Лабораторная работа №4	37	2023
OpenMP	Илляхунов Ансар Адылович	

Цель работы: знакомство с основами многопоточного программирования

Инструментарий и требования к работе: работа выполнена на языке C++, использован компилятор msvc (компилятор от Microsoft Visual Studio) и стандартный OpenMP 2.0.

Вариант: easy.

Описание использованных конструкций OpenMP.

#pragma omp parallel – определяет параллельную область (parallel region), которая выполняется несколькими потоками одновременно. Это фундаментальная конструкция, которая запускает параллельное выполнение.

Условие *if (threadCnt != -1)* позволяет распараллелить область только в указанном случае.

Если в параллельной области встречается оператор цикла, то, согласно общему правилу, он будет выполнен всеми потоками, т. е. каждый поток выполнит все итерации этого цикла. Для распределения итераций цикла между различными потоками используется директива *#pragma omp for*

Условие указывает, как итерации цикла for распределяются между потоками команды.

При задании параметра *schedule(static, chunk_size)* итерации разбиваются на фрагменты, размер которых определяется параметром chunk_size. Фрагменты статически назначаются потокам в группе в циклическом порядке номеров потоков. Когда chunk_size не указан, пространство итераций делится на фрагменты примерно одинакового размера, при этом каждому потоку назначается один фрагмент.

При задании параметра *schedule(dynamic, chunk_size)* происходит такое же деление на фрагменты, но каждый фрагмент назначается потоку, ожидающему назначения. Поток выполняет свой фрагмент, а затем ожидает своего следующего назначения, пока не останется ни одной части для назначения. Когда chunk_size не указан, по умолчанию он равен 1.

#pragma omp atomic гарантирует, что определенное место в памяти обновляется атомарно и не подвергается одновременному изменению из

нескольких потоков. Таким образом, она синхронизирует взаимодействие различных потоков с одной общей переменной.

Описание работы написанного кода:

В начале программы происходит ввод и обработка данных из файла и аргументов программы (кол-во потоков threadCnt, радиус r, кол-во рандомных точек n) вместе с базовой обработкой ошибок.

Дальше идет основная часть алгоритма – параллельная область, состоящая из одного цикла for, итерации которого распределяются по потокам с помощью директивы #pragma omp for. Итерация алгоритма подразумевает проверку попадания точки, рандомно сгенерированной через функцию getRand (число в диапозоне [-r, r]), в окружность радиуса r с помощью функции pointIn. И в случае положительного результата происходит атомарное увеличение счетчика через директиву #pragma omp atomic, так что изменения над ним синхронизируются.

В заключительной части происходит оценка площади круга, как произведение площади квадрата (4 * r * r) на отношение числа попадания в круг (circ) к общему числу сгенерированных точек (n).

Результат работы:

Программа высчитывает площадь круга с некой погрешностью в зависимости от значения n.

Для заданных значений радиуса r = 17, n = 10.000.000 результат алгоритма: 911.607 (погрешность -3.686).

Среднее время работы: 184.944 ms

Процессор: AMD Ryzen 5 5600H (12CPUs), ~3.3GHz.

Экспериментальная часть

Эксперименты проводились при фиксированных значениях:

r = 17 и n = 10.000.000. Время было замерено как среднее по 3 запускам.

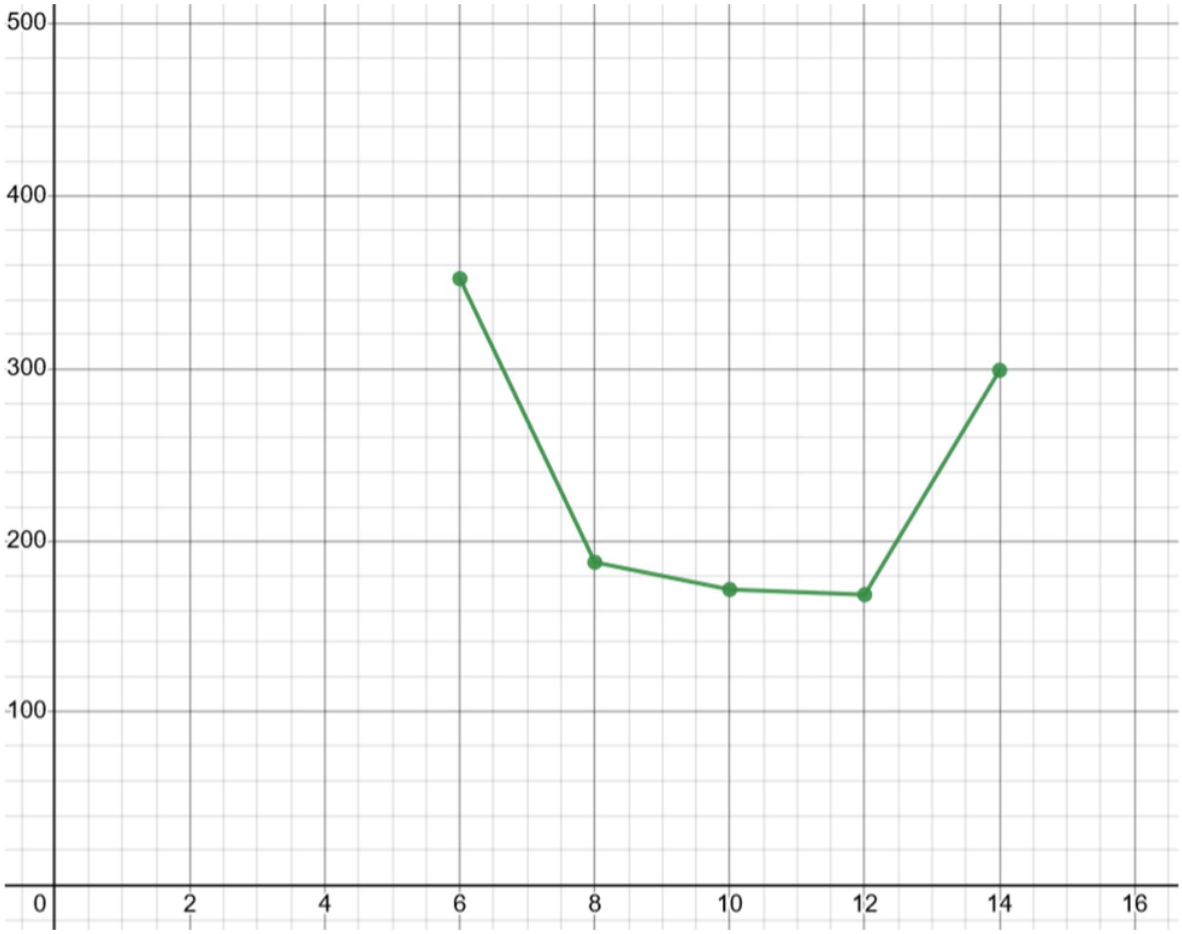

График № 1 – по оси X – кол-во потоков, по оси Y – время в ms при schedule(static)

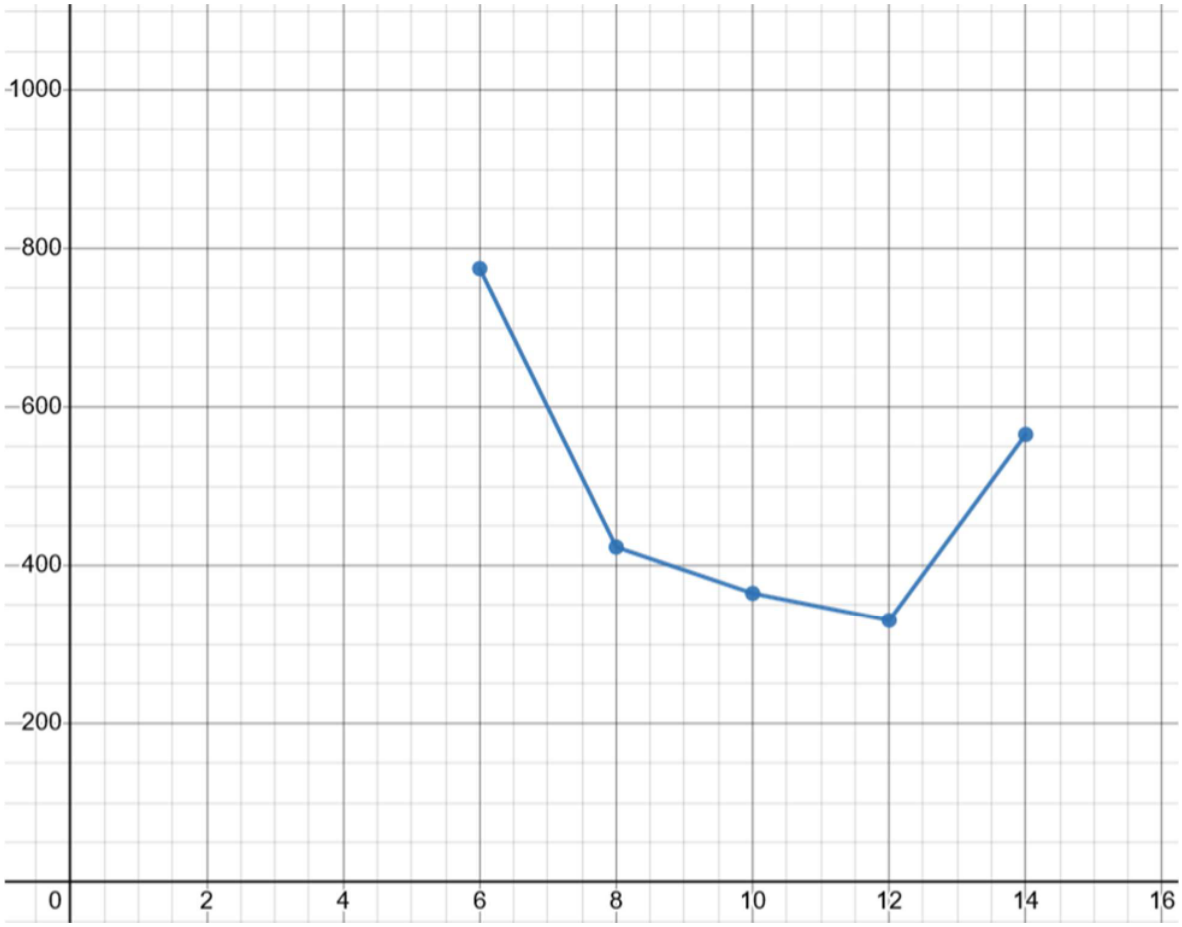

График № 2 – по оси X – кол-во потоков, по оси Y – время в ms при schedule(dynamic)

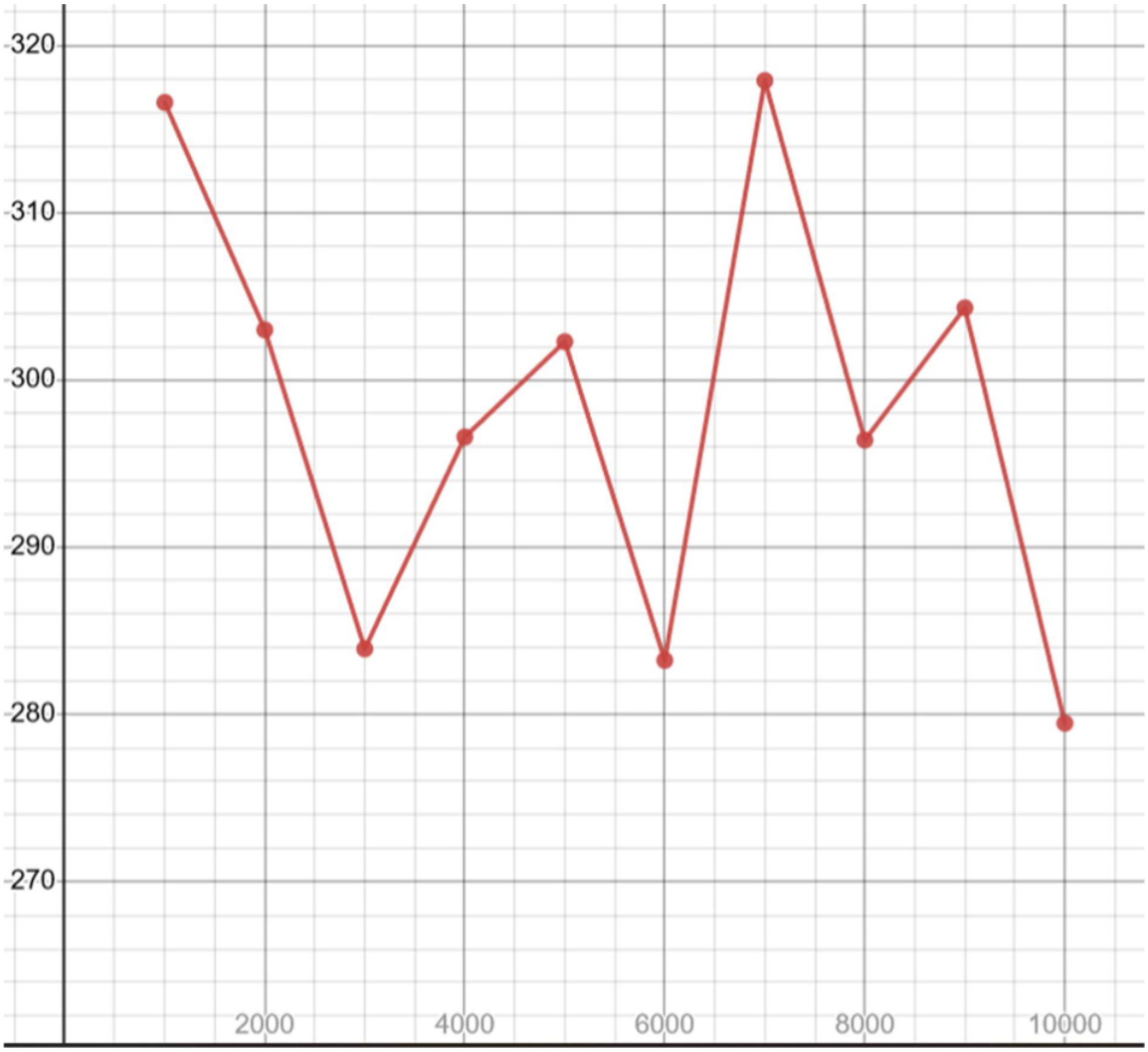

График № 3 – по оси X – chunk_size, по оси Y время в ms, при 12 потоках и schedule(static, chunk_size)

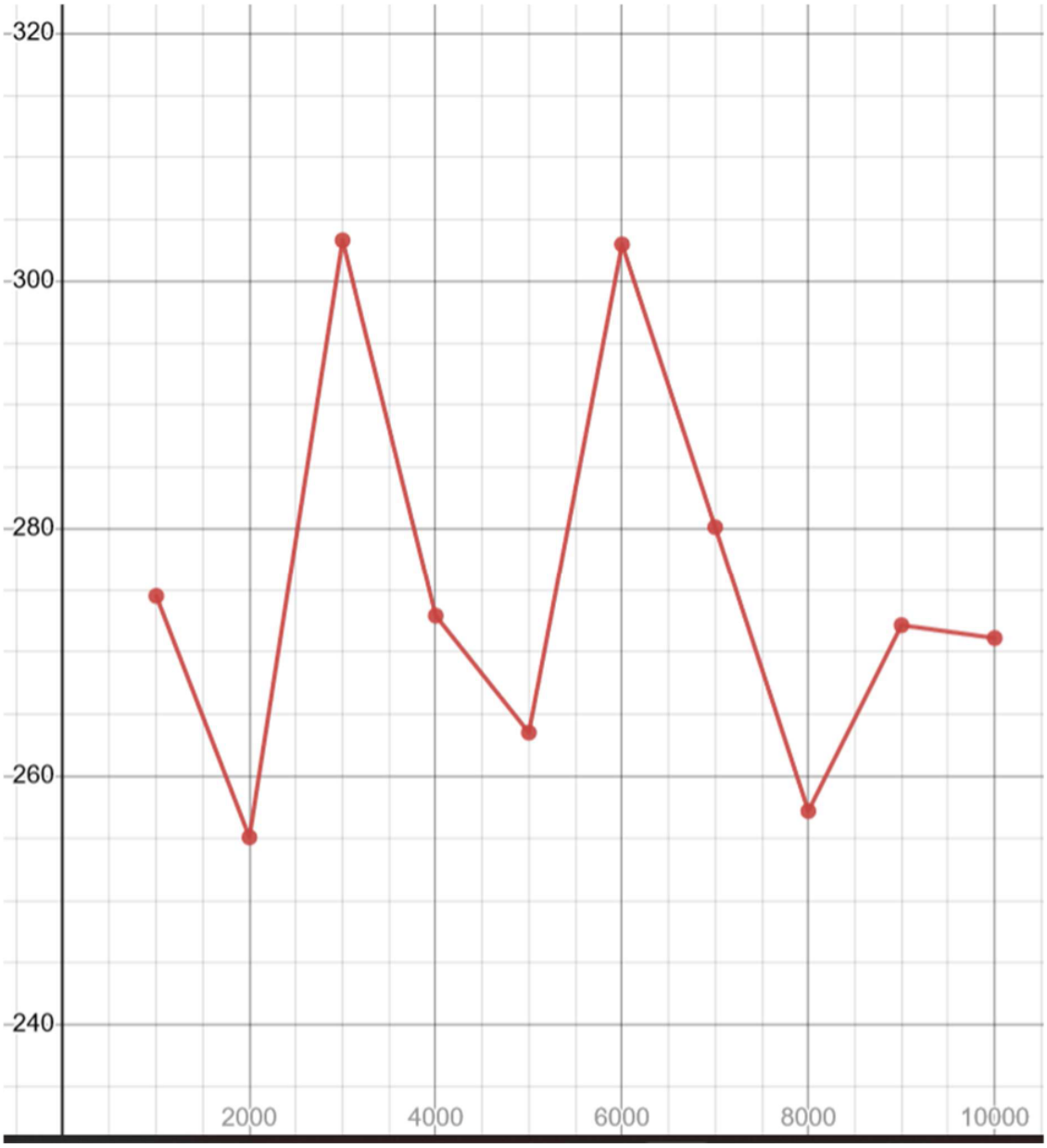

График №4 – по оси X – chunk_size, по оси Y время в ms, при 12 потоках и schedule(dynamic, chunk_size)

Наименьшее время достигается в 169.233 ms при кол-во потоков 12 и schedule(static)

Источники

https://www.youtube.com/watch?v=ziD-oKu3KMI

https://www.openmp.org//wp-content/uploads/cspec20.pdf

Исходный код

easy.cpp

```
#include <fstream>
#include <time.h>
#include <omp.h>
#include<iostream>
#include<string>
using namespace std;
int n;
double r;
bool pointIn(double x, double y) {
    return x * x + y * y <= r * r;
}
double getRand() {
    return (rand() / 32767.0 - 0.5) * 2 * r;
}
int main(int argc, char* argv[]) {
    if (argc < 4) {
        cout << ("not enough arguments, example: " +
            (string)"omp4 <number of threads> <input file name> <output file
name>");
        return 0;
    }
    int threadCnt = stoi(argv[1]);
    threadCnt = 12;
    if (threadCnt < -1 && threadCnt == 0) {
        cout << ("thread count of '" + (string)argv[1] + "' isn't supported");
        return 0;
    }
    else if (threadCnt > 0) {
```

```
            omp_set_num_threads(threadCnt);
        }
        ifstream in(argv[2]);
        if (!in) {
            cout << "Couldn't open file for reading: '" + (string)argv[2] + "'";
            return 0;
        }
        in >> r >> n;
        if (r <= 0 || n <= 0) {
            cout << "n/r isn't supported";
            return 0;
        }
        in.close();
        srand(time(0));
        double start = omp_get_wtime();
        int circ = 0;
    #pragma omp parallel if (threadCnt != -1)
        {
            #pragma omp for schedule(static)
                for (int i = 0; i < n; i++) {
                    if (pointIn(getRand(), getRand())) {
                        #pragma omp atomic
                        circ++;
                    }
                }
        }
        double end = omp_get_wtime();
        printf("Time (%i thread(s)): %g ms\n", threadCnt > 0? threadCnt: 1, (end -
start) * 1000);
```

```
    ofstream out(argv[3]);
    if (!out) {
        cout << ("Couldn't open file for writing: '" + (string)argv[2] + "'");
        return 0;
    }
    out << (4 * r * r * circ) / n;
    out.close();
    return 0;
}
```