AMS 530 HW1

Bihua Yu

1 Project Description

Write a parallel program using Monte Carlo method for integration to compute the volume of the following ellipsoid

$$(\frac{x}{3})^2 + (\frac{y}{4})^2 + (\frac{z}{5})^2 = 1$$

You perform minimum necessary calculations to achieve four digits of accuracy. You may use $\frac{3}{4}\pi abc$ to obtain the "exact" result for comparison where $\pi = 3.141592653589793238462$. Now, please measure the time you need to obtain the results using the following cases of processor counts: P=1, 10, 20, 30, 40. Please explain your results.

2 Algorithm and pseudo-code

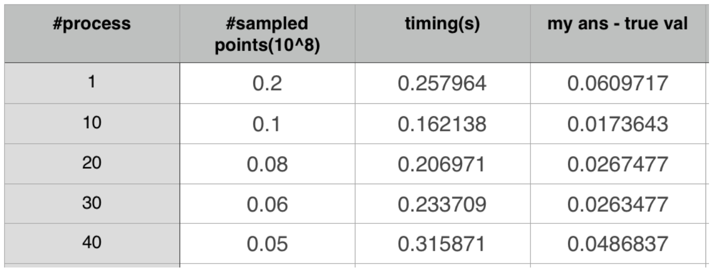
```
int main(&argc,&argv) { // declare variables
// start parallel code
    MPI_Init(&argc,&argv);
    ...
    if (rank == 0) {
    // input N which is the number of samples per process
    }
// broadcast N
    MPI_Bcast()
// generate 3*N random numbers in each process to assign to the coordinates. All points
generated are in the first quadrant
    double *xv = new double[N];
    double *yv = new double[N];
    double *zv = new double[N];
    double left = 0.0;
    double right = 1.0;
    RngUniform(N, xv, left, right);
    RngUniform(N, yv, left, right);
    RngUniform(N, zv, left, right);
// test and count how many points locate inside the 1/8 ellipsoid on each process
    for (int i = 0; i < N; ++ i) {
        ...
        r = $x^2 + y^2 + z^2$;
        if (r < 1) {
        ++ count;
        }
    }
// collect the total number of points are inside the ellipsoid using reduce function
    MPI_Reduce()
    ...
// using the host node to compute and report the volume
    if (myid == 0) {
        double a = 3.0;
        double b = 4.0;
        double c = 5.0;
        double my_vol = 1.0×sum/(num_procs×N)×a×b×c×8;
        double tr_vol = 4.0/3.0×$\pi$×a×b×c;
        ...
    }
//Finish parallel code
    MPI_Finalize()
    ...
}
```

2.1 Source code with compile- and run-time options

Source code is attached with the email.
Compile:
```
module load torque/6.0.2
module load shared
module load intel-stack
mpiicpc -std=c++11 -o dis distribute.cc -mkl
```
Run:
```
mpirun -np 10 ./dis 5000000
```
Or using *.pdb which is attached to the email, too.

3 Results

The results are shown in Figure 1 and Figure 2. In the table, for each row, the results are from one execution. It is a just a representation for many other executions in this



#process	#sampled points(10^8)	timing(s)	my ans - true val
1	0.2	0.257964	0.0609717
10	0.1	0.162138	0.0173643
20	0.08	0.206971	0.0267477
30	0.06	0.233709	0.0263477
40	0.05	0.315871	0.0486837

Figure 1: Performance comparison table

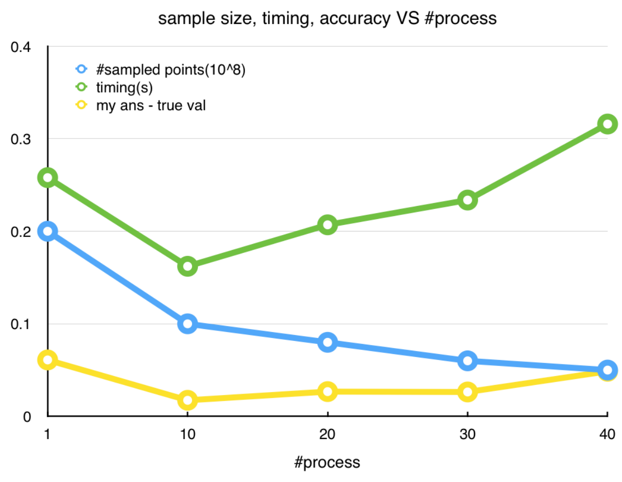


Figure 2: Performance comparison visulization

condition. The typical data is selected and represented in the table. The representative execution is selected when most executions with the same parameter will reach similar results.

4 Analysis

From Figure 1 and Figure 2, we can see that in order to get the same accurate result, using parallel code can not significantly reduce the total number of sampled points in this case. Therefore, as number of process become bigger, the total number of sample points can be bigger, which results in more computing time. If the number of process is too much, parallel computing dose not save much time.