МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе № 3.2
по дисциплине «Операционные системы»
Тема: «Процессы и потоки»

Студент гр. 3311 Пасечный Л.В._____

Преподаватель Тимофеев А. В._____

Санкт-Петербург
2025

Введение

Цель работы:

исследовать механизмы создания и управления процессами и потоками в ОС Windows.

Постановка задачи:

Создайте приложение, которое вычисляет число пи с точностью N знаков после запятой по следующей формуле

pi = (4/(1+x0^2) + 4/(1+x1^2) + ... + 4/(1+xN-1^2))*(1/N);

xi = (i + 0.5)*(1/N), i = 0,9999999,

где N=10000000.

Распределите работу по потокам с помощью OpenMP-директивы for.

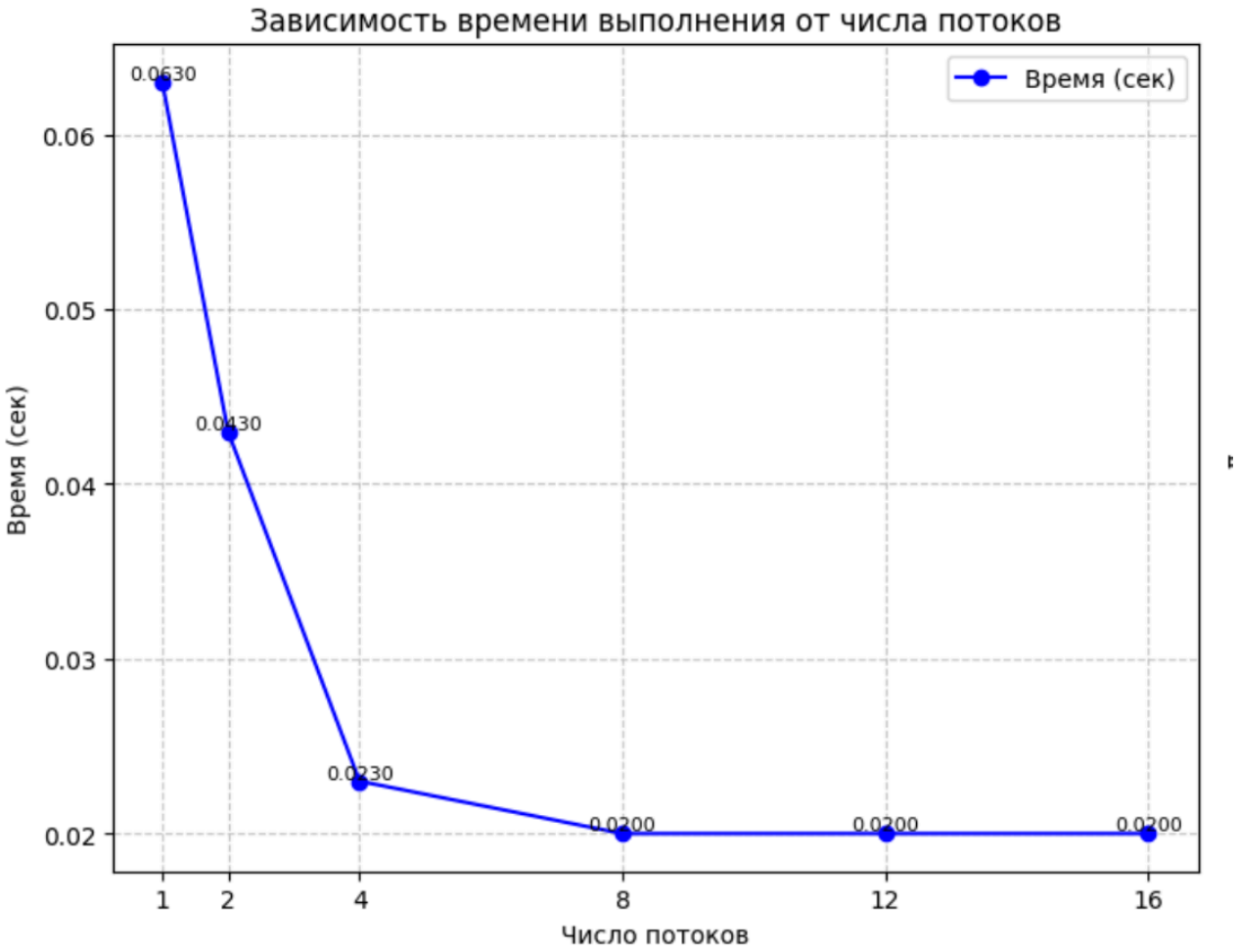
Используйте динамическое планирование блоками итераций (размер блока = 10 * 331118).

2. Произведите замеры времени выполнения приложения для разного числа потоков (1, 2, 4, 8, 12, 16). По результатам измерений постройте график и определите число потоков, при котором достигается наибольшая скорость выполнения. Запротоколируйте результаты в отчет, сравните с результатами прошлой работы.

Результаты:

```
N = 10000000, Chunk size = 3311180
Threads Time (sec)      Pi Value
------  ------  ------
1       0.0180          3.1415926536
2       0.0160          3.1415926536
4       0.0110          3.1415926536
8       0.0070          3.1415926536
12      0.0070          3.1415926536
16      0.0060          3.1415926536
Best perfomance at 16 threads (time = 0.0060000420 sec)
```

График:



Заключение

В работе разработано многопоточное приложение для вычисления числа π с высокой точностью (N = 10,000,000 знаков после запятой) с использованием Win32 API (CreateThread, ResumeThread, SuspendThread). Задача была распараллелена путем динамического распределения блоков итераций (по 33110 итераций) между потоками: каждый поток брал блок, обрабатывал, приостанавливался (SuspendThread) и получал новый свободный блок при возобновлении (ResumeThread).

График показывает, что время выполнения сокращается с 0.063 сек (1 поток) до 0.023 сек (4 потока), но дальнейшее увеличение потоков не даёт эффекта - время стабилизируется на уровне 0.020 сек. Это происходит из-за ограничений процессора и накладных расходов на многопоточность. Оптимально использовать 4 потока - большее количество не ускорит вычисления.

Код программы

```
#include <iostream>
```

```
#include <iomanip>
#include <omp.h>
#include <cmath>

using namespace std;
#define N 10000000
#define CHUNK_SIZE (10*331118)

double calculate_pi(int num_threads){
    double pi = 0.0;
    double sum = 0.0;
    double h = 1.0/N;

    omp_set_num_threads(num_threads);

    #pragma omp parallel for reduction(+:sum) schedule(dynamic, CHUNK_SIZE)
    for (int i = 0; i < N; i++){
        double x = (i+0.5)*h;
        sum += 4.0 / (1.0 + x * x);
    }

    pi = sum * h;

    return pi;
}

int main(){
    int thread_counts[] = {1, 2, 4, 8, 12, 16};
    const int num_tests = sizeof(thread_counts) / sizeof(thread_counts[0]);

    cout << "N = " << N << ", Chunk size = " << CHUNK_SIZE << endl;
    cout << "Threads\tTime (sec)\tPi Value" << endl;
    cout << "------\t------\t------" << endl;

    double reference_pi = 0.0;
    double min_time = 1e9;
    int best_threads = 0;

    for (int t = 0; t < num_tests; t++){
        int num_threads = thread_counts[t];
        double start_time = omp_get_wtime();
        double pi = calculate_pi(num_threads);
        double end_time = omp_get_wtime();

        double elapsed = end_time - start_time;

        if (t == 0) reference_pi = pi;

        if(abs(pi - reference_pi) > 1e-6){
            cerr << "Error: Pi mismatch"<< endl;
            return 1;
        }
```

```
        cout << num_threads << "\t"<< fixed << setprecision(4) << elapsed <<"\t\t" << setprecision(10) << pi
<< endl;

        if (elapsed < min_time){
            min_time = elapsed;
            best_threads = num_threads;
        }
    }

    cout << "Best perfomance at " << best_threads << " threads (time = " << min_time << " sec)" << endl;
    return 0;
}`
```