Лабораторная работа №4	м3138	2023
OpenMp	Попович Виталий Сергеевич	

Цель работы: знакомство с основами многопоточного программирования.

Инструментарий и требования к работе: работа должна быть выполнена на C или C++. В отчете указать язык и компилятор, на котором вы работали. Стандарт OpenMP 2.0.

Вариант: Easy

Дан радиус r, и значение N.

Нужно рассчитать площадь круга методом Монте-Карло, используя N различных(случайных) точек.

Описание конструкций OpenMP для распараллеливания команд:

`#pragma omp parallel for` - указывает на то, что данный(следующий) цикл следует разделить по итерациям между потоками.

`schedule(type[,chunk])` - Этим условием контролируется то, как итерации цикла распределяются между потоками

При `schedule type = static` - итерации равномерно

распределяются по потокам.
При schedule type = dynamic – итерации распределяются пакетами(блоками) заданного размера.

#pragma omp atomic - конструкция синхронизации.Используется для корректной работы с общей переменной для разных потоков.

Описание работы написанного кода:

int pow(int x, int y) - бинарное возведение числа x в степень y.

double getRandomPoint(double left, double right) - функция возвращает случайное значение в диапазоне [left;right].

bool isPointIn(double x, double y) - проверяет, находится ли точка с координатами (x, y) в окружности заданной в условии.

int stringToInt(string str) - переводит данную строку str в число.(к примеру: строку “-1245” в число: -1245)

int main(int argc, char* argv[]) - основная функция программы разбита на 6 частей(ограниченных двумя пробелами в коде):
1.Считывание переданных аргументов из argv[]

2.Считывание данных из файла с данным названием

3.Расчет площади для N точек с использованием многопоточности. С помощью `#pragma omp parallel for` N итераций цикла разбивается между заданным количеством потоков, где в каждой итерации берется случайная точка(в диапазоне ([0, 2r], [0, 2r])) и проверяется, входит ли она в заданный круг, если да, то увеличивается общая переменная cnt(количество таких точек). Для решения проблемы синхронизации общей переменной между разными потоками используется `#pragma omp atomic`

4.Расчет/вывод затраченного времени

5.Расчет площади(точной и приблизительной(по методу Монте-карло))

6.Вывод приблизительной площадь в файл и закрытие потока вывода файла.

Результат работы:

В результате работы, алгоритмы рассчитывает площадь с некоторой погрешностью, в зависимости от заданного N.

При r = 13, и N = 50.000.000.

Алгоритм вычисляет площадь: 532.65(погрешность:-1.71)

С наилучшим временем работы: 1406.58 миллисекунд

На базе процессора: Intel Core i7-11800H 2.30GHz

Экспериментальная часть:

В запуске используется значения: r = 13, и N =

50.000.000.

Каждое значение каждой конфигурации было рассчитано как среднее по 3-ём запускам.Каждая точка в графике - какой-то замер на какой-то конфигурации.

График 1:

При количестве потоков 8, где schedule kind = dynamic

X - Количество чанков Y - Затраченное время в миллисекундах.

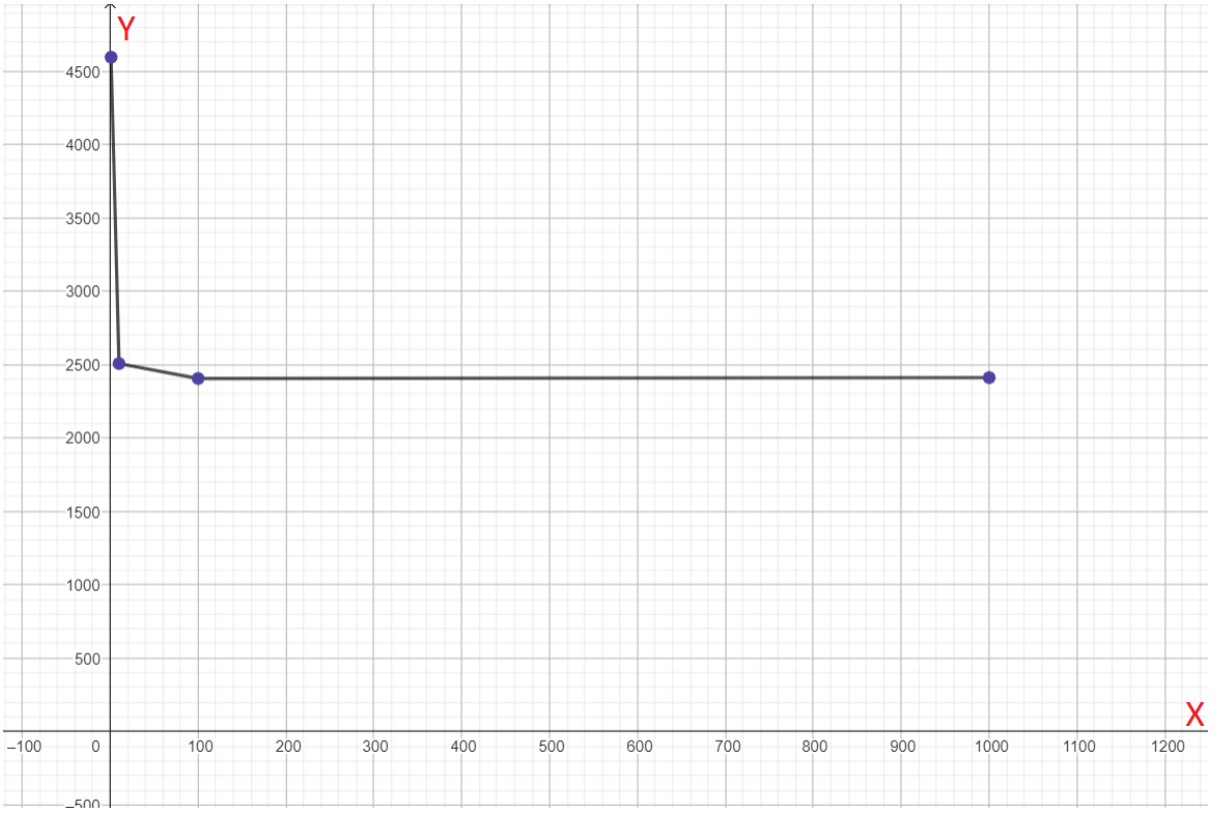


График 2:

При количестве потоков 8, где `schedule kind = static`

X - Количество чанков Y - Затраченное время в миллисекундах.

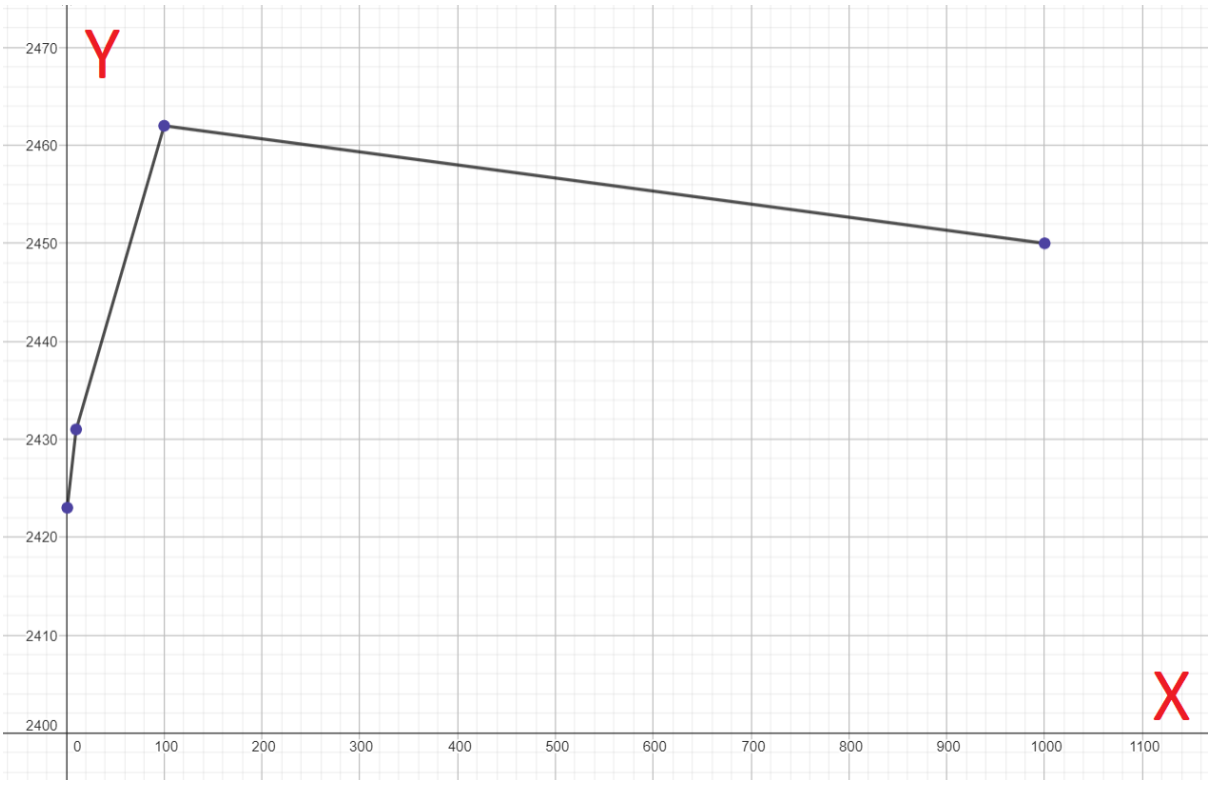


График 3:

Где `schedule kind = static. chunks` не установлены.

X - количество используемых потоков Y - Затраченное время в миллисекундах.

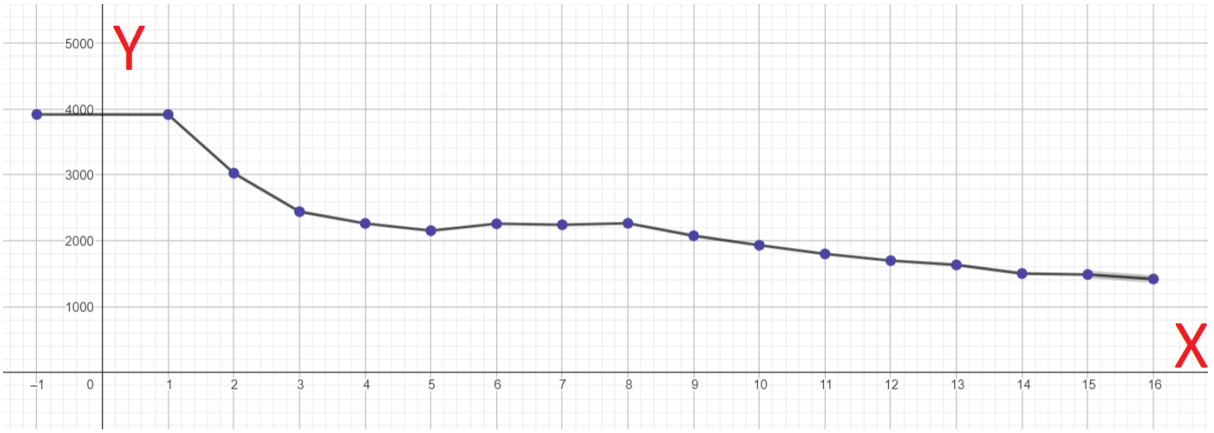
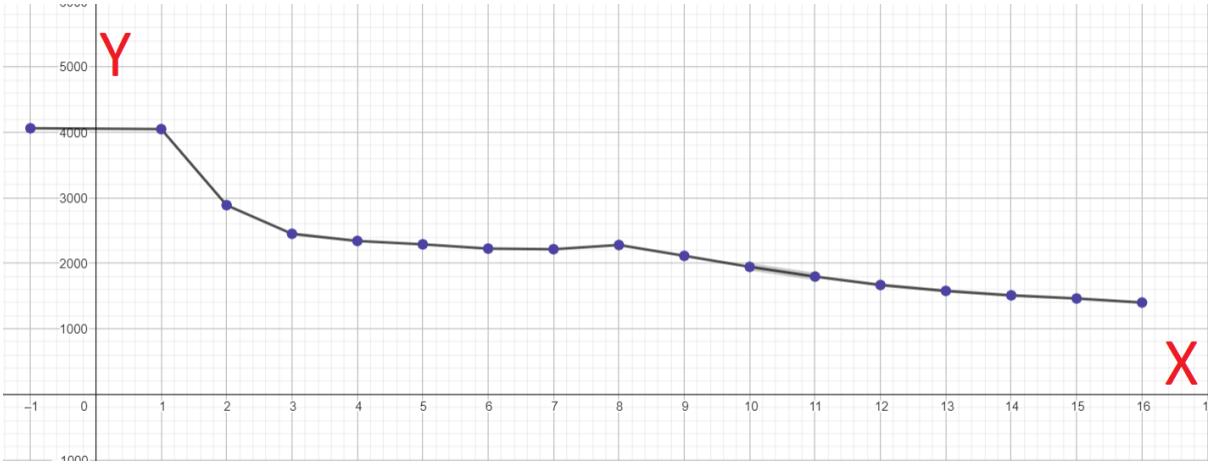


График 4:

Где `schedule kind = dynamic. chunks = 10000`

X - количество используемых потоков Y - Затраченное время в миллисекундах.



В результате измерений, наилучшее время достигается при `schedule kind = static`, с использованием 16-ти потоков.

Список источников:

Параллельное программирование на OpenMP

OpenMP C and C++ Application Program Interface

Листинг кода:

easy.cpp

```
#include <fstream>
```

```
#include <omp.h>
#include <time.h>
using namespace std;

const int precision = 3;
const double p = 3.14159265;

int radius, N;
double midX, midY;

int pow(int x, int y) {
	if (y == 0)
		return 1;
	int res = pow(x, y >> 1);
	res *= res;
	if (y % 2 != 0)
		res *= x;
	return res;
}

double getRandomPoint(double left, double right) {
	double point;
	point = rand() % int(pow(10, precision));
	point = left + (point/pow(10, precision)) * (right-left);
	return point;
}

bool isPointIn(double x, double y) {
	double distance = (midX-x)*(midX-x) + (midY-y)*(midY-y);
	return distance <= radius * radius;
}

int cnt = 0;
```

```
int stringToInt(string str) {
	int res = 0;
	int from = (str[0] == '-');
	for (int i = from; i < str.length(); i++) {
		int c = (str[i] - '0');
		res = res * 10 + c;
	}
	if (str[0] == '-')
		res *= -1;
	return res;
}

int main(int argc, char* argv[]) {
	srand(time(NULL));

	int nThread = stringToInt(argv[1]);
	string inName = argv[2];
	string outName = argv[3];

	ifstream in(inName);
	if (!in.is_open()) {
		printf("Error in opening input file!");
		exit(1);
	}
	in >> radius >> N;
	in.close();

	midX = midY = radius;

	if (nThread > 0) {
		omp_set_num_threads(nThread);
	}
```

```
	double Tstart = omp_get_wtime();
#pragma omp parallel for schedule(static) if (nThread !=-1)
	for (int i = 0; i < N; i++) {
		double x = getRandomPoint(0, radius + radius);
		double y = getRandomPoint(0, radius + radius);
		if (isPointIn(x, y)) {

#pragma omp atomic
			cnt++;
		}
	}

	printf("Time (msec): %f", (omp_get_wtime() - Tstart) *
1000);

	double subArea = 2 * radius * 2 * radius;
	double area = double(cnt) / double(N);
	area = subArea * area;

	ofstream out(outName);
	out << area << endl;
	out.close();

	return 0;
}
```