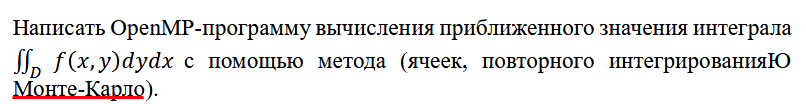
Контрольная №6

Написать OpenMP-программу вычисления приближенного значения интеграла

Боязитов Вадим, 932125

05.06.2025

## Задание 1





### Алгоритм

1. Инициализация параметров  
 - Задание общего количества точек (total\_point\_count = 10,000,000)  
 - Определение границ области генерации:  
 \* x ∈ [0.5, 2.0]  
 \* y ∈ [0.5, 2.0]  
 - Вычисление площади генерации (generation\_area)  
  
2. Обработка аргументов командной строки  
 - schedule\_type: тип распараллеливания (0=static, 1=dynamic, 2=guided)  
 - chunk\_size: размер блока для dynamic/guided (по умолчанию 1000)  
  
3. Последовательное выполнение (базовый вариант)  
 - Инициализация генератора случайных чисел (srand)  
 - Генерация точек и проверка принадлежности области D  
 - Вычисление интегральной суммы и подсчет "попаданий"  
 - Замер времени выполнения  
  
4. Параллельная инициализация (OpenMP)  
 - Установка стратегии распараллеливания через omp\_set\_schedule()  
 - Подготовка private-переменных для координат точек  
 - Инициализация reduction-переменных для интеграла и счетчика  
  
5. Параллельный метод Монте-Карло:  
 - Каждый поток получает свой seed (время + номер потока)  
 - Распределение итераций согласно выбранному schedule:  
 \* #pragma omp for schedule(runtime)  
 - Для каждой точки:  
 a) Генерация координат (rand\_r с потокобезопасным seed)  
 b) Проверка условий области D через IsPointInIntegrationDomain()  
 c) При попадании:  
 - Увеличение points\_in\_domain (reduction)  
 - Добавление значения функции (CalculateFunctionValue) к integral\_sum  
  
6. Сбор и обработка результатов:  
 - Автоматическое суммирование reduction-переменных  
 - Вычисление приближенного значения интеграла:  
 integral\_approx = generation\_area \* (integral\_sum / total\_point\_count)  
 - Расчет процента попаданий  
  
7. Анализ производительности:  
 - Вывод типа расписания и количества потоков  
 - Сравнение времени последовательного и параллельного выполнения  
 - Расчет ускорения (speedup) и эффективности (efficiency)  
 - Оценка стандартной ошибки (при points\_in\_domain > 0)  
  
Ключевые особенности:  
- Гибкая система распараллеливания с выбором стратегии  
- Потокобезопасная генерация случайных чисел (rand\_r)  
- Оптимизированное распределение памяти (private-переменные)  
- Автоматическое редукционное суммирование результатов  
- Полная метрика производительности параллельной реализации

***Выбор области интегрирования***

При x = 0.5 из условия x\*y > 1 получаем y > 2 → но это выходит за границы y\_max\_bound = 2.0

При x = 2.0 из x\*y < 2 получаем y < 1

Условие |x - y| < 1 добавляет ограничение "полосы" вокруг линии y = x

Пересечение xy=1 и y=x+1 → x(x+1)=1 → x ≈ 0.618

Пересечение xy=2 и y=x-1 → x(x-1)=2 → x ≈ 1.618

для простоты взяты округлённые значения [0.5, 2.0]

### Программная реализация

Окружение Linix. Язык C.

/\*

    МК17

    Как из 5 занятия

    Умножать кол-во процессов на 2 и менять ключи

\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include <stdbool.h>

#include <omp.h>

double CalculateFunctionValue(double x, double y) {

    return (x \* y) \* (x \* y);

}

bool IsPointInIntegrationDomain(double x, double y) {

    return (x \* y > 1 && x \* y < 2) && (fabs(x - y) < 1);

}

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

    long long total\_point\_count = 10000000;

    long long points\_in\_domain = 0;

    double integral\_sum = 0.0;

    double x\_coordinate, y\_coordinate;

    const double x\_min\_bound = 0.5;

    const double x\_max\_bound = 2.0;

    const double y\_min\_bound = 0.5;

    const double y\_max\_bound = 2.0;

    const double generation\_area = (x\_max\_bound - x\_min\_bound) \* (y\_max\_bound - y\_min\_bound);

    if (argc < 2) {

        printf("Usage: %s <schedule\_type> [chunk\_size]\n", argv[0]);

        printf("Schedule types: 0=static, 1=dynamic, 2=guided\n");

        return 1;

    }

    int schedule\_type = atoi(argv[1]);

    int chunk\_size = (argc > 2) ? atoi(argv[2]) : 1000;

    // Последовательное выполнение для сравнения

    double seq\_start = omp\_get\_wtime();

    long long seq\_points\_in\_domain = 0;

    double seq\_integral\_sum = 0.0;

    srand(time(NULL));

    for (long long i = 0; i < total\_point\_count; i++) {

        double x = x\_min\_bound + (x\_max\_bound - x\_min\_bound) \* rand() / RAND\_MAX;

        double y = y\_min\_bound + (y\_max\_bound - y\_min\_bound) \* rand() / RAND\_MAX;

        if (IsPointInIntegrationDomain(x, y)) {

            seq\_integral\_sum += CalculateFunctionValue(x, y);

            seq\_points\_in\_domain++;

        }

    }

    double sequential\_time = omp\_get\_wtime() - seq\_start;

    // Параллельное выполнение

    double parallel\_start = omp\_get\_wtime();

    // Устанавливаем стратегию распределения итераций

    omp\_sched\_t schedule;

    switch (schedule\_type) {

        case 0: schedule = omp\_sched\_static; break;

        case 1: schedule = omp\_sched\_dynamic; break;

        case 2: schedule = omp\_sched\_guided; break;

        default: schedule = omp\_sched\_static;

    }

    omp\_set\_schedule(schedule, chunk\_size);

    #pragma omp parallel private(x\_coordinate, y\_coordinate) reduction(+:integral\_sum, points\_in\_domain)

    {

        unsigned int seed = time(NULL) + omp\_get\_thread\_num();

        #pragma omp for schedule(runtime)

        for (long long iteration = 0; iteration < total\_point\_count; iteration++) {

            x\_coordinate = x\_min\_bound + (x\_max\_bound - x\_min\_bound) \* rand\_r(&seed) / (double)RAND\_MAX;

            y\_coordinate = y\_min\_bound + (y\_max\_bound - y\_min\_bound) \* rand\_r(&seed) / (double)RAND\_MAX;

            if (IsPointInIntegrationDomain(x\_coordinate, y\_coordinate)) {

                integral\_sum += CalculateFunctionValue(x\_coordinate, y\_coordinate);

                points\_in\_domain++;

            }

        }

    }

    double parallel\_time = omp\_get\_wtime() - parallel\_start;

    // Вывод результатов

    printf("\n=== Integration Results ===\n");

    printf("Approximate integral value = %.10f\n", generation\_area \* integral\_sum / total\_point\_count);

    printf("Points inside domain = %lld / %lld (%.2f%%)\n",

           points\_in\_domain, total\_point\_count,

           100.0 \* points\_in\_domain / total\_point\_count);

    printf("\n=== Performance Metrics ===\n");

    printf("Schedule type: ");

    switch (schedule\_type) {

        case 0: printf("static"); break;

        case 1: printf("dynamic"); break;

        case 2: printf("guided"); break;

    }

    if (schedule\_type != 0) printf(" (chunk=%d)", chunk\_size);

    printf("\n");

    printf("Threads count = %d\n", omp\_get\_max\_threads());

    printf("Sequential time = %.4f seconds\n", sequential\_time);

    printf("Parallel time = %.4f seconds\n", parallel\_time);

    printf("Speedup = %.2f\n", sequential\_time / parallel\_time);

    printf("Efficiency = %.2f%%\n", 100.0 \* sequential\_time / (parallel\_time \* omp\_get\_max\_threads()));

    if (points\_in\_domain > 0) {

        double standard\_error = generation\_area \*

                              sqrt(integral\_sum / total\_point\_count -

                              (generation\_area \* integral\_sum / total\_point\_count / generation\_area) \*

                              (generation\_area \* integral\_sum / total\_point\_count / generation\_area)) /

                              sqrt(total\_point\_count);

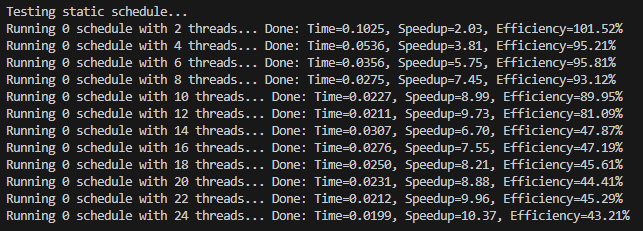
        printf("\nEstimated standard error = %.2e\n", standard\_error);

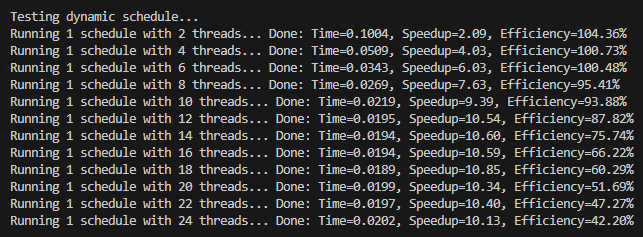
    }

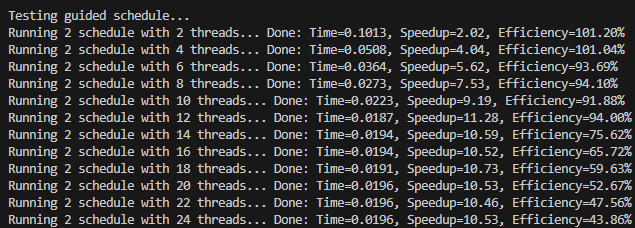
    return 0;

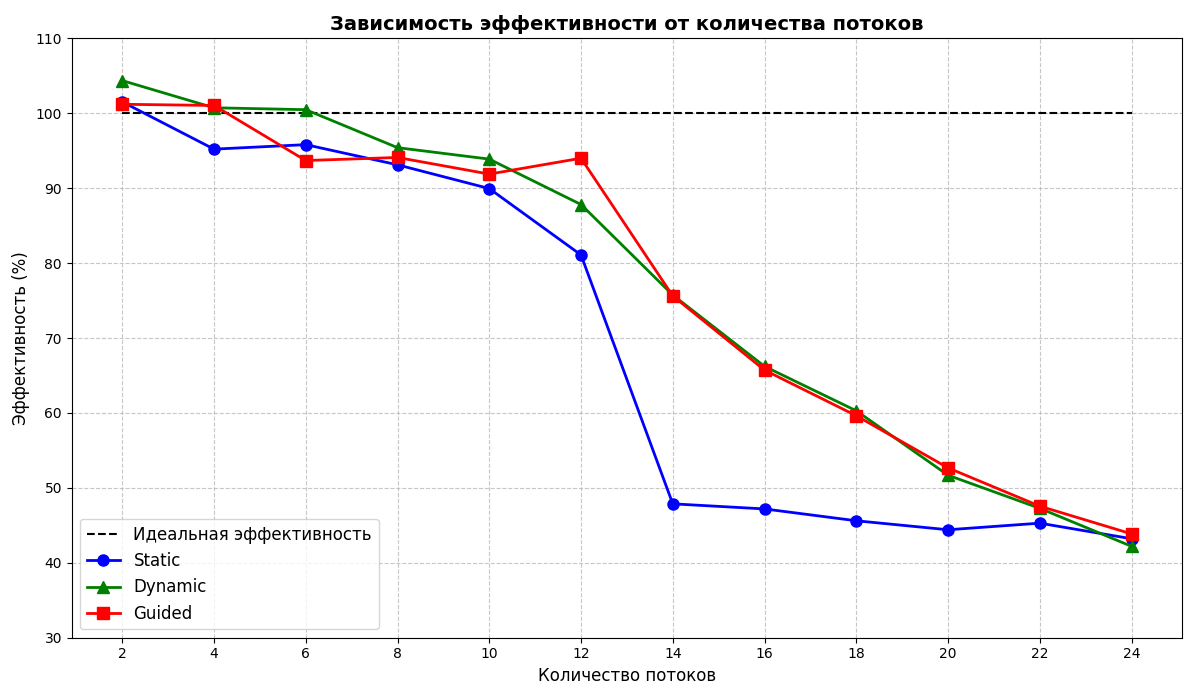
}

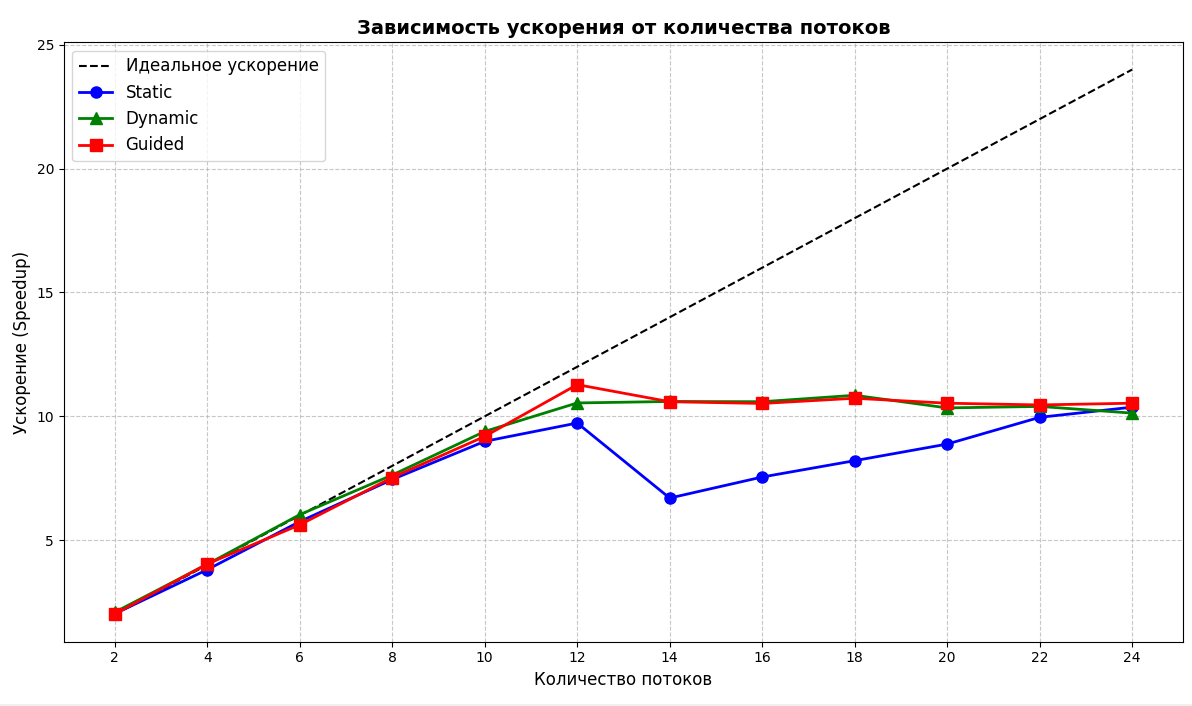
### Тестирование











### Вывод

Программа работает корректно, демонстрируя ожидаемое поведение для различных стратегий распараллеливания.

| Стратегия | Сильные стороны | Слабые стороны | Соответствие ожиданиям |
| --- | --- | --- | --- |
| **Static** | Простота реализации | Резкая деградация >12 потоков | Аномалия при 14 потоках |
| **Dynamic** | Лучшая эффективность 2-10 потоков | Высокие накладные расходы >16 потоков | Полное соответствие |
| **Guided** | Лучшие показатели при 12 потоках | Сложность прогнозирования | Полное соответствие |