Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ»

Институт системной и программной инженерии

и информационных технологий (СПИНТех)

**Отчёт**

по дисциплине «Технологии параллельного программирования»

**Лабораторная работа №1**

**Вариант-7**

Руководитель

\_\_\_\_\_\_\_\_\_\_ Можжухина A. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Студент группы ПИН-23

\_\_\_\_\_\_\_\_\_\_ Исламов Р. Р.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

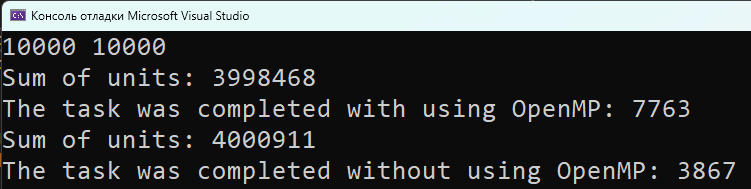
*Москва*

*2023*

# 1 Листинг

/\*Задание 1. Написать на C++ программу в соответствии с вариантом задания, при этом программы должны оптимально использовать предлагающиеся вычислительные ресурсы.Для реализации многопоточности использовать технологию OpenMP.Измерять время работы программы для различных значений параметров.Результаты занести в отчёт.Также, при запуске программ, запустить диспетчер задач Windows, в нём установить счётчик потоков, и просмотреть сколько потоков выполнения использует процесс, соответствующий запущенной программе.Результаты также занести в отчёт.Вариант-7 Исламов1 Дана матрица из MxN натуральных(ненулевых) элементов(задаются случайно).Написать программу, считающую количество семёрок в десятеричной записи числа всех попарных сумм элементов для каждой строки.\*/#include <iostream>#include <ctime>#include <omp.h>using namespace std;int arr1[10000][10000]; // двумерный массивint arr2[10000][10000]; // двумерный массивint i, j, N, M; // счётчики и размеры матрицыint outR1; // Счётчик выводаint outR2; // Счётчик выводаint main(){ unsigned int start\_time = clock(); // начальное время srand(time(0)); // автоматическая рандомизация cin >> N; cin >> M; // вводим размеры NxM { for (i = 0; i < N; i++) { for (j = 0; j < M; j++) { arr1[i][j] = rand() % 10; //запись в матрицу случайных чисел от 1 до 9 } } //for (i = 0; i < n; i++) { // for (j = 0; j < m; j++) { // cout << arr[i][j] << " "; //вывод элементов // } // cout << endl; //} #pragma omp parallel shared(a) private(i) { #pragma omp for private(i,arr) { for (i = 0; i < N; i += 1) { for (j = 0; j < M; j += 2) { if (arr1[i][j] + arr1[i][j + 1] == 7) outR1 += 1; } } } } cout << "Сумма единиц: "<< outR1; //вывод счётчика cout << endl; unsigned int end\_time = clock(); // конечное время unsigned int search\_time = end\_time - start\_time; // искомое время cout << "Задача выполнена с использованием OpenMP: " << search\_time; cout << endl; } { for (i = 0; i < N; i++) { for (j = 0; j < M; j++) { arr2[i][j] = rand() % 10; //запись в матрицу случайных чисел от 1 до 9 } } for (i = 0; i < N; i += 1) { for (j = 0; j < M; j += 2) { if (arr2[i][j] + arr2[i][j + 1] == 7) outR2 += 1; } } cout << "Сумма единиц: "<< outR2; //вывод счётчика cout << endl; unsigned int end\_time = clock(); // конечное время unsigned int search\_time = end\_time - start\_time; // искомое время cout << "Задача выполнена без использования OpenMP: " << search\_time; }}

# 2 Результат работы



# 3 Контрольные вопросы

1. Проблемы, которые могут возникнуть при использовании общих данных в параллельно выполняемых потоках в параллельном программировании, включают в себя:

- Гонки данных (Data Races): Когда несколько потоков одновременно обращаются к общей памяти без синхронизации, это может привести к непредсказуемым результатам и ошибкам в программе.

- Проблемы согласованности данных (Data Consistency Issues): Несинхронизированные доступы к общим данным могут привести к некорректному чтению и записи, что может нарушить целостность данных.

- Взаимное исключение (Mutexes, Locks): Использование мьютексов (механизмов взаимного исключения) для защиты общих данных от одновременного доступа может вызвать проблемы с производительностью и дедлоками.

2. Директивы OpenMP обычно записываются в виде комментариев с специфическими префиксами. Например, директива для создания параллельной области выглядит так:

```c

#pragma omp parallel

```

Здесь `#pragma` - это специфическая директива компилятора, а `omp parallel` - это инструкция OpenMP для создания параллельной секции.

3. Директива `#pragma omp parallel` используется для создания параллельной секции в программе. Она указывает компилятору на необходимость выполнения блока кода в нескольких потоках. Когда эта директива выполняется, каждый поток выполняет свой набор инструкций внутри параллельной области.

4. В контексте параллельных программ:

- \*\*Фрагмент\*\* - это часть кода программы, которая может быть выполнена параллельно несколькими потоками. Фрагмент может быть представлен, например, как блок кода внутри параллельной секции.

- \*\*Область\*\* - это часть программы, в которой определены параллельные секции или другие конструкции OpenMP. Область определяет, где и как выполняется параллельный код.

- \*\*Секция\*\* - это конкретная часть программы, в которой выполняется код в параллельном режиме. Секции могут быть определены с использованием директив OpenMP, таких как `#pragma omp parallel` или `#pragma omp sections`.

5. Распараллеливание циклов в OpenMP обычно осуществляется с использованием директивы `#pragma omp for`. Пример:

```c

#pragma omp parallel for

for (int i = 0; i < n; i++) {

// код, выполняемый в параллельном режиме

}

```

Чтобы циклы могли быть распараллелены, они должны соответствовать определенным условиям, таким как:

- Цикл должен иметь достаточно большое количество итераций, чтобы оправдать распараллеливание, так как создание и управление потоками имеет свои накладные расходы.

- Итерации цикла должны быть независимыми, то есть результат выполнения одной итерации не должен зависеть от результатов других итераций.

6. В OpenMP можно использовать различные методы для управления распределением итераций циклов между потоками. Например:

- `schedule` clause: Директива `#pragma omp for` может иметь опциональный параметр `schedule`, который позволяет указать, как распределить итерации между потоками, например, равномерно (`static`) или динамически (`dynamic`).

- `collapse` clause: Директива `#pragma omp for` может использовать `collapse`, чтобы объединить несколько вложенных циклов в один и распараллелить их как единое целое.

7. Порядок выполнения итераций в распараллеливаемых циклах обычно не определен и зависит от распределения итераций между потоками. Однако в OpenMP можно использовать различные методы для управления порядком выполнения, такие как:

- `ordered` clause: Этот клоз позволяет соблюдать порядок выполнения итераций, даже если они распределены между потоками. Вы можете использовать `#pragma omp ordered` внутри цикла, чтобы гарантировать последовательное выполнение итераций.

- `nowait` clause: Ключевое слово `nowait` может быть использовано в директиве `#pragma omp for`, чтобы позволить потокам продолжать выполнение после завершения своих итераций, не ожидая другие потоки.

Важно отметить, что контроль порядка выполнения может повлиять на производительность и потребление ресурсов, поэтому следует использовать его осторожно в зависимости от требований вашей программы.