ЛАБОРАТОРНАЯ РАБОТА №4	M3137	2023
openMP	Булавко Тимофей Евгеньевич	

Цель работы: знакомство с основами многопоточного программирования.

Инструментарий: работа выполнена на языке C++ (ISO c++ 14) . Стандарт OpenMP 2.0.

Вариант: easy

Дано ***r (радиус круга)*** и ***n (количество случайных точек)***. Нужно рассчитать площадь круга методом ***Монте-Карло***.

Конструкции openMP использованные в моей программе для распараллеливания команд:

#pragma omp parallel for – делит итерации следующего цикла между потоками.

#pragma omp atomic – конструкция синхронизации, используемая для корректной работы переменной (общей) в разных потоках.

schedule(type[, chunk]) – это условие контролирует, как итерации цикла распределяются между потоками, ***type = static*** - итерации равномерно распределяются. ***type = dynamic*** – итерации распределяются блоками заданного размера.

Описание работы написанного кода:

В качестве аргументов подается используемое количество потоков, в случае -1 openMP отключается, в случае 0 используется стандартное количество потоков, и название входного и выходного файла

```
int thread_count = atoi(string(argv[1]).c_str());
string input = argv[2];
string output = argv[3];
```

после происходит чтение r (радиус круга) и n (количество случайных точек) из входного файла.

```
ifstream in(input);
int r, n;
in >> r >> n;
in.close();
```

middle_x и middle_y – координаты центра круга

```
double middle_x = r, middle_y = r;
```

cnt – количество точек попавших в круг

start – начало работы параллельного кода

Дальше находится подсчет точек, попавших в круг. x и y – рандомная точка. После если эта точка находится на расстоянии меньше r от центра круга cnt синхронизируется и увеличивается на 1.

```
#pragma omp parallel for schedule(static) if (thread_count != -1)
	for (int i = 0; i < n; i++) {
		double x = (double(rand()%1000) / 1000) * r * 2;
		double y = (double(rand()%1000) / 1000) * r * 2;
		if ((middle_x - x) * (middle_x - x) + (middle_x - y) * (middle_y - y) <= r * r) {
#pragma omp atomic
			cnt++;
		}
	}
```

После окончания работы параллельного кода в переменную finish записывается время окончания работы, и считается время работы этой части кода, котрое после выводится в консоль.

```
double finish = omp_get_wtime();
double time = (finish - start) * 1000;
printf("Time(%i thread(s)) : %g ms\n", thread_count, time);
```

Дальше считается площадь круга и выводится в выходной файл. После программа заканчивает свою работу.

```
ofstream out(output);
double circle_erea = 4 * r * r * double(cnt) / double(n);
out << circle_erea << endl;
out.close();
return 0;
```

Результаты работы программы:

Все тесты проводились для r = 10, n = 100000000, на 12 потоках.

Среднее время работы: 2050 ms

Средняя погрешность: -1.066

Программа исполнялась на процессоре: intel core i7 10750H 2.6 GHz

Экспериментальная часть:

Во всех тестах используются значения r = 10, n = 100000000.

График 1

schedule type = static, y – время в миллисекундах, x – количество потоков

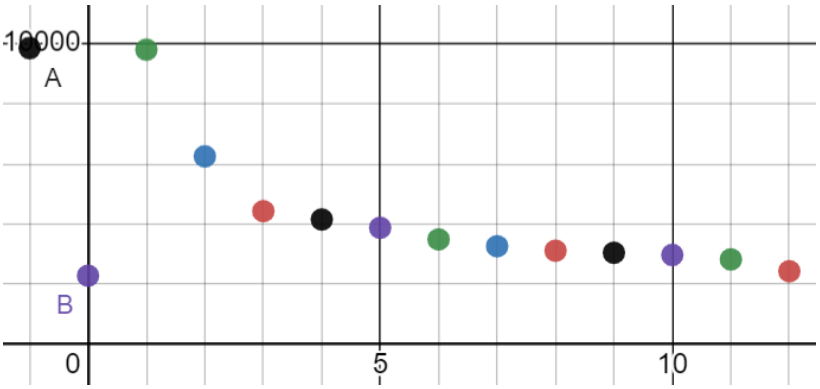
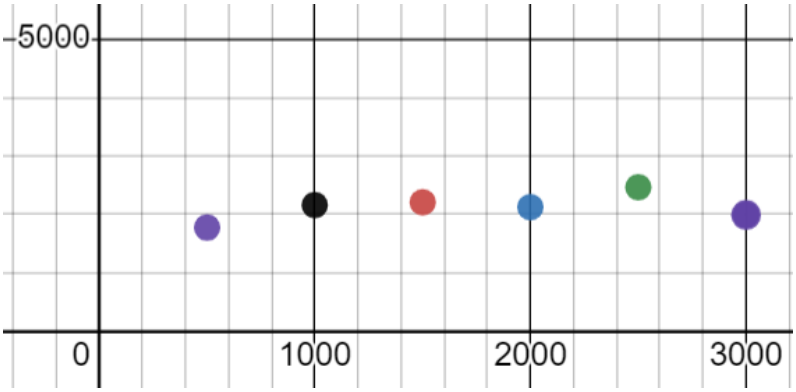


График 2

При 12 потоках, schedule type = dynamic, y – время работы в миллисекундах, x – количество чанков.



Листинг кода:

```
#include <fstream>
#include <omp.h>
#include <time.h>
#include <stdlib.h>

using namespace std;

int main(int argc, char* argv[]) {

	srand(time(0));

	int thread_count = atoi(string(argv[1]).c_str());
	string input = argv[2];
	string output = argv[3];

	ifstream in(input);
	int r, n;
	in >> r >> n;
	in.close();

	double middle_x = r, middle_y = r;
	thread_count = 12;
	if (thread_count > 0) {
		omp_set_num_threads(thread_count);
	}
	int cnt = 0;
	double start = omp_get_wtime();
#pragma omp parallel for schedule(dynamic, 3000) if (thread_count != -1)
	for (int i = 0; i < n; i++) {
		double x = (double(rand()%1000) / 1000) * r * 2;
		double y = (double(rand()%1000) / 1000) * r * 2;
		if ((middle_x - x) * (middle_x - x) + (middle_x - y) * (middle_y - y) <= r * r) {
#pragma omp atomic
			cnt++;
		}
	}

	double finish = omp_get_wtime();
	double time = (finish - start) * 1000;
	printf("Time(%i thread(s)) : %g ms\n", thread_count, time);

	ofstream out(output);
	double circle_erea = 4 * r * r * double(cnt) / double(n);
	out << circle_erea << endl;
	out.close();
	return 0;
}
```