Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №4

Название: Параллельное программирование

Дисциплина: Анализ алгоритмов

Студент ИУ7-54Б _____ Л.Е.Тартыков

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Содержание

Введение

При выполнении множества задач необходимо обеспечить такую скорость вычислений, чтобы пользователь не подумал, что программа зависает. Для этого необходимо увеличивать скорость выполнения программ. В настоящее время по определенным техническим причинам стало невозможным увеличивать тактовую частоту процессора. Однако есть другой способ увеличения производительности – размещение нескольких ядер в процессоре, но это требует другого подхода в программировании.

Параллельное программирование - новый подход в технологии разработки программного обеспечения, которое основывается на понятии "поток". Поток - часть кода программы, которая может выполняться параллельно с другими частями кода программы. Многопоточность - способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько потоков.

Целью лабораторной работы является изучение и реализация параллельного программирования для решения поиска минимального элемента в матрице. Для её достижения необходимо выполнить следующие задачи:

- исследовать подходы паралелльного программирования;
- привести схемы алгоритмов последовательного и паралелльного поиска минимального элемента матрицы;
- описать используемые структуры данных;
- выполнить тестирование реализации алгоритмов методом черного ящика;
- провести сравнительный анализ этих алгоритмов по процессорному выполнению времени на основе экспериментальных данных.

1 Аналитический раздел

В данном разделе рассматривается описание общей задачи, принцип распределения задач.

1.1 Определение матрицы

Матрицей размера $m \times n$ называется прямоугольная таблица элементов некоторого множества (например, чисел или функций), имеющая m строк и n столбцов [**?**]. Элементы a_{ij} , из которых составлена матрица, называются элементами матрицы. Условимся, что первый индекс i элемента a_{ij} а соответствует номеру строки, второй индекс j – номеру столбца, в котором расположен элемент a_{ij}. Матрица может быть записана по формуле (1.1).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \tag{1.1}$$

1.2 Алгоритм поиска минимума в матрице.

Для решения нахождения поиска минимума в матрице необходимо выполнить следующие действия: для каждой строки матрицы $A[i]$ нужно выполнить вычисления локального максимума по формуле (1.2).

$$loc_{min} = min(a_{i0}, ..., a_{in}) \tag{1.2}$$

Затем полученные каждые локальные минимумы сравниваются друг с другом. Таким образом, будет выполнен поиск глобального минимума в матрице.

1.3 Параллельная реализация алгоритма

В каждой строке матрицы выполняются схожие вычисления по нахождению в ней минимального элемента. Эти действия являются независимыми, причем строка матрицы не изменяется, поэтому для параллельного вычисления целесообразно разделить данную задачу между потоками. Каждый поток будет выполняться с необходимым объемом данных.

1.4 Вывод

В данном разделе были описаны необходимая задача и принцип её разделения. На вход программному обеспечению подается матрица; на выход программа должна выдать результат - значение максимального элемента в матрице. Все данные, подаваемые на вход, являются корректными.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритма поиска минимального элемента в матрице последовательным способом и с помощью параллельной реализации. Было выполнено описание способов тестирование и выделенных классов эквивалентностей. Также представлены выбранные типы и структуры данных.

2.1 Выбранные типы и структуры данных

В данной работе используются следующие типы и структуры данных:

- matrix_args_t - структура, содержащая матрицу, количество строк и столбцов в ней;
- pthread_args_t - структура, содержащая информацию о номере потока, количестве потоков, количестве строк матрицы, локальный минимум и структуру matrix_args_t.

2.2 Схемы алгоритмов

Ниже представлены следующие схемы алгоритмов:

- рисунок 2.1 - схема алгоритма поиска минимума в матрице (последовательная реализация);

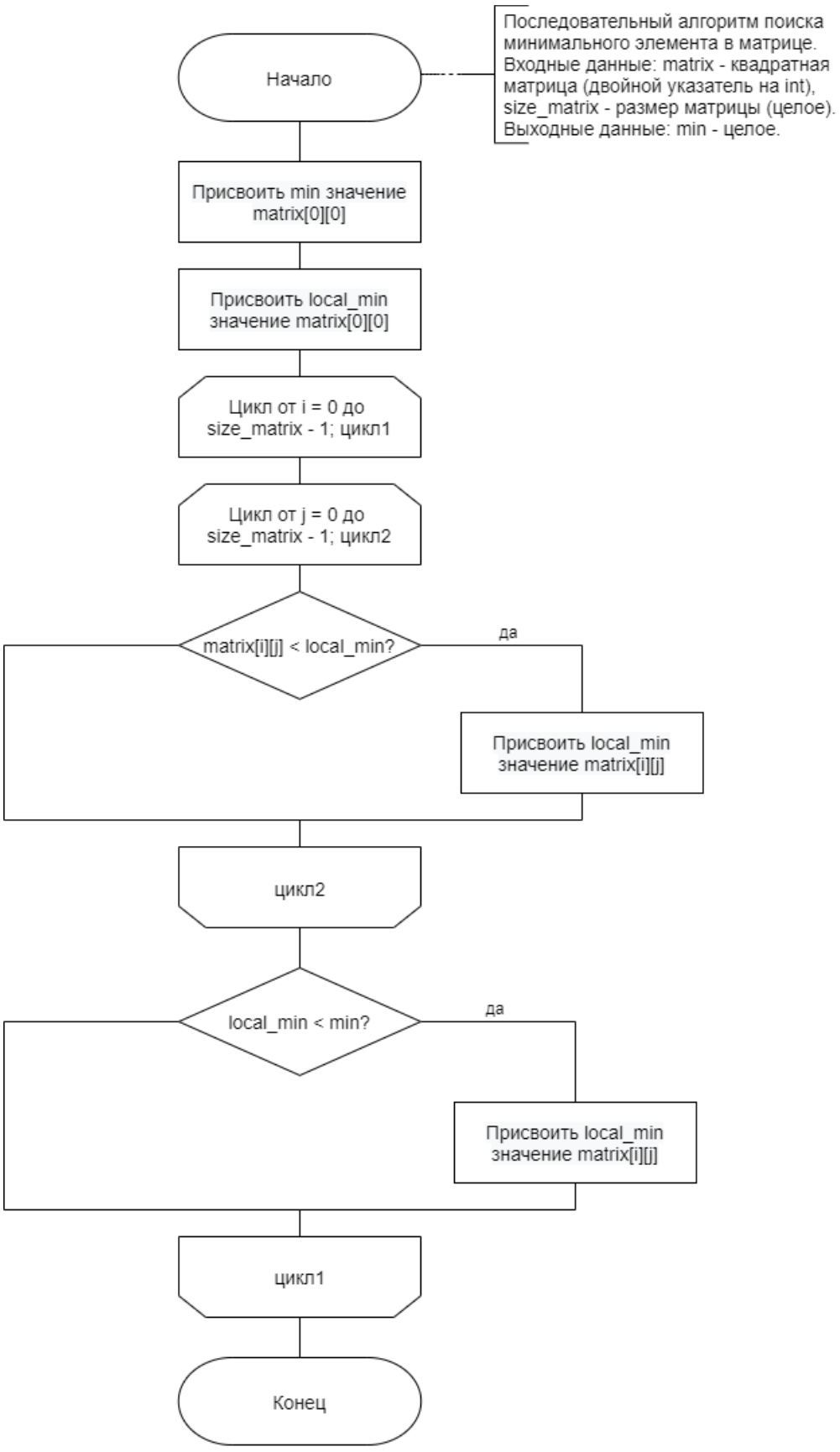

Рисунок 2.1 – Схема алгоритма поиска минимума в матрице (последовательная реализация).

- рисунок 2.2 - схема алгоритма поиска минимума в матрице (параллельная реализация);

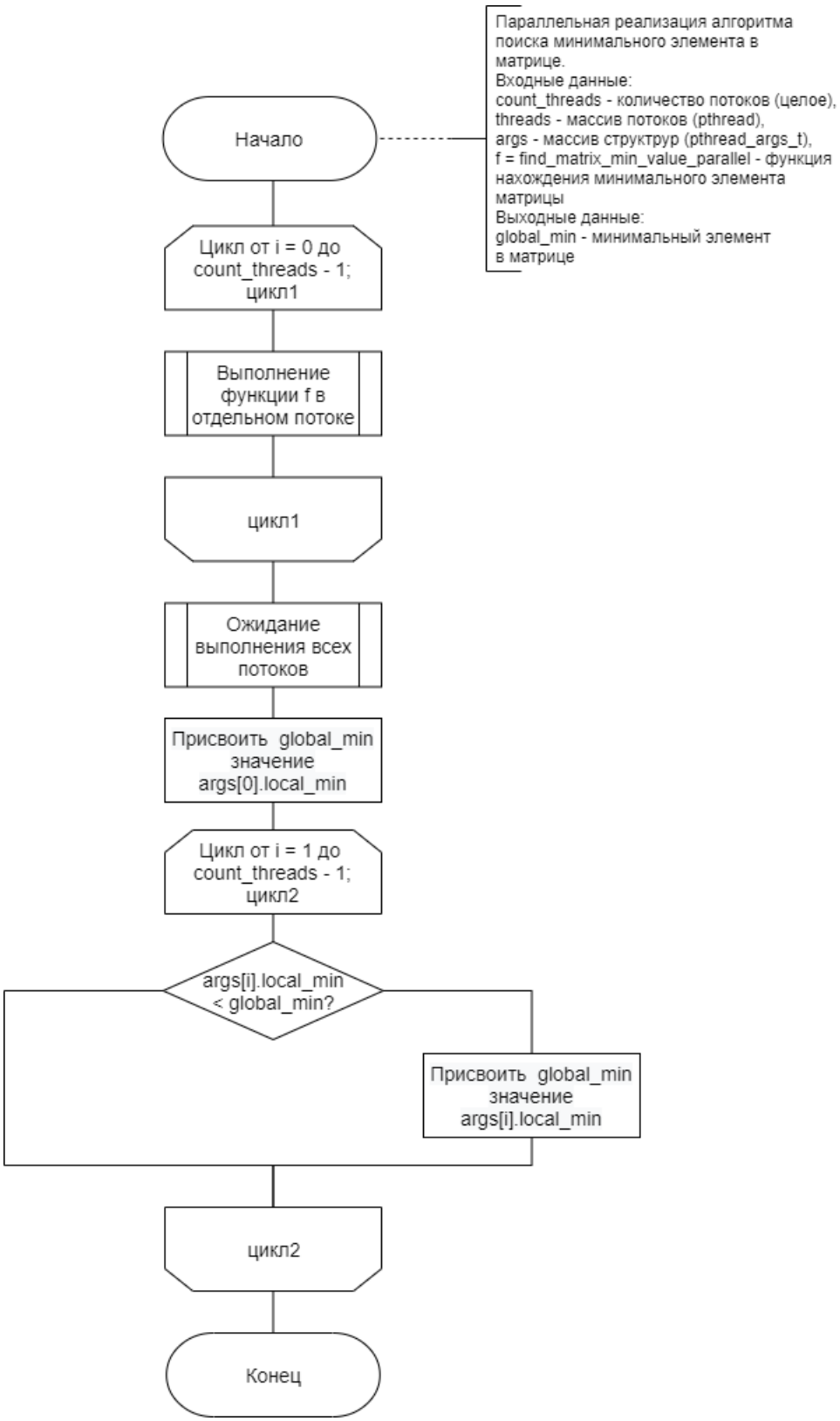

Рисунок 2.2 – Схема алгоритма поиска минимума в матрице (параллельная реализация).

- рисунок 2.3 - распределение задач поиска минимума в матрице (параллельная реализация);

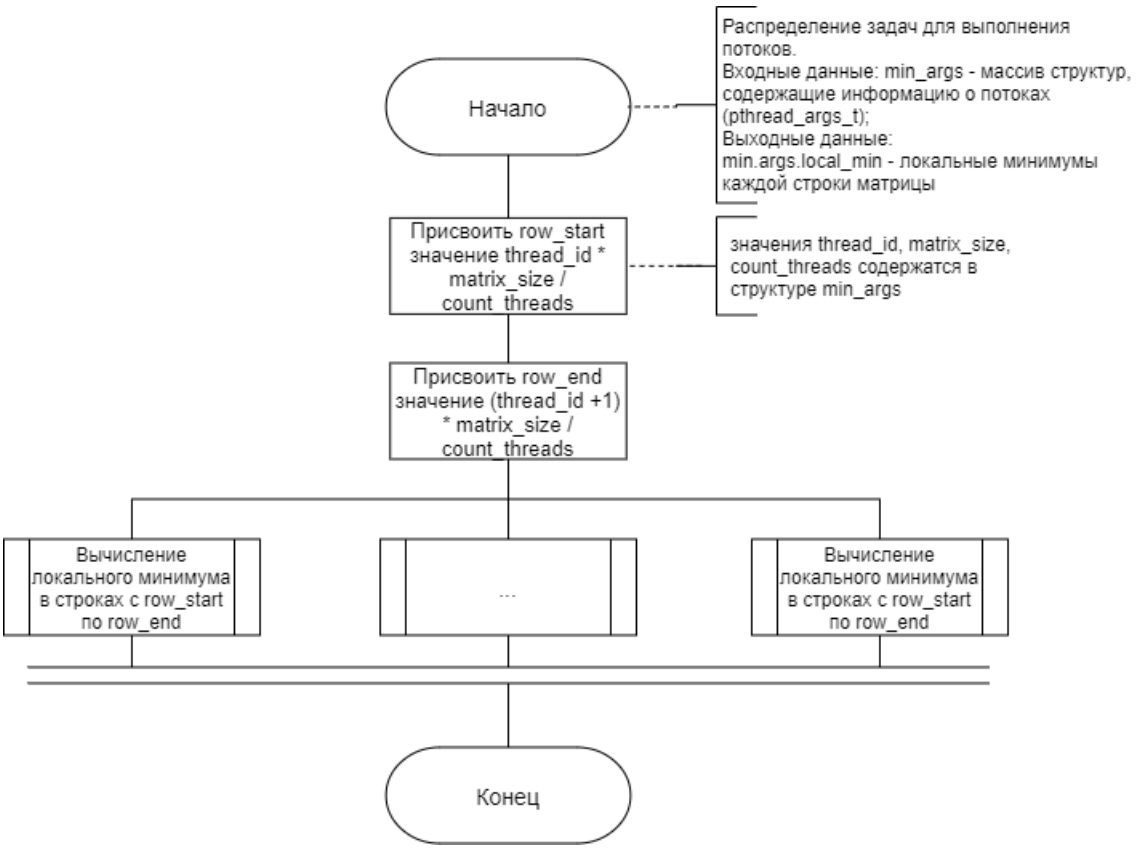

Рисунок 2.3 – Распределение задач поиска минимума в матрице (параллельная реализация).

2.3 Вывод

На основе теоретических данных, полученных в аналитическом разделе, были построены схемы необходимых реализаций алгоритма, представлены необходимые типы и структуры данных.

3 Технологический раздел

В данном разделе приведены средства реализации и листинги кода.

3.1 Выбор средств реализации

Для реализации алгоритмов в данной лабораторной работе был выбран язык программирования C++ [**?**]. Данный выбор обусловен тем, что он содержит нативный код, что необходимо для уменьшения времени выполнения алгоритмов с помощью распараллеливания. Выбранным механизмом работы с потоками предоставляет библиотека pthread. POSIX определяет набор интерфейсов для программирования потоков. В качестве среды разработки был использован Visual Studio Code[**?**], так как в нем присутсвует поддержка практически всех языков программирования. Выполнение замера процессорного времени осуществлено при помощи библиотеки chrono [**?**].

3.2 Листинги программ

Ниже представлены листинги разработанных алгоритмов умножения матриц.

Листинг 3.1 – Программный код алгоритма нахождения минимума в матрице (последовательная реализация).

```
int find_matrix_min_value(int **matrix, int size_matrix)
{
    int min = matrix[0][0], local_min = matrix[0][0];
    for (int i = 0; i < size_matrix; i++)
    {
        for (int j = 0; j < size_matrix; j++)
        {
            if (matrix[i][j] < local_min)
            local_min = matrix[i][j];
        }
        if (local_min < min)
        min = local_min;
    }
    return min;
}
```

Листинг 3.2 – Программный код алгоритма нахождения минимума в матрице (параллельная реализация).

```
void *find_matrix_min_value_parallel(void *args)
{
    pthread_args_t *mult_args = (pthread_args_t *)args;
    int row_start = mult_args->thread_id * (mult_args->matrix_size /
        mult_args->count_threads);
    int row_end = (mult_args->thread_id + 1)* (mult_args->
       matrix_size / mult_args->count_threads);
    int **matrix = mult_args->args->matrix;
    mult_args->local_min = matrix[0][0];

    for (int i = row_start; i < row_end; i++)
    {
        for (int j = 0; j < mult_args->matrix_size; j++)
        {
            if (matrix[i][j] < mult_args->local_min)
            mult_args->local_min = matrix[i][j];
        }
    }

    return NULL;
}
```

3.3 Вспомогательные модули

На листингах представлены программные модули, которые используются в данных функциях:

Листинг 3.3 – Программный код класса по работе с матрицами.

```
    for (int k = 0; k < count_threads; k++){
        pthread_create(threads + k, NULL, find_matrix_min_value_parallel
            , args + k);
    }

    for (int k = 0; k < count_threads; k++){
        pthread_join(threads[k], NULL);
    }
    int global_min = args[0].local_min;
    for (int i = 1; i < count_threads; i++){
        if (args[i].local_min < global_min)
        global_min = args[i].local_min;
    }
```

3.4 Тестирование

Для тестирования используется метод черного ящика. В данном разделе приведена таблица 3.1, в которой указаны классы эквивалентностей тестов:

Таблица 3.1 – Таблица тестов

№	Описание теста	Матрица	Ожидаемый результат
1	Квадратная матрица	$\begin{pmatrix} 6 & 9 & 8 \\ 0 & 3 & 6 \\ 4 & 9 & 5 \end{pmatrix}$	0
2	Прямоугольная матрица	$\begin{pmatrix} 2 & 3 & -2 \\ 3 & 9 & 2 \end{pmatrix}$	-2
3	Несколько минимумов	$\begin{pmatrix} 5 & 1 \\ 1 & 4 \\ 6 & 4 \end{pmatrix}$	1

Для последовательной и параллельной реализации алгоритма поиска

минимума в матрице все тесты пройдены успешно.

3.5 Вывод

В данном разделе был выбран язык программирования, среда разработки, описан механизм работы с потоками. Выполнена реализация последовательного и параллельного алгоритма. Было успешно проведено их тестирование методом черного ящика по таблице 3.1.