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

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 C или C++, 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:

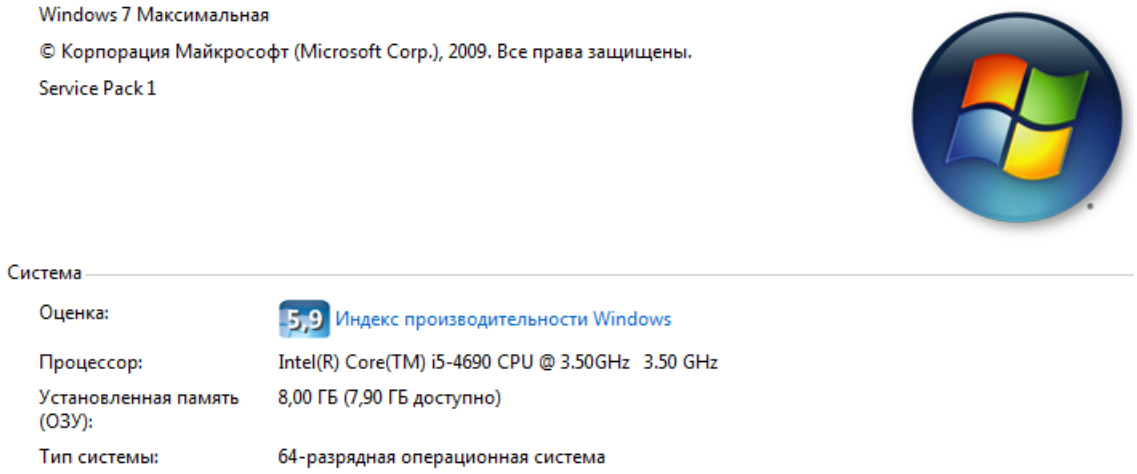

Рисунок 3.1. Сведения о системе

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 область. Область должны удовлетворять двум условиям

1. Фигура должны полностью лежать внутри области

2. Площать данной области должны быть известна

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

После этого подсчитывается количество точек, которы 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 динамечески во избежании 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 У, 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 У каждой случайной точки, а так же от массива с начальными даннными вызывается функция isInside, в случае возращения 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 деректива **#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)**

```
#pragma omp parallel for
  for (int j = 0; j < numberOFDots; j++) {
    dotsX[j] = (float)(rand() % (int)(area[0] - area[1]) + area[1]);
    dotsY[j] = (float)(rand() % (int)(area[2] - area[3]) + area[3]);
  }


#pragma omp parallel for reduction(+:dotsInside)
  for (int i = 0; i < numberOFDots; i++)
  {
    if (isInside(dotsX[i], dotsY[i], rounds))
      dotsInside++;
  }
```

5. Тестирование

Для тестирования были реализованы вспомогательные функции.

Замер времени рассчетов проводится с помощью библиотеки **chrono**

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

```
#include "MonteKarlo.h"
#include <chrono>
using namespace std;
using namespace std::chrono;
std::chrono::time_point<std::chrono::system_clock> start, stop;

void defaultSquare() {
  float result;
  start = std::chrono::system_clock::now();
  result = consistentMethod();
  stop = std::chrono::system_clock::now();

  cout<<"Default Time: "<< duration_cast<duration<double>>(stop - start).count()
    ↪ <<endl;
  cout<<"Default Square: "<<result<<endl;
}

void openMPSquare() {
  float result;
  start = std::chrono::system_clock::now();
  result = parallelMethod();
  stop = std::chrono::system_clock::now();

  cout << "OpenMP Time: " << duration_cast<duration<double>>(stop - start).count
    ↪ () << endl;
```

```
24    cout << "OpenMP Square: " << result << endl;
25 }
26
27
28 int main() {
29    float result1, result2;
30    srand(time(0));
31
32    defaultSquare();
33    openMPSquare();
34 }
```

Таблица 5.1

Измерение точности вычислений

№	Количество кругов	Радиусы кругов	Аналитически	С распараллеливанием	Без распараллеливания
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 процентаю. Разницы в погрешности между методами программного подсчета вычвлено не было.

Для ручного регулирования количества потоков была использована директива **num_threads(x)**

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

Таблица 5.2

Результаты запусков

Количество потоков	Время выполнения
Без распараллеливания	1.50609
1	1.48908
2	0.756043
4	0.678039
6	0.574033
8	0.564032
16	0.488028
32	0.488020

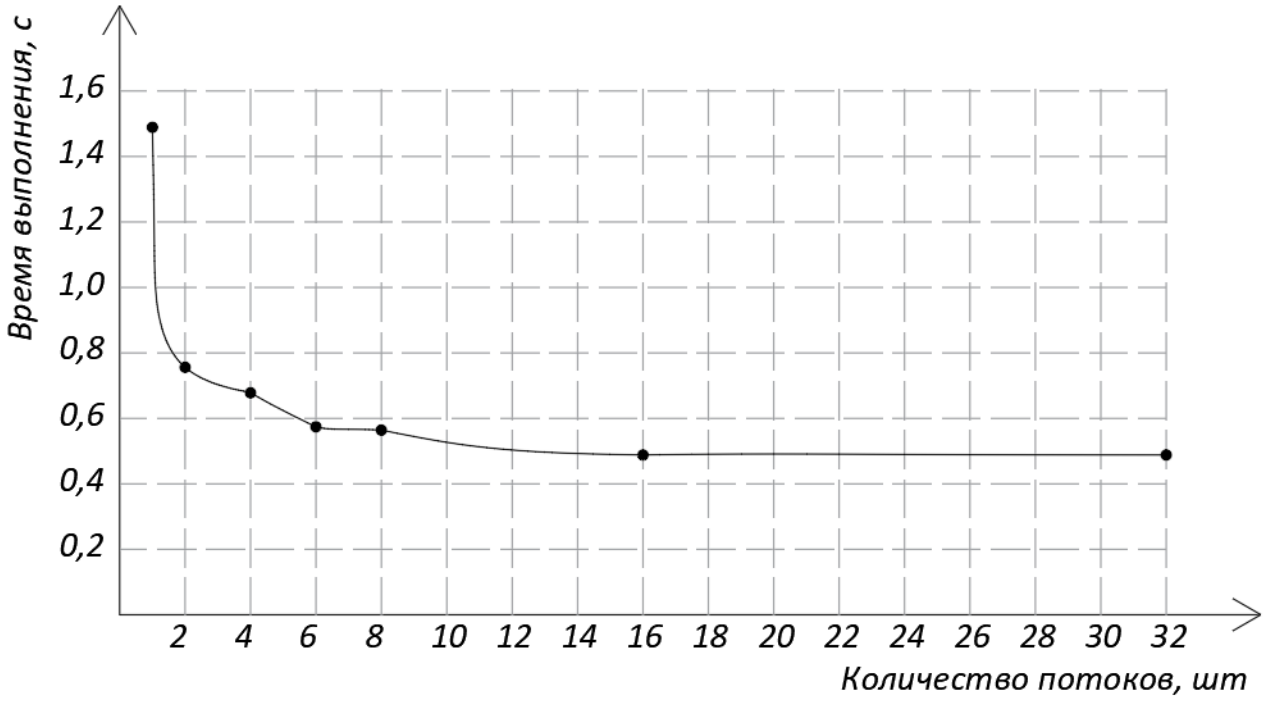

Рисунок 5.1. График зависимости времени выполнения от количества потоков

6. Выводы

В результате данной работы был получен опыт раоты с модулями ядра Linux. механизм модулей удобен тем, что модули не мешают ядру при згрузке, с помощью модулей можно добавлять в ядро широкий спектр функкций, а так же моуль начинает раоту не требуя перезагрузки. Так же данный механизм позволяет создавать драйвера для новых внешних устройств и тем самым быстро подстраиваться под выход новых компонентов ввода и вывода. Принцип работы и иерархия механизма модулей понятны и логичны.