Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Параллельные вычисления

Отчет по лабораторным работам

Определение площади набора кругов, заданных массивом с координатами центров и радиусами, методом Монте-Карло

Выполнила студентка гр. 3540901/11501 Дуботолкова Н.Д.
Принял преподаватель Стручков И. В.

Санкт-Петербург
2022

Содержание

Список лабораторных работ

1. многопоточные программы на C/C++ с использованием Pthreads
2. многопоточные программы с использованием OpenMP

Структура лабораторных работ

Для лабораторных работ студент выбирает один из предложенных алгоритмов и вариант реализации (MPI/OpenMP), для которого выполняется:

1. разработка алгоритма
2. создание последовательной программы, реализующей алгоритм
3. разработка тестового набора, оценка тестового покрытия
4. выделение частей для параллельной реализации, определение общих данных, анализ потенциального выигрыша
5. выбор способов синхронизации и защиты общих данных
6. разработка параллельной программы с использованием Pthreads
7. разработка параллельной программы с использованием MPI или OpenMP
8. отладка на имеющихся тестах, анализ и доработка тестового набора с учетом параллелизма
9. измерение производительности, сравнение с производительностью последовательной программы
10. анализ полученных результатов, доработка параллельной программы
11. написание отчета

1. Разработка алгоритма

Метод Монте-Карло относится к численному стохастическому приближенному математическому методу с использованием датчика случайных чисел.

Точность метода зависит от числа повторений опыта, то есть чем больше количество опытов (повторений), тем выше точность метода.

Итак, алгоритм расчета площади фигур методом Монте-Карло сводится к следующему:

1. Группа фигур «вписывается» в прямоугольник.
    a. Находим координаты прямых, ограничивающих круги:
        i. Находим наибольшую/наименьшую сумму координат x/y и радиусов соответствующих кругов

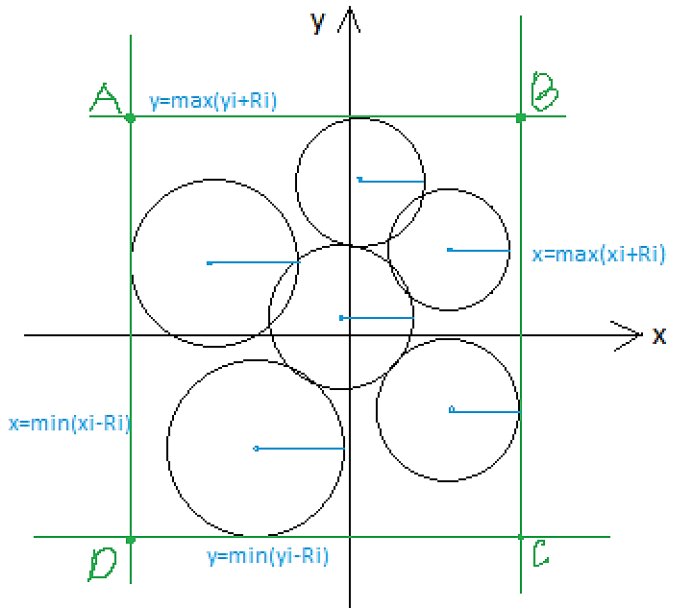

Рис.1.1. Схематичное изображение данных задачи

    b. Находим площадь прямоугольника, зная координаты его углов:
        - $A(\min(x_i - R_i)\,;\max(y_i + R_i))$
        - $B(\max(x_i + R_i)\,;\max(y_i + R_i))$
        - C $(\max(x_i + R_i)\,;\min(y_i - R_i))$
        - $AB = \sqrt{(A_x - B_x)^2 + \left(A_y - B_y\right)^2}$
        - $BC = \sqrt{(C_x - B_x)^2 + \left(C_y - B_y\right)^2}$
        - $S = AB * BC$
2. Псевдослучайным образом внутри прямоугольника генерируется большое количество точек:
    a. Генерируем координату X
    b. Генерируем координату Y

c. Проверяем, попала ли точка в один из кругов по формуле:

$$(x_i - x_0)^2 + (y_i - y_0)^2 \leq r_i^2$$

Если это условие хотя бы раз выполняется, увеличиваем счётчик попаданий на 1.

3. Посчитаем площадь исходной фигуры:
$s = \frac{n}{N} s_{\text{прямоугольника}}$, где n – кол-во попаданий, N – общее кол-во точек (задается нами).

2. Создание последовательной программы, реализующей алгоритм

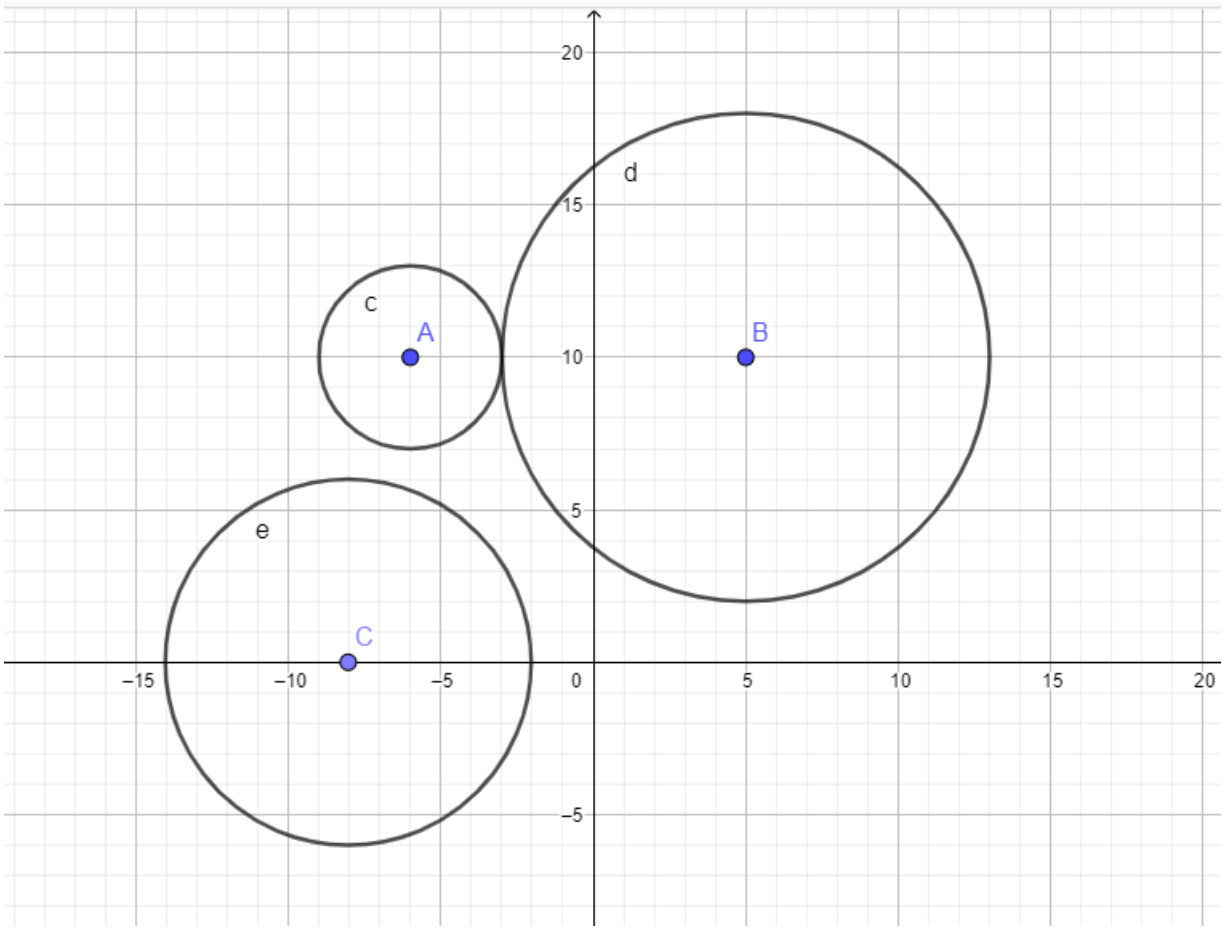

Рис.2.1 Визуальное представление кругов

Листинг 1.1 Последовательная программа

```
#include <iostream>
#include <ctime>
#include <iomanip>
#include <windows.h>
#include <cmath>
#include <ctime>

using namespace std;

const unsigned int DIM1 = 3; //три значения: x,y и r
const unsigned int DIM2 = 3; //кол-во кругов

int input_array[DIM1][DIM2] = {
    { -6, 10, 3 },
    { 5, 10, 8 },
    { -8, 0, 6 }
};

struct Center
{
    int x;
    int y;
};

struct Circle{
    Center center;
    int rad;
};

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Circle * array_of_circles = new Circle[DIM2];

    for (int i = 0; i < DIM1; i++) {
        array_of_circles[i].center.x = input_array[i][0];
        array_of_circles[i].center.y = input_array[i][1];
        array_of_circles[i].rad = input_array[i][2];
    }

    //находим уравнения прямых, ограничивающих круги
    int max_x, min_x, max_y, min_y;

    max_x = array_of_circles[0].center.x + array_of_circles[0].rad;
    min_x = array_of_circles[0].center.x - array_of_circles[0].rad;
    max_y = array_of_circles[0].center.y + array_of_circles[0].rad;
    min_y = array_of_circles[0].center.y - array_of_circles[0].rad;

    for (int i = 1; i < (DIM2); i++)
    {
        int temp = array_of_circles[i].center.x +
array_of_circles[i].rad;
        if (max_x < temp)
```

```
            max_x = temp;
        temp = array_of_circles[i].center.x -
array_of_circles[i].rad;
        if (min_x > temp)
            min_x = temp;
        temp = array_of_circles[i].center.y +
array_of_circles[i].rad;
        if (max_y < temp)
            max_y = temp;
        temp = array_of_circles[i].center.y -
array_of_circles[i].rad;
        if (min_y > temp)
            min_y = temp;
    }

    double height_rectangle = sqrt(pow((min_x-max_x),2)+pow((max_y-
max_y),2));
    double width_rectangle = sqrt(pow((max_x-max_x),2)+pow((min_y-
max_y),2));
    double s_rectangle = height_rectangle * width_rectangle;

    srand(time(0)); // генерация случайных чисел

    int positives = 100000000;

    int sum = 0;

    for (int i = 0; i < positives; i++)
    {
        double x0 = rand();
        double y0 = rand();
        x0 = x0/RAND_MAX*(max_x-min_x) + min_x;
        y0 = y0/RAND_MAX*(max_y-min_y) + min_y;

        bool t = false;
        for (int j = 0; j < DIM2; j++)
        {
            double len = pow((x0-array_of_circles[j].center.x),2) +
pow((y0-array_of_circles[j].center.y),2);
            len = sqrt(len);

            if (len <= array_of_circles[j].rad)
                t = true;
        }
        if (t) {
            sum++;
        }
    }

    double s = s_rectangle*((double)(sum)/(double)(positives));
    cout << "Result: " << s << endl;
    double check_sum;
    for (int j = 0; j < DIM2; j++){
        check_sum = check_sum + 3.1415926 *
pow(array_of_circles[j].rad, 2);
    }
    cout << "Check sum: " << check_sum << endl;
```

```
    cout << "runtime = " << clock()/1000.0 << endl; // время работы программы
    return 0;
}
```

Result: 342.395 //площадь, получившаяся методом Монте Карло
Check sum: 342.434 // площадь, подсчитанная по формуле
runtime = 11.791

3. Разработка тестового набора, оценка тестового покрытия

В качестве тестовых данных выбраны круги, которые не пересекаются друг с другом. Это было сделано для того, чтобы было проще считать площадь по обычной формуле. Программа будет запускаться несколько раз с разным количеством точек: 100, 10000, 100000000.

4. Выделение частей для параллельной реализации, определение общих данных, анализ потенциального выигрыша

В данной программе распараллелить можно момент «бросания» точек в прямоугольник, так как это самое долгое действие.

Тогда, схема распараллеливания выглядит следующим образом:

a. Каждый из p потоков генерирует и бросает N/p точек;
b. Глобальная сумма формируется в мастер-потоке.

Однако, здесь мы можем столкнуться с некорректным сложением сумм в получившихся потоках. Для этого ограничим доступ к переменной, которая считает общее количество попаданий.

5. Выбор способов синхронизации и защиты общих данных

В качестве примитива синхронизации были выбраны мьютексы.

6. Разработка параллельной программы с использованием Pthreads

Листинг 6.1 Параллельная программа с использованием Pthreads

```
#include <iostream>
#include <ctime>
#include <windows.h>
#include <cmath>
#include <ctime>
#include <pthread.h>

using namespace std;

#define NUMBER_OF_POINTS 100000000
#define NUMBER_OF_THREADS 4
```

```
const unsigned int DIM1 = 3; //три значения: x,y и r
const unsigned int DIM2 = 3; //кол-во кругов

int point_count = 0; //кол-во точек в кругах

int input_array[DIM1][DIM2] = {
    { -6, 10, 3 },
    { 5, 10, 8 },
    { -8, 0, 6 }
};

int MIN_X, MIN_Y, MAX_X, MAX_Y;

pthread_mutex_t mutex;

struct Center
{
    int x;
    int y;
};

struct Circle{
    Center center;
    int rad;
};

Circle * array_of_circles = new Circle[DIM2];

void FindRectangleArea(){
   MAX_X = array_of_circles[0].center.x + array_of_circles[0].rad;
       MIN_X = array_of_circles[0].center.x -
array_of_circles[0].rad;
       MAX_Y = array_of_circles[0].center.y +
array_of_circles[0].rad;
       MIN_Y = array_of_circles[0].center.y -
array_of_circles[0].rad;
   for (int i = 1; i < (DIM2); i++)
       {
              int temp = array_of_circles[i].center.x +
array_of_circles[i].rad;
              if (MAX_X < temp)
                     MAX_X = temp;
              temp = array_of_circles[i].center.x -
array_of_circles[i].rad;
              if (MIN_X > temp)
                     MIN_X = temp;
              temp = array_of_circles[i].center.y +
array_of_circles[i].rad;
              if (MAX_Y < temp)
                     MAX_Y = temp;
              temp = array_of_circles[i].center.y -
array_of_circles[i].rad;
              if (MIN_Y > temp)
                     MIN_Y = temp;
       }
}

float FindRectangleSquare(){
```

```
    float height_rectangle = sqrt(pow((MIN_X-MAX_X),2)+pow((MAX_Y-
MAX_Y),2));
        float width_rectangle = sqrt(pow((MAX_X-MAX_X),2)+pow((MIN_Y-
MAX_Y),2));
        return height_rectangle * width_rectangle;
}

double random_double()
{
     return rand() / ((double)RAND_MAX + 1);
}

void *CountThrowBusters(void *param)
{
   int POINTS;
   POINTS = *((int *)param);
   int hit_count = 0;
   double x0, y0;

   for (int i = 0; i < POINTS; i++) {
      x0 = random_double()*(MAX_X-MIN_X) + MIN_X;
      y0 = random_double()*(MAX_Y-MIN_Y) + MIN_Y;
      // cout << "x0: "<< x0 << "; " << "y0: " << y0 << endl;
      bool t = false;
      for (int j = 0; j < DIM2; j++)
      {
         double len = pow((x0-array_of_circles[j].center.x),2) +
pow((y0-array_of_circles[j].center.y),2);
         len = sqrt(len);
         if (len <= array_of_circles[j].rad)
            t = true;
      }
      if (t) {
         hit_count++;
      }
   }

   pthread_mutex_lock(&mutex);
   point_count += hit_count;
   pthread_mutex_unlock(&mutex);

   pthread_exit(0);
}

int main(int argc, const char * argv[])
{
   SetConsoleCP(1251);
   SetConsoleOutputCP(1251);

   //инициализация структур
   for (int i = 0; i < DIM1; i++) {
        array_of_circles[i].center.x = input_array[i][0];
        array_of_circles[i].center.y = input_array[i][1];
        array_of_circles[i].rad = input_array[i][2];
   }

   FindRectangleArea();
   //находим площадь прямоугольника
```

```
    double s_rectangle = FindRectangleSquare();

    int points_per_thread = NUMBER_OF_POINTS/ NUMBER_OF_THREADS;
    int i;
    pthread_t workers[NUMBER_OF_THREADS];

    pthread_mutex_init(&mutex,NULL);

    srand(time(0)); // генерация случайных чисел

    for (i = 0; i < NUMBER_OF_THREADS; i++)
       pthread_create(&workers[i], 0, CountThrowBusters,
&points_per_thread);

    for (i = 0; i < NUMBER_OF_THREADS; i++)
       pthread_join(workers[i],NULL);

    double s =
s_rectangle*((double)(point_count)/(double)(NUMBER_OF_POINTS));
    cout << "s_rectangle: " << s_rectangle <<"; " << "point_count: "
<< point_count << "; " << "NUMBER_OF_POINTS: "<< NUMBER_OF_POINTS <<
endl;
    cout << "NUMBER OF POINTS = " << NUMBER_OF_POINTS << endl;
    cout << "Result: " << s << endl;
        double check_point_count;
        for (int j = 0; j < DIM2; j++){
               check_point_count = check_point_count + 3.1415926 *
pow(array_of_circles[j].rad, 2);
        }
        cout << "Check result (formula): " << check_point_count <<
endl;

    cout << "Runtime = " << clock()/1000.0 << endl; // время работы
программы
    return 0;
}
```

7. Разработка параллельной программы с использованием OpenMP

Листинг 7.1 Параллельная программа с использованием OpenMP

```
#include <iostream>
#include <ctime>
#include <windows.h>
#include <cmath>
#include <ctime>
#include <omp.h>

using namespace std;

#define NUMBER_OF_POINTS 1000000000

const unsigned int DIM1 = 3; //три значения: x,y и r
const unsigned int DIM2 = 3; //кол-во кругов
```

```
int point_count = 0; //кол-во точек в кругах

int input_array[DIM1][DIM2] = {
    { -6, 10, 3 },
    { 5, 10, 8 },
    { -8, 0, 6 }
};

int MIN_X, MIN_Y, MAX_X, MAX_Y;

struct Center
{
    int x;
    int y;
};

struct Circle{
    Center center;
    int rad;
};

Circle * array_of_circles = new Circle[DIM2];

void FindRectangleArea(){
   MAX_X = array_of_circles[0].center.x + array_of_circles[0].rad;
       MIN_X = array_of_circles[0].center.x -
array_of_circles[0].rad;
       MAX_Y = array_of_circles[0].center.y +
array_of_circles[0].rad;
       MIN_Y = array_of_circles[0].center.y -
array_of_circles[0].rad;
   // cout << " " << MAX_X << " " <<  MIN_X << " " << MAX_Y << " " <<
MIN_Y << endl;
   for (int i = 1; i < (DIM2); i++)
       {
              int temp = array_of_circles[i].center.x +
array_of_circles[i].rad;
              if (MAX_X < temp)
                     MAX_X = temp;
              temp = array_of_circles[i].center.x -
array_of_circles[i].rad;
              if (MIN_X > temp)
                     MIN_X = temp;
              temp = array_of_circles[i].center.y +
array_of_circles[i].rad;
              if (MAX_Y < temp)
                     MAX_Y = temp;
              temp = array_of_circles[i].center.y -
array_of_circles[i].rad;
              if (MIN_Y > temp)
                     MIN_Y = temp;
       }
}

float FindRectangleSquare(){
   float height_rectangle = sqrt(pow((MIN_X-MAX_X),2)+pow((MAX_Y-
MAX_Y),2));
```

```
        float width_rectangle = sqrt(pow((MAX_X-MAX_X),2)+pow((MIN_Y-
MAX_Y),2));
        return height_rectangle * width_rectangle;
}

double random_double()
{
    return rand() / ((double)RAND_MAX + 1);
}

int main(int argc, const char * argv[])
{
   SetConsoleCP(1251);
   SetConsoleOutputCP(1251);

   //инициализация структур
   for (int i = 0; i < DIM1; i++) {
        array_of_circles[i].center.x = input_array[i][0];
        array_of_circles[i].center.y = input_array[i][1];
        array_of_circles[i].rad = input_array[i][2];
   }

   FindRectangleArea();
   //находим площадь прямоугольника
   double s_rectangle = FindRectangleSquare();

   int i;

   srand(time(0)); // генерация случайных чисел
   #pragma omp parallel
   {
      int hit_count = 0;
      double x0, y0;
      #pragma omp for private(hit_count)
      {
         for (int i = 0; i < NUMBER_OF_POINTS; i++) {
                  x0 = random_double()*(MAX_X-MIN_X) + MIN_X;
                  y0 = random_double()*(MAX_Y-MIN_Y) + MIN_Y;
                  // cout << "x0: "<< x0 << "; " << "y0: " << y0 <<
endl;
                  bool t = false;
                  for (int j = 0; j < DIM2; j++)
                  {
                     double len = pow((x0-
array_of_circles[j].center.x),2) + pow((y0-
array_of_circles[j].center.y),2);
                     len = sqrt(len);
                     if (len <= array_of_circles[j].rad)
                        t = true;
                  }
                  if (t) {
                     hit_count++;
                  }
               }
      }

      point_count += hit_count;
   }
```

```
   double s =
s_rectangle*((double)(point_count)/(double)(NUMBER_OF_POINTS));
   cout << "s_rectangle: " << s_rectangle <<"; " << "point_count: "
<< point_count << "; " << "NUMBER_OF_POINTS: "<< NUMBER_OF_POINTS <<
endl;
   cout << "NUMBER OF POINTS = " << NUMBER_OF_POINTS << endl;
   cout << "Result: " << s << endl;
       double check_point_count;
       for (int j = 0; j < DIM2; j++){
              check_point_count = check_point_count + 3.1415926 *
pow(array_of_circles[j].rad, 2);
       }
       cout << "Check result (formula): " << check_point_count <<
endl;

   cout << "Runtime = " << clock()/1000.0 << endl; // время работы
программы
   return 0;
}
```

8. Отладка на имеющихся тестах, анализ и доработка тестового набора с учетом параллелизма

Во время тестирования не рассматривались варианты меньше 100, так как там особой разницы в производительности не наблюдалось, а также была большая погрешность в вычислении.

9. Измерение производительности, сравнение с производительностью последовательной программы

Таблица 9.1. Сводная таблица результатов

	Потоки	Количество точек				Результат по формуле
		100	10000	1000000	1000000000	
Последовательная	-	349.92	342.338	342.288	342.423	342.434
Pthreads	2	285.12	341.755	342.554	342.428	
	4	233.28	346.032	342.815	342.402	
	6	272.16	353.03	343.112	342.402	
OpenMP	2	349.92	339.552	342.28	342.434	

Таблица 9.2. Сводная таблица временных результатов

	Потоки	Количество точек			
		100	10000	1000000	1000000000
Последовательная	-	0.004	0.006	0.126	12.261
Pthreads	2	0.006	0.005	0.078	65.088
	4	0.003	0.003	0.072	39.796
	6	0.004	0.004	0.039	32.296
OpenMP	-	0.006	0.007	0.131	123.268

10. Анализ полученных результатов, доработка параллельной программы

Изначально, в программе, которая использует OpenMP было только одно место, которое будет распараллеливать код: *#pragma omp parallel*. Однако, OpenMP умеет работать с циклами, поэтому перед циклом for я добавила *#pragma omp for private(hit_count)*. Дополнительно выделила переменную hit_count как приватную, так как счётчик попаданий должен быть у каждого потока свой. Цикл можно распараллелить, так как здесь нет зависимости по данным, принудительных выходов, а условие проверяется путём сравнения с переменной.

Другое улучшение также касается OpenMP. Как уже говорилось ранее, у нас есть критическая секция, которая складывает значения попаданий. Её необходимо явно выделить, чтобы избежать ошибок в выполнении:

Листинг 10.1. Выделение критической секции

```
#pragma omp critical
        point_count += hit_count;
```

Также, в OpenMP можно явно задавать количество потоков, поэтому сводные таблицы были дополнены.

Таблица 9.1. Сводная таблица результатов

	Потоки	Количество точек				Результат по формуле
		100	10000	1000000	1000000000	
Последовательная	-	349.92	342.338	342.288	342.423	342.434
Pthreads	2	285.12	341.755	342.554	342.428	
	4	233.28	346.032	342.815	342.402	
	6	272.16	353.03	343.112	342.402	
OpenMP	2	336.96	348.494	342.363	342.438	
	4	362.88	337.219	342.428	342.44	
	6	375.84	346.874	341.887	342.434	

Таблица 9.2. Сводная таблица временных результатов

	Потоки	Количество точек			
		100	10000	1000000	1000000000
Последовательная	-	0.004	0.006	0.126	12.261
Pthreads	2	0.006	0.005	0.078	65.088
	4	0.003	0.003	0.072	39.796
	6	0.004	0.004	0.039	32.296
OpenMP	2	0.005	0.003	0.129	128.473
	4	0.004	0.008	0.133	120.35
	6	0.002	0.003	0.132	126.272

Выводы

В данной работе были рассмотрены методы распараллеливания программ с использованием OpenMP и Pthreads.

Реализация на OpenMP заняла меньшее количество строк кода, по сравнению с Pthreads. Например, в Pthreads необходимо использовать функцию pthread_join для синхронизации потоков, в то время как в OpenMP это контролирует сам фреймворк.

Из экспериментов можно сделать вывод, что на маленьком количестве точек, точность и скорость будут лучше у последовательной программы. На больших значениях точность лучше у OpenMP, однако, время выполнения практически в 3 рада выше. Причиной этому может быть неправильно составленный алгоритм. Однако, если нам более важна точность, то данный алгоритм точен с вероятностью 99%.

Для более точных и обоснованных результатов необходимо провести некоторое количество испытаний, посчитать математическое ожидание, дисперсию и доверительный интервал.