并行计算HW3

15.3

float data[1024],buff[10];
for(int i = 0; i < 10; i++) buff[i] = data[32*i];
MPI_Send(buff, 10, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);

###15.13
（1）
串行代码：

```
#include <iostream>
#include <ctime>
#include <cmath>
using namespace std;
double buffon(int l, int a, int b, int n) {
    clock_t start, end;
    start = clock();
    int hit = 0;
    for (int i = 0; i < n; i++) {
        // rand base
        double base_x = a * (double)rand() / (double)(RAND_MAX);
        double base_y = b * (double)rand() / (double)(RAND_MAX);
        // rand pin
        double cita = (double)rand();
        double pin_x = base_x + l * cos(cita);
        double pin_y = base_y + l * sin(cita);
        if (pin_x <= 0 || pin_x >= a || pin_y <= 0 || pin_y >= b) {
            hit++;
        }
    }
    end = clock();
    cout << "Simulation time is: " << end - start << "ms" << endl;
    return (double)hit / (double)n;
}
int main() {
    int l, a, b, n;
    cin >> n >> a >> b >> l;
    double pos = buffon(l, a, b, n);
    double pi = (double)(2 * l * (a + b) - l * l) / (pos * a * b);
    cout << "The result of PI is : " << pi << endl;
    return 0;
}
```

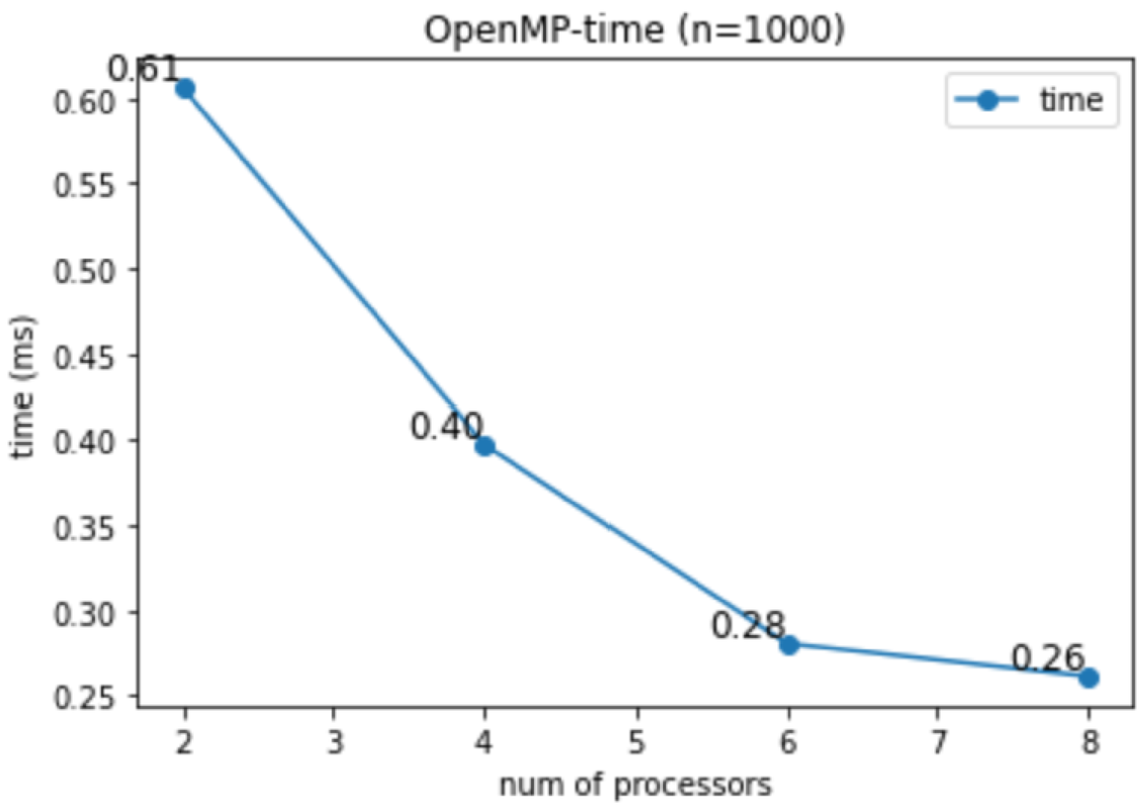
运行结果为：

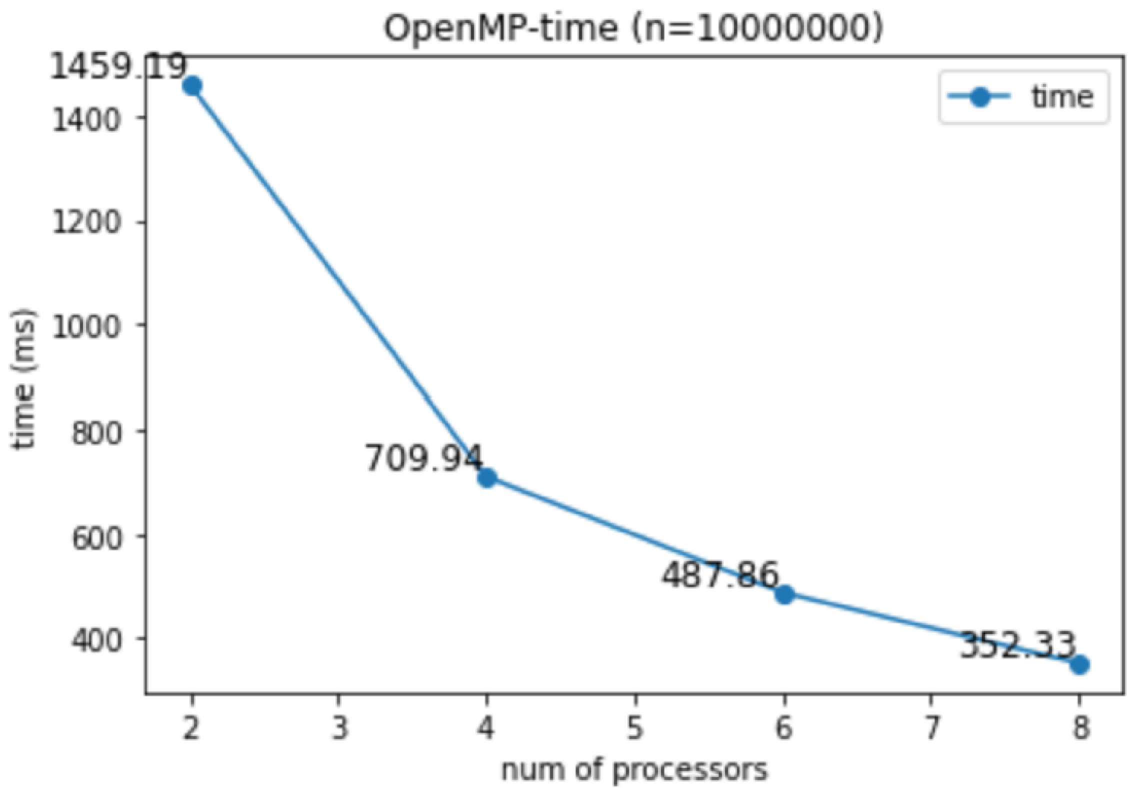
```
1000000
15
16
8
Simulation time is: 121ms
The result of PI is : 3.14155
```

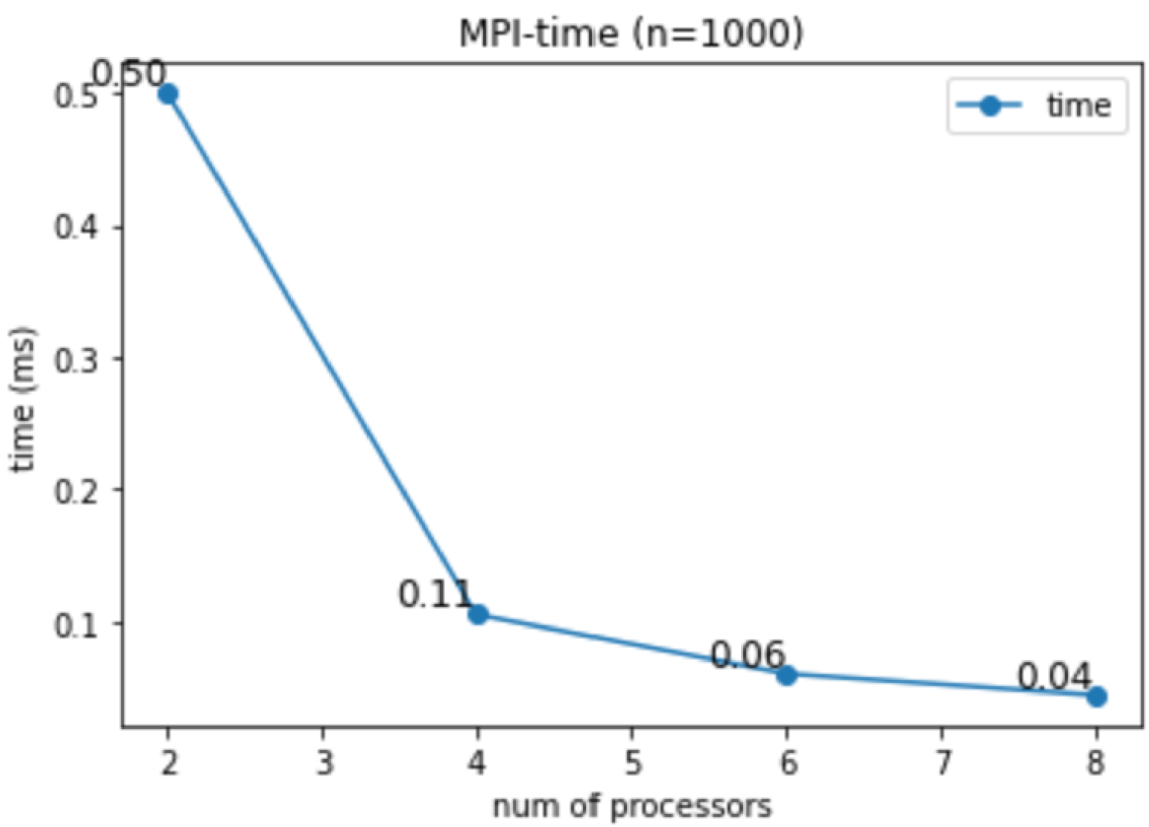
可以看出，当模拟1000000次、针长为8，a为15，b为16时，模拟时间为121ms，精度为小数点后5位（其中3位与实际的pi相同）

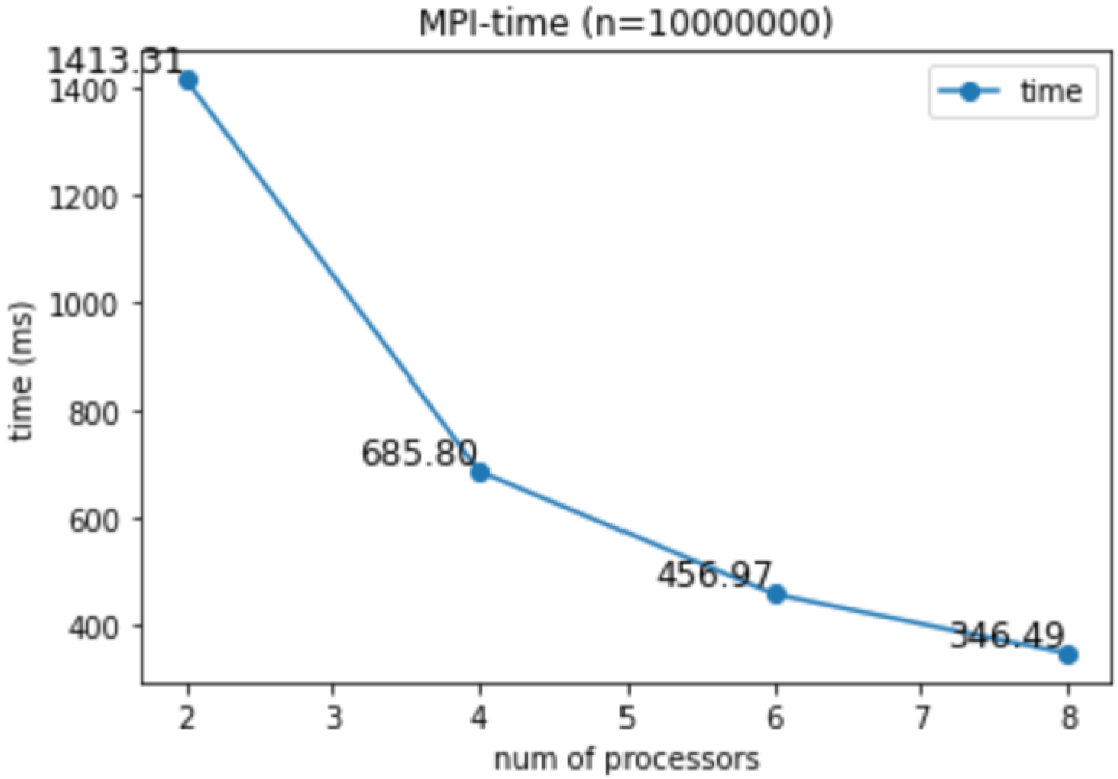
（2）程序是中等规模的

以下为openMP和MPI上运行次数和处理器数量的关系：




OpenMP-time (n=10000000)
time
1459.19
709.94
487.86
352.33
1400
1200
1000
800
600
400
time (ms)
2
3
4
5
6
7
8
num of processors


MPI-time (n=1000)
time
0.50
0.11
0.06
0.04
0.5
0.4
0.3
0.2
0.1
time (ms)
2
3
4
5
6
7
8
num of processors



可以看出，不管是什么规模的数据，运行时间都随着处理器的增加而减少。其中MPI的程序整体运行速度更快，运行时间随处理器增加而减小的幅度也更大。

OpenMP更适用于共享内存系统，更容易使用，并且在负载平衡方面具有一定优势。而MPI更适用于分布式内存系统，虽然编程复杂度较高，但可以实现更大规模的并行计算，并且具有更好的可扩展性。

并行程序代码：（openMP）

```
#include <iostream>
#include <ctime>
#include <cmath>
#include <omp.h>
using namespace std;

double buffon_omp(int l, int a, int b, int n, int num_threads) {
    double hit = 0;
    #pragma omp parallel num_threads(num_threads)
    {
        unsigned int seed = time(NULL) + omp_get_thread_num();
        #pragma omp for reduction(+:hit)
        for (int i = 0; i < n; ++i) {
            double base_x = a * (double)rand_r(&seed) / (double)(RAND_MAX);
            double base_y = b * (double)rand_r(&seed) / (double)(RAND_MAX);
            double cita = (double)rand_r(&seed);
            double pin_x = base_x + l * cos(cita);
            double pin_y = base_y + l * sin(cita);
            if (pin_x <= 0 || pin_x >= a || pin_y <= 0 || pin_y >= b) {
                hit++;
```

```
            }
        }
    }
    return hit / n;
}

int main() {
    int l, a, b, n;
    cin >> n >> a >> b >> l;

    double timings[4];

    for (int p = 0; p < 4; p++) {
        int num_threads[] = {2, 4, 6, 8};
        double start = omp_get_wtime();
        double pos = buffon_omp(l, a, b, n, num_threads[p]);
        double end = omp_get_wtime();
        timings[p] = (end - start) * 1000; // Convert to milliseconds
        cout << "Threads: " << num_threads[p] << "\tTime: " << timings[p] << " ms"
<< endl;
    }

    return 0;
}
```

并行程序代码（MPI）：

```
#include <iostream>
#include <ctime>
#include <cmath>
#include <mpi.h>
using namespace std;

double buffon_mpi(int l, int a, int b, int n, int rank, int size) {
    srand(time(NULL) + rank);
    double hit = 0;
    for (int i = rank; i < n; i += size) {
        double base_x = a * (double)rand() / (double)(RAND_MAX);
        double base_y = b * (double)rand() / (double)(RAND_MAX);
        double cita = (double)rand();
        double pin_x = base_x + l * cos(cita);
        double pin_y = base_y + l * sin(cita);
        if (pin_x <= 0 || pin_x >= a || pin_y <= 0 || pin_y >= b) {
            hit++;
        }
    }
    double total_hit;
    MPI_Reduce(&hit, &total_hit, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    return total_hit / (n / size);
}
```

```
int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int l, a, b, n;
    cin >> n >> a >> b >> l;

    double timings[4];

    for (int p = 0; p < 4; p++) {
        int num_procs[] = {2, 4, 6, 8};
        MPI_Barrier(MPI_COMM_WORLD);
        double start = MPI_Wtime();
        double pos = buffon_mpi(l, a, b, n, rank, num_procs[p]);
        double end = MPI_Wtime();
        timings[p] = (end - start) * 1000; // Convert to milliseconds

        if (rank == 0) {
            cout << "Processors: " << num_procs[p] << "\tTime: " << timings[p] << "
ms" << endl;
        }
    }

    MPI_Finalize();
    return 0;
}
```