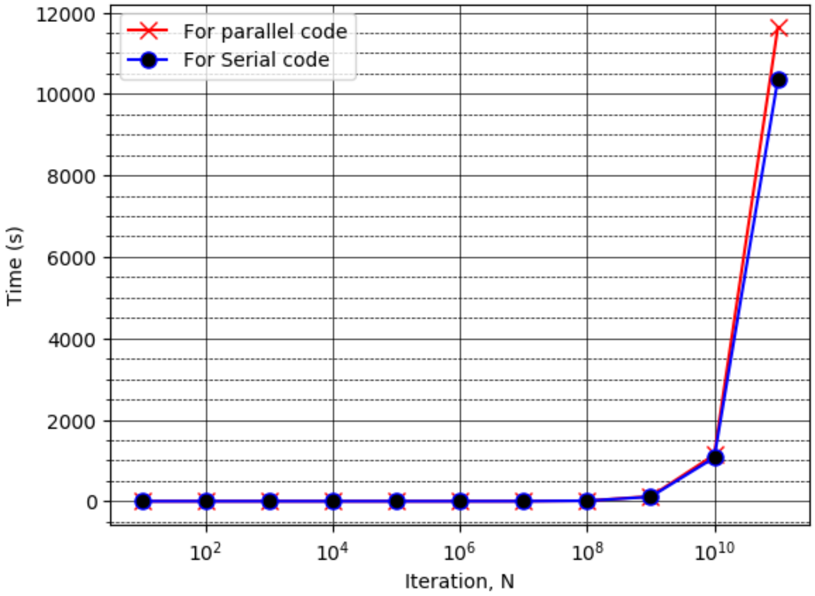

Figure 2: Time comparison between Serial and Parallel code.

Problem d

Comparison between Serial and parallel code.

Solution

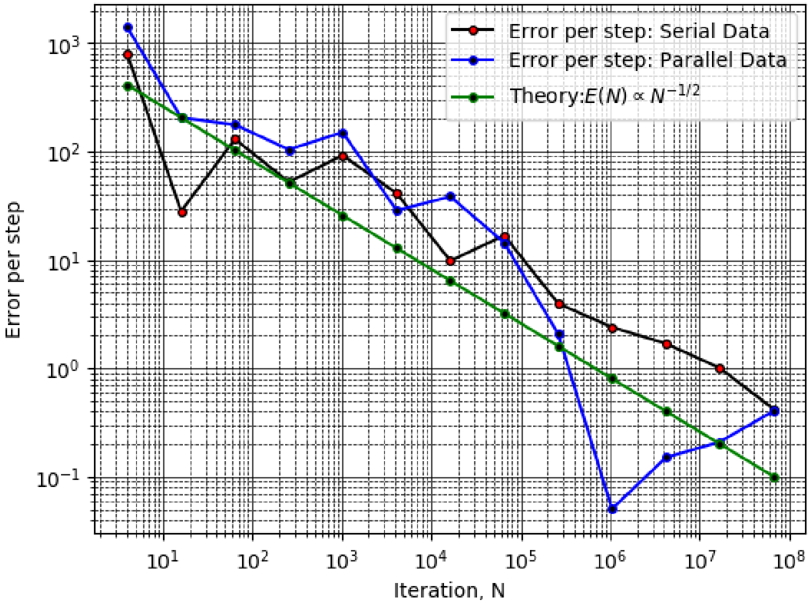

Figure 3: Step by step error (For every 4^k step) and Theoretical Error for 10D MC integration for serial code and Parallel. Error is decreasing, so we can say code is working properly.

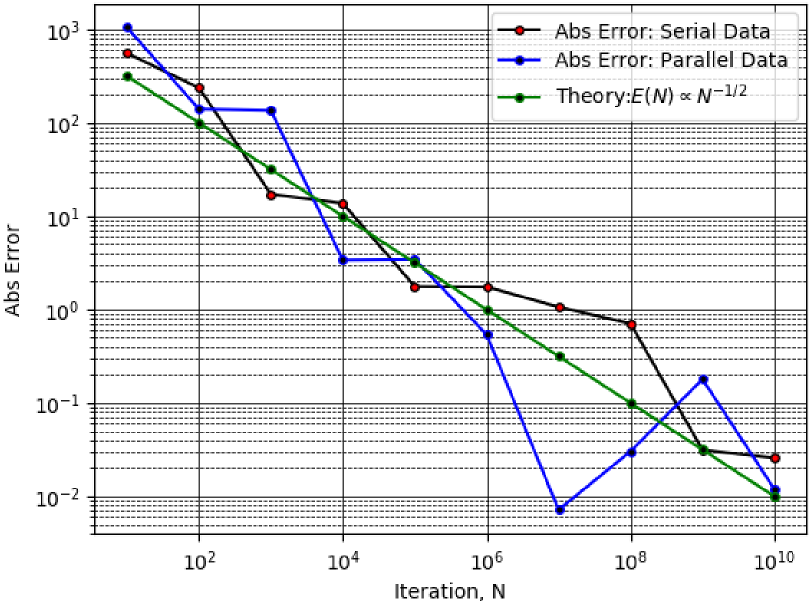

Figure 4: Absolute Error and Theoretical Error for 10D MC integration for serial code and Parallel. Error is decreasing, so we can say code is working properly.

Task 3

Problem a

Modification of OpenMP parallelized code to for Likelihood function.

Solution

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

```

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;
    double sumFinal = 0.0;
    double V = volume(a, b, dimention);
    //unsigned long long int check = 4;
    //double previous = 0.0;
    unsigned int seed = 0;
    double timeStart = omp_get_wtime();

#pragma omp parallel firstprivate(x, sum) private(seed)
    {
        //double sum = 0.0;
        const int thread_rank = omp_get_thread_num();
        seed = time(NULL) * (int)(thread_rank+1);
        //cout << "Thread " << thread_rank << " reporting for work<< endl;
        // printf("Thread %i is reporting for work \n", thread_rank);

#pragma omp for
        for (unsigned long long int i = 0; i < N_attemps; ++i)
        {

            interval_map(a, b, dimention, &seed, x);
            sum = sum + L(x, dimention);
        }
#pragma omp atomic
        sumFinal = sumFinal + sum;
    }

    double timeEnd = omp_get_wtime();
    double wallTime = timeEnd - timeStart;
    delete[] x;

    I = V * sumFinal / (double)N_attemps;
    //I = exp(I);
    cout << N_attemps << " " << I << " " << wallTime << " " << omp_get_max_threads() << endl;

    return 0;
}

void interval_map(const double a, const double b, const int dim, unsigned int *seed, double *x)
{

    for (int i = 0; i < dim; ++i)
    {
        double random = ((double)rand_r(seed)) / ((double)RAND_MAX);
        random = (b - a) * random + a;
        x[i] = random;
    }

}
```

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

By using the driver program:

```
#!/bin/bash
g++ -Wall -fopenmp -O3 -ffast-math -mavx -o mc10d_parallel_optimized_3_a mc10d_parallel_3_a.cpp -lm


file_1="3_a_data.dat"

if [ -f $file_1 ] ; then
    rm $file_1
fi

threadNum=4
i=1
while [[ i -le 10 ]]
do
    ./mc10d_parallel_optimized_3_a $((10**i)) $threadNum >> 3_a_data.dat
    ((i = i + 1))
done
echo "Done! Check data file."
```

Listing 13: Driver program for Task 3, problem a

```
10 4.35824e-32 0.0331868 4
100 1.17318e-17 0.0401403 4
1000 1.39793e-16 0.0318567 4
10000 7.75728e-13 0.0487708 4
100000 1.72123e-11 0.030293 4
1000000 1.07764e-11 0.133178 4
10000000 3.2812e-11 0.904849 4
100000000 3.19912e-11 8.36386 4
1000000000 3.09318e-11 84.5904 4
10000000000 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)
{

    int dimention = 10; // In this case 10D intigration

    double y = lnLtest(dimention);
    cout << "For x_i = 2: lnL = " << y << " L = exp(" << y << ")" << endl;

    return 0;
}

/*Bayesian Likelihood Function*/

double lnL(double *x, int dim)
{

    double fqn = 0.0;

    for (int i = 0; i < dim - 1; ++i) //sum 0 to 9
    {
        fqn = fqn + (1.0 - x[i]) * (1.0 - x[i]) + 100. * (x[i + 1] - x[i] * x[i])
   * (x[i + 1] - x[i] * x[i]);
    }

    fqn = -fqn;

    return fqn;
}

double lnLtest(int dimention)
{
    double *x;
    double sum = 0.0;
    x = new double[dimention];
    for (int j = 0; j < dimention; ++j)
    {
        x[j] = 2;
    }
    sum = sum + lnL(x, dimention);
    delete[] x;

    return sum;
}
```

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
#SBATCH -t 00:20:00
#SBATCH --mail-user=skhan2@umassd.edu
#SBATCH --mail-type=all
#
g++ -Wall -fopenmp -O3 -ffast-math -mavx -o mc10d_parallel_optimized_3_c_scaling mc10d_parallel_3_c_scaling.cpp -lm
## Strong scaling test ##

file_1="3_c_dataStrong.dat"
file_2="3_c_dataWeak.dat"

if [ -f $file_1 ] ; then
    rm $file_1
fi


if [ -f $file_2 ] ; then
    rm $file_2
fi

```

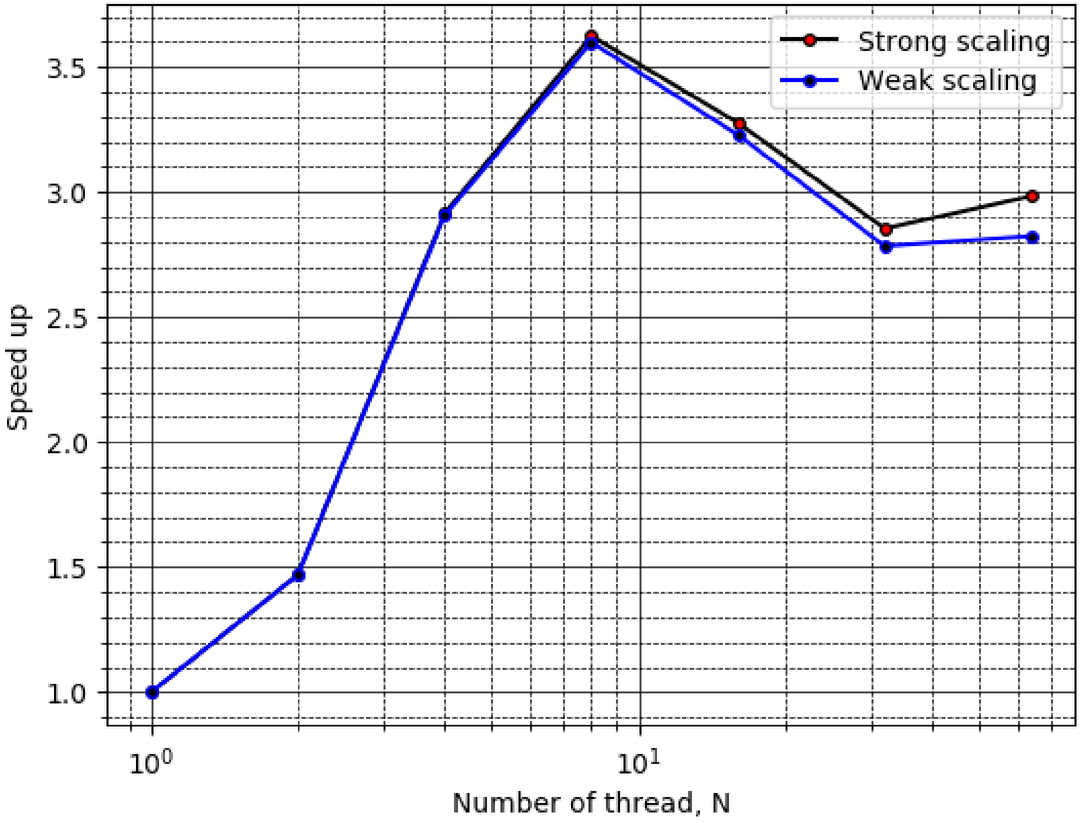

Figure 5: Strong and weak scaling test.

```

i=1
threadNum="1 2 4 8 16 32 64"
for th in $threadNum
do
    echo "Number of thread in use: "$th
    while [[ i -le 100 ]]
    do
    echo "Strong Scaling Test: "$i
    ./mc10d_parallel_optimized_3_c_scaling $((10**6)) $th >> 3_c_dataStrong.dat
    ((i = i + 1))
    done
    i=1
done


echo "______Weak Scaling test_______"
## Weak scaling test ##
i=1
threadNum="1 2 4 8 16 32 64"

    for th in $threadNum
    do
    while [[ i -le 100 ]]
    do
        echo "Weak Scaling Test: "$i
        ((N = 100000 * $th))
        echo "Number of thread in use: "$th
        echo "Number of sample point: "$N
        ./mc10d_parallel_optimized_3_c_scaling $N $th >> 3_c_dataWeak.dat
        ((i = i + 1))
    done
    i=1
done

# wait
# python3 plot.py
# 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>
#include <ctime>
#include <cmath>
#include <omp.h>
#include <chrono>

using namespace std;

void interval_map(const double a, const double b, const int dim, unsigned int *seed, double *x)
{

    for (int i = 0; i < dim; ++i)
    {
        double random = ((double)rand_r(seed)) / ((double)RAND_MAX);
        random = (b - a) * random + a;
        x[i] = random;
    }
}

double L(double *x, int dim)
{

    double fqn = 0.0;

    for (int i = 0; i < dim - 1; ++i) //0 to 9 (given)
    {
        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]);
    }

    fqn = exp(-fqn);

    return fqn;
```

```
}

double lnL(double *x, int dim)
{

    double fqn = 0.0;

    for (int i = 0; i < dim - 1; ++i) //0 to 9 (given)
    {
        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]);
    }

    fqn = -fqn;

    return fqn;
}

double volume(double a, double b, int dim) //, double a2, double b2)
{
    int length = b - a;
    /*************
     * V = L**dim
    **************/
    return std::pow(length, dim);
}

void intigration(unsigned long long int N, const int dimention)
{
    const double a = -0.5;
    const double b = 0.5;
    const double V = volume(a, b, dimention);
    unsigned long long int start_loop = 1;
    unsigned long long int end_loop = 4;

    bool continue_work = true;
    double sumFinal = 0.0;

    double x[dimention]; // array of random numbers
    double sum = 0.0;

#pragma omp parallel firstprivate(x) shared(start_loop, end_loop, sum,
   continue_work)
    {
        //double sum = 0.0;
        double previous = 0.0;
        const int thread_rank = omp_get_thread_num();
        //printf("Thread %i reporting for work.\n", thread_rank);
        //cout << "Thread " << thread_rank << " reporting for work" << endl;
        unsigned int seed = time(NULL) * (int)(thread_rank + 1);
        while (continue_work)
        {
#pragma omp for //reduction(+ \
                          : sum)
            for (unsigned int i = start_loop; i <= end_loop; ++i)
            {
                interval_map(a, b, dimention, &seed, x);
                double fqn_value = L(x, dimention);
#pragma omp atomic
                sum += fqn_value;
            }
// #pragma omp atomic
//             sumFinal = sumFinal + sum;
#pragma omp barrier
#pragma omp master
            {
                double current = V * sum / (double)end_loop;
                double err_per_step = fabs(current - previous);
                previous = current;
                cout << end_loop << " " << current << " " << err_per_step << endl;
                start_loop = end_loop + 1;
                if (start_loop >= N)
                {
                    continue_work = false;
                }
                else
                {
                    if (end_loop > (N / 4))
                    {
                        end_loop = N;
                    }
                    else
                    {
                        end_loop = 4 * end_loop;
                    }
                }
            }
#pragma omp barrier
        }
    }
    //return 0;
}

int main(int argc, char **argv)
{
    const unsigned long long int N = atol(argv[1]);
    const int NThreads = atoi(argv[2]);
    const int dimention = 10;
    omp_set_num_threads(NThreads);
    double timeStart = omp_get_wtime();
    intigration(N, dimention);
    double timeEnd = omp_get_wtime();
    double wallTime = timeEnd - timeStart;
    cout << N << " " << wallTime << " " << omp_get_max_threads() << endl;
    // << "s" << endl;
    return 0;
}
```

Listing 18: Code for Task 3, problem d.

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

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

```
#SBATCH -o 3_d.o%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
#SBATCH --mail-type=all
#

## Strong scaling test ##
g++ -Wall -fopenmp -O3 -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"

if [ -f $file_1 ] ; then
    rm $file_1
fi

N=68719476736
th=64

./mc10d_parallel_optimized_3_d_1 $N $th >> 3_d_data.dat

wait

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.

```
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;
    double sumFinal = 0.0;
    double V = volume(a, b, dimention);
    unsigned long long int check = 4;
    double previous = 0.0;
    unsigned int seed = 0;
    double timeStart = omp_get_wtime();
    //  int counter = 0;

#pragma omp parallel firstprivate(x) private(seed)
    {
        const int thread_rank = omp_get_thread_num();
        seed = time(NULL) * (int)(thread_rank + 1);
#pragma omp for reduction(+ \
                          : sum)
        for (unsigned long long int i = 0; i < N_attemps; ++i)
        {

            for (int j = 0; j < dimention; ++j)
            {
                x[j] = interval_map(a, b, &seed);
            }
            //#pragma omp atomic
            sum += L(x, dimention);
#pragma omp atomic
            sumFinal = sumFinal + sum;
//#pragma omp barrier
#pragma omp critical
            {
                if (i == check)
                {
                    double current = V * sumFinal / double(i);
                    double err_per_step = error_per_step(current, previous);
                    double absErr = absError(current);
                    previous = current;
                    check = 4 * check;
                    int th = omp_get_thread_num();
                    // double timeEnd = omp_get_wtime();
                    // double wallTime = timeEnd - timeStart;
                    cout << i << " " << current << " " << absErr << endl;
                    // << " " << th << endl;
                    //cout << i << " " << current << " " << err_per_step << " " << wallTime << endl;
                }
            }
        }
    }

    delete[] x;

    double timeEnd = omp_get_wtime();
    double wallTime = timeEnd - timeStart;
    cout << "Total wallTime: " << wallTime << " "
         << "s" << endl;

    return 0;
}

double interval_map(double lowerLim, double upperLim, unsigned int *seed)
{
    /**************************************************************
     * Mapping the [0,1]=>[lowerLim, upperLim]
     * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
     * 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);
    return ((upperLim - lowerLim) * randomNumber + lowerLim);
}
```

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
1048576 2.0324e-11 1.0676e-11
4194304 4.1365e-11 1.0365e-11
16777216 3.33074e-11 2.30745e-12
67108864 3.01826e-11 8.1741e-13
268435456 3.03172e-11 6.82809e-13
1073741824 3.03789e-11 6.21054e-13
4294967296 3.07488e-11 2.51177e-13
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.