**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

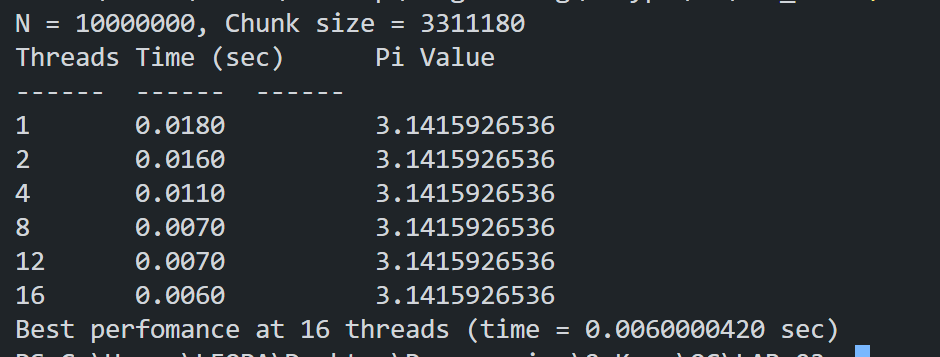
# постройте график и определите число потоков, при котором достигается

# наибольшая скорость выполнения. Запротоколируйте результаты в

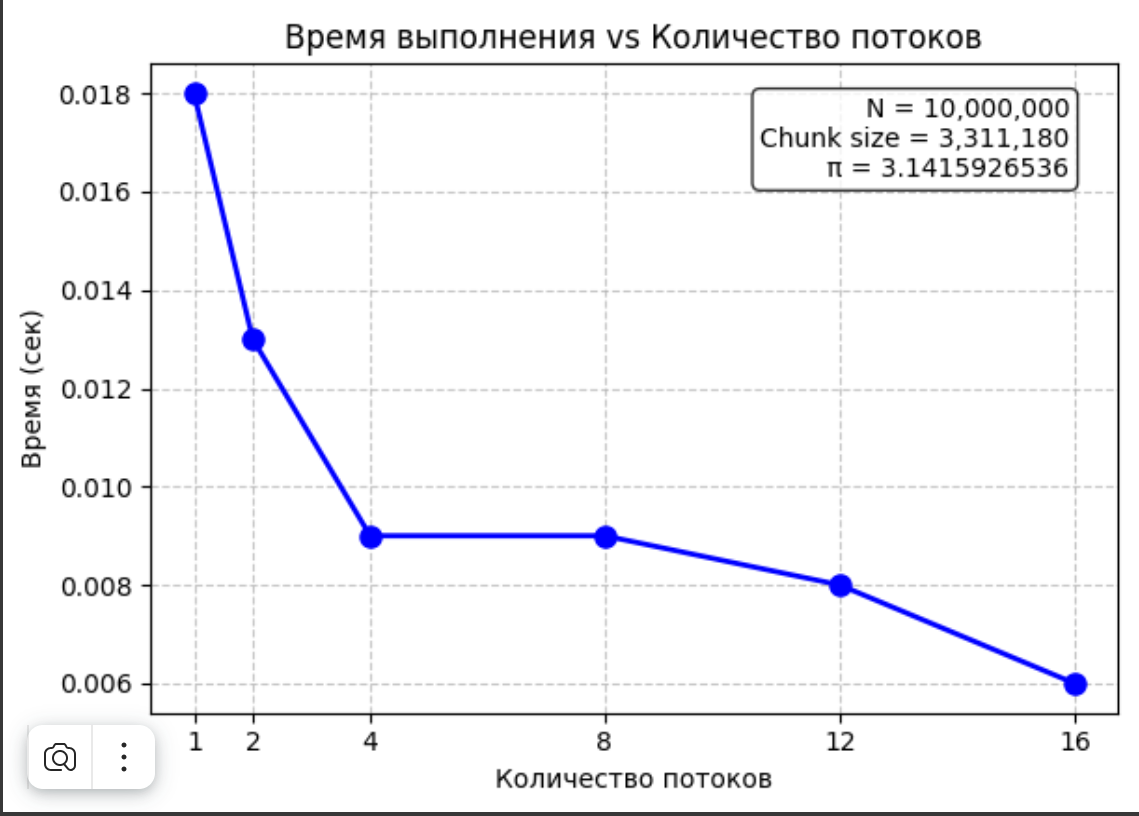
# отчет, сравните с результатами прошлой работы.

# 

# **Результаты:**



**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).

# **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;

}`