

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Кафедра компьютерных систем и программных технологий

# Параллельные вычисления»

Отчет по лабораторной работе

Создание многопоточных программ на языке C++ с использованием  
OpenMP

**Работу**

**выполнил:**

Иванов А.А.

Группа:

3540901/91502

**Преподаватель:**

Стручков И.В.

Санкт-Петербург  
2020

# Содержание

<b>1. Цель работы</b>	<b>3</b>
<b>2. Программа работы</b>	<b>3</b>
<b>3. Характеристики системы</b>	<b>3</b>
<b>4. Ход работы</b>	<b>4</b>
4.1. Структура проекта . . . . .	4
4.2. Реализация метода . . . . .	4
4.3. Вспомогательные функции . . . . .	4
4.4. Реализация метода подсчета площади . . . . .	5
4.5. Параллельный алгоритм с использованием OpenMP . . . . .	6
<b>5. Тестирование</b>	<b>6</b>
<b>6. Выводы</b>	<b>8</b>

## 1. Цель работы

**Вариант 9, OpenMP** Определение площади набора кругов, заданных массивом с координатами центров и радиусами, методом Монте-Карло.


## 2. Программа работы

1. Для алгоритма из полученного задания написать последовательную программу на языке С или С++, реализующую этот алгоритм.
2. Для созданной последовательной программы необходимо написать 3-5 тестов, которые покрывают основные варианты функционирования программы.
3. Проанализировать полученный алгоритм, выделить части, которые могут быть распараллелены, разработать структуру параллельной программы. Определить количество используемых потоков, а также правила и используемые объекты синхронизации.
4. Согласовать разработанную структуру и детали реализации параллельной программы с преподавателем.
5. Написать код параллельной программы и проверить ее корректность на созданном ранее наборе тестов. При необходимости найти и исправить ошибки.
6. Провести эксперименты для оценки времени выполнения последовательной и параллельной программ. Проанализировать полученные результаты.
7. Сделать общие выводы по результатам проделанной работы

## 3. Характеристики системы

Работа производилась на реальной системе, со следующими характеристиками:

Windows 7 Максимальная  
© Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.  
Service Pack 1



---

Система

Оценка:	<b>5,9</b> Индекс производительности Windows
Процессор:	Intel(R) Core(TM) i5-4690 CPU @ 3.50GHz 3.50 GHz
Установленная память (ОЗУ):	8,00 ГБ (7,90 ГБ доступно)
Тип системы:	64-разрядная операционная система

Рисунок 3.1. Сведения о системе

Количество ядер процессора - 4.

## 4. Ход работы

### 4.1. Структура проекта

Проект состоит из трех файлов

1. Main.cpp
2. MonteKarlo.h
3. MonteKarlo.cpp

Точка входа, расположена в файле **Main.cpp**, в котором вызываются необходимые функции, реализованные в **MonteKarlo.cpp**.

### 4.2. Реализация метода

Суть подсчета площади методом Монте-Карло состоит в том, чтобы нарисовать вокруг фигуры, площадь которой является искомой областью. Область должна удовлетворять двум условиям

1. Фигура должна полностью лежать внутри области
2. Площадь данной области должна быть известна

Далее внутри области случайным образом ставится большое количество точек. В данном проекте использовалось от 300 000 точек и больше.

После этого подсчитывается количество точек, которые попали внутрь искомой фигуры. Площадь фигуры вычисляется по формуле

Площадь фигуры = площадь области \* количество попавших точек / общее количество точек

### 4.3. Вспомогательные функции

Вспомогательными функциями в данном проекте являются

- distance - функция вычисления расстояния между двумя точками
- isInside - функция проверки точки на положение внутри круга
- square - функция вычисления площади прямоугольника по крайним точкам

Код функций приведен ниже

```
1 float distance(float x1, float y1, float x2, float y2)
2 {
3     return sqrt(pow((x2 - x1), 2) + pow((y2 - y1), 2));
4 }
5
6 bool isInside(float x, float y, float rounds[3][numberOfRounds]) {
7     for (int j = 0; j < numberOfRounds; j++)
8     {
9         if (distance(x, y, rounds[0][j], rounds[1][j])
10             <= rounds[2][j])
11             return true;
12     }
13     return false;
14 }
15 float square(float x1, float x2, float y1, float y2) {
16     return (distance(x1, 0, x2, 0) * distance(0, y1, 0, y2));
17 }
```

## 4.4. Реализация метода подсчета площади

Изначально задаются две глобальные переменные -

1. numberOfRounds - количество заданных кругов
2. numberOfDots - количество точек

Далее задаются необходимые переменные и массивы

- float area[4] - массив для определения границ области поиска
- float\* dotsX = new float[numberOfDots]; - массив для X-координат случайных точек
- float\* dotsY = new float[numberOfDots]; - массив для X-координат случайных точек
- float rounds[3][numberOfRounds]; - массив для исходных данных (кругов)
- int dotsInside = 0; - счетчик для точек, которые попали внутрь кругов

Массивы для случайных точек задаются динамически во избежание переполнения стека.

По условию круги заданы координатами центров и радиусами. Для их хранения используется двумерный массив.

Далее следует определение границ области поиска. Для каждого круга вычисляется крайние точки по X и Y, затем максимальные и минимальные значения заносятся в массив area.

```
1  area[0] = rounds[0][0] + rounds[2][0]; //max X
2  area[1] = rounds[0][0] - rounds[2][0]; //min X
3  area[2] = rounds[1][0] + rounds[2][0]; //max Y
4  area[3] = rounds[1][0] - rounds[2][0]; //min Y
5
6  for (int i = 0; i < numberOfRounds; i++)
7  {
8      if (rounds[0][i] + rounds[2][i] > area[0])
9          area[0] = rounds[0][i] + rounds[2][i];
10     if (rounds[0][i] - rounds[2][i] < area[1])
11         area[1] = rounds[0][i] - rounds[2][i];
12     if (rounds[1][i] + rounds[2][i] > area[2])
13         area[2] = rounds[1][i] + rounds[2][i];
14     if (rounds[1][i] - rounds[2][i] < area[3])
15         area[3] = rounds[1][i] - rounds[2][i];
16 }
```

Далее следуют два цикла. Первый - цикл генерации случайных точек с помощью функции генерации случайных чисел rand() в пределах области поиска.

Второй - цикл определения количества точек, попадающих в круги. От координат X и Y каждой случайной точки, а так же от массива с начальными данными вызывается функция isInside, в случае возвращения true увеличивается счетчик.

```
1  for (int j = 0; j < numberOfDots; j++) {
2      dotsX[j] = (float)(rand() % (int)(area[0] - area[1]) + area[1]);
3      dotsY[j] = (float)(rand() % (int)(area[2] - area[3]) + area[3]);
4  }
5
6  for (int i = 0; i < numberOfDots; i++)
7  {
8      if (isInside(dotsX[i], dotsY[i], rounds))
9          dotsInside++;
10 }
11
```

```

12  return square(area[0], area[1], area[2], area[3]) * dotsInside / numberOFDots;
13 }

```

## 4.5. Параллельный алгоритм с использованием OpenMP

В данной программе распараллеливанию подлежат два цикла `for`.

В общем случае в OpenMP для распараллеливания цикла используется директива **`#pragma omp parallel for`**

В случае цикла для генерации случайных точек итерации цикла независимы друг от друга, поэтому дополнительных параметров указывать нет необходимости

В случае с подсчетом вошедших внутрь круга точек нам требуется использовать редукцию. Директива в таком случае выглядит так **`#pragma omp parallel for reduction(+:dotsInside)`**

```

1 #pragma omp parallel for
2   for (int j = 0; j < numberOFDots; j++) {
3       dotsX[j] = (float)(rand() % (int)(area[0] - area[1]) + area[1]);
4       dotsY[j] = (float)(rand() % (int)(area[2] - area[3]) + area[3]);
5   }
6
7
8 #pragma omp parallel for reduction(+:dotsInside)
9   for (int i = 0; i < numberOFDots; i++)
10  {
11      if (isInside(dotsX[i], dotsY[i], rounds))
12          dotsInside++;
13  }

```

## 5. Тестирование

Для тестирования были реализованы вспомогательные функции.

Замер времени расчетов проводится с помощью библиотеки **`chrono`**

Для измерения точности вычислений были выбраны случаи, в которых площадь можно измерить аналитически. Для простоты вычислений были взяты не соприкасающиеся между собой круги

```

1 #include "MonteKarlo.h"
2 #include <chrono>
3 #include <vector>
4 using namespace std;
5 using namespace std::chrono;
6 time_point<system_clock> start, stop;
7
8 double defaultSquare() {
9     float result;
10    start = std::chrono::system_clock::now();
11    result = consistentMethod();
12    stop = std::chrono::system_clock::now();
13
14    return duration_cast<duration<double>>(stop - start).count();
15 }
16
17 double openMPSquare() {
18     float result;
19    start = std::chrono::system_clock::now();
20    result = parallelMethod();

```

```

21  stop = std::chrono::system_clock::now();
22  return duration_cast<duration<double>>(stop - start).count();
23 }
24
25
26 int main() {
27     srand(time(0));
28     vector<double> time(50);
29
30     int size = time.size();
31     for (auto i = 0; i < size; i++) {
32         time[i] = openMPSquare();
33     }
34
35     double avg = 0;
36     for (auto& n : time)
37         avg += n;
38     avg /= size;
39
40
41     double d = 0;
42     for (auto& n : time)
43         d += pow(avg - n, 2);
44
45
46     d /= size == 1 ? 1 : size - 1;
47
48
49     double maxError = 2.6778 * pow(d / size, 0.5);
50
51
52     cout << "_:m=_ " << avg << "_ms,_d=_ " << d << endl;
53     cout << "99%_interval:_ " << avg << "_+_ " << maxError << "_ms" << endl << endl
54         << "\n";
55 }

```

Таблица 5.1

### Измерение точности вычислений

№	Количество кругов	Радиусы кру- гов	Аналитически	С распаралеле- живанием	Без распа- раллелива- ния
1	1	5	78.5	79.004	78.8956
2	2	5, 4	128.74	128.075	128.006
3	3	5, 4, 3	157	157.163	158.072

Погрешность измерений не превышает 1 процент. Разницы в погрешности между методами программного подсчета вычлнено не было.

Для ручного регулирования количества потоков была использована директива **num\_threads(x)**

Время работы программы напрямую зависит от количества кругов. Для эксперимента был выбран вариант с 3 кругами.

Расчеты производятся по 50 раз. Затем по полученным данным рассчитываются математическое ожидание и дисперсия.

Далее был выбран доверительный интервал 0.99. Для него коэффициент доверия для нашей выборки равен 2.6778. Затем рассчитываем предел погрешности.

Таблица 5.2

Результаты запусков			
Количество потоков	Мат. ожидание	Дисперсия	Доверительный интервал
Без распараллеливания	1.44288	9.33985e-05	0.00365985
1	1.42538	4.41378e-05	0.00251593
2	0.724841	0.000159528	0.00478313
4	0.662118	0.000443146	0.00797199
6	0.598294	0.000478088	0.00828032
8	0.552512	0.00172061	0.0157085
16	0.513269	0.00115779	0.0128857
32	0.492888	0.00353073	0.0225022

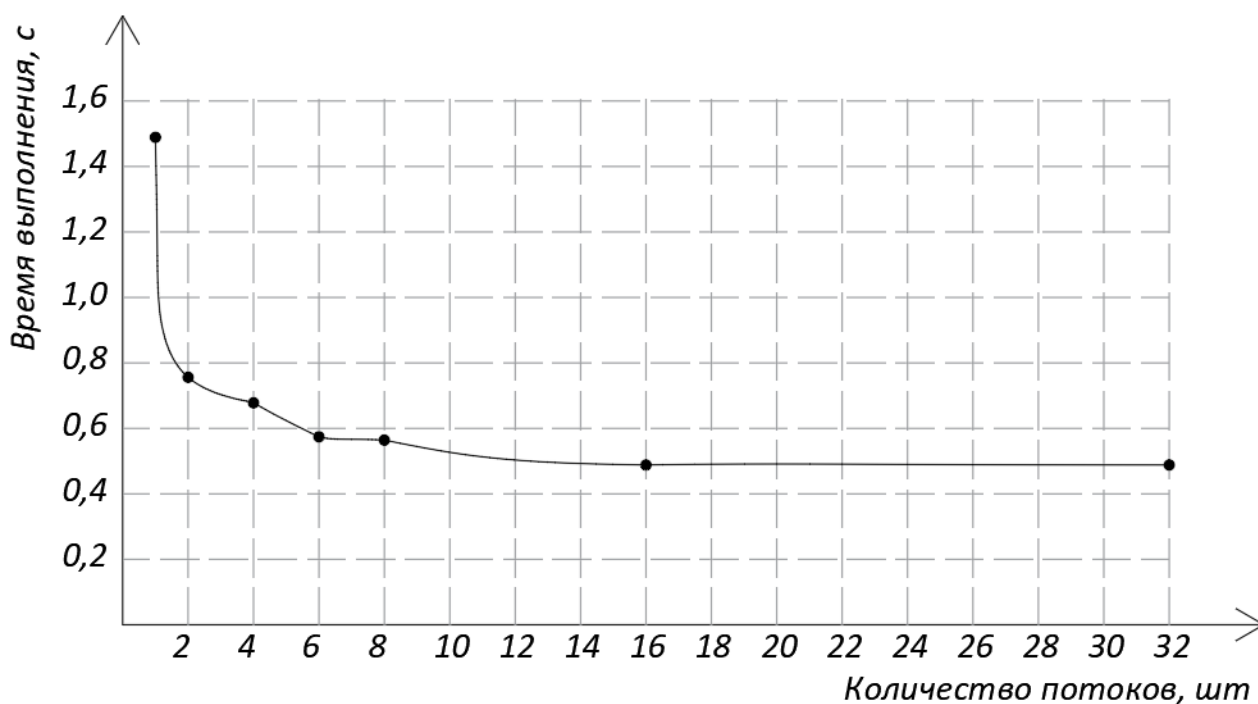


Рисунок 5.1. График зависимости времени выполнения от количества потоков

## 6. Выводы

В данной работе были рассмотрены методы распараллеливания программ с использованием OpenMP

OpenMP очень удобен в плане написания кода, поскольку его код является достаточно коротким и удобным.

Эксперименты показали, что распараллеливание кода влечет к приросту производительности. График роста похож на экспоненциальный и в данной конфигурации компьютера приходит к минимальному времени выполнения начиная с 16 потоков.

Задача, реализованная в работе очень удобна для распараллеливания.

В итоге работы можно сделать вывод, что распараллеливании программ имеет смысл в трудоемких задачах, в то время как в тривиальных задачах, последовательное решение будет быстрее.