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»

Отчет по лабораторной работе №4 по курсу "Анализ алгоритмов"

Тема Многопоточное программирование

Студент Чепиго Д.С.

Группа ИУ7-54Б

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2022 г.

Содержание

Введение

Одной из задач программирования является ускорение решения вычислительных задач. Один из способов ее решения – использование параллельных вычислений.

В последовательном алгоритме решения какой-либо задачи есть операции, которые может выполнять только один процесс, например, операции ввода и вывода. Кроме того, в алгоритме могут быть операции, которые могут выполняться параллельно разными процессами. Способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой, называют многопоточностью [1].

Для распараллеливания может быть рассмотрена задача поиска кратчайших путей между всеми парами вершин графа [2]. Данная задача решается при помощи алгоритма Флойда. Этот алгоритм был одновременно опубликован в статьях Роберта Флойда (Robert Floyd) и Стивена Уоршелла (Stephen Warshall) в 1962 году, хотя в 1959 году Бернард Рой (Bernard Roy) опубликовал практически такой же алгоритм, но это осталось незамеченным. Часто этот алгоритм называют алгоритмом Флойла-Уоршелла или алгоритмом Роя-Флойда. Далее будем использовать название алгоритм Флойда.

Цель лабораторной работы – получить навык организации параллельных вычислений на базе нативных потоков.

Задачи лабораторной работы:

— проанализировать последовательный и параллельный варианты алгоритма поиска кратчайших расстояний между всеми парами вершин графа;

— определить средства программной реализации выбранного алгоритма;

— реализовать разработанный алгоритм;

— провести сравнительный анализ по времени реализованного алгоритма;

— подготовить отчет о лабораторной работе.

1 Аналитическая часть

1.1 Основы теории графов

Многие объекты, возникающие в жизни человека, могут быть смоделированы при помощи графов. Например, транспортные схемы изображают в виде станций, соединенных линиями. В терминах графов станции называются вершинами графа, а линии – ребра.

Графом называется конечное множество вершин и множество ребер [3]. Каждому ребру сопоставлены две вершины – концы ребра. Число вершин графа называют порядком. Путем на графе называется последовательность ребер, в которой конец одного ребра является началом следующего ребра. Начало первого ребра называется началом пути, конец последнего ребра – концом пути.

Бывают различные варианты определения графа. В данном определении концы у каждого ребра – равноправны. В этом случае нет разницы где начало, а где – конец у ребра. Но, например, в транспортных сетях бывают случаи одностороннего движения по ребру, тогда говорят об ориентированном графе – графе, у ребер которого одна вершина считается начальной, а другая – конечной.

Очень часто рассматриваются графы, в которых каждому ребру приписана некоторая числовая характеристика – вес. Вес может означать длину дороги или стоимость проезда по данному маршруту. Такие графы называются взвешенными.

Способы представления графов

Существует два способа представления графа. Рассмотрим их далее для следующего графа представленного на 1.1.

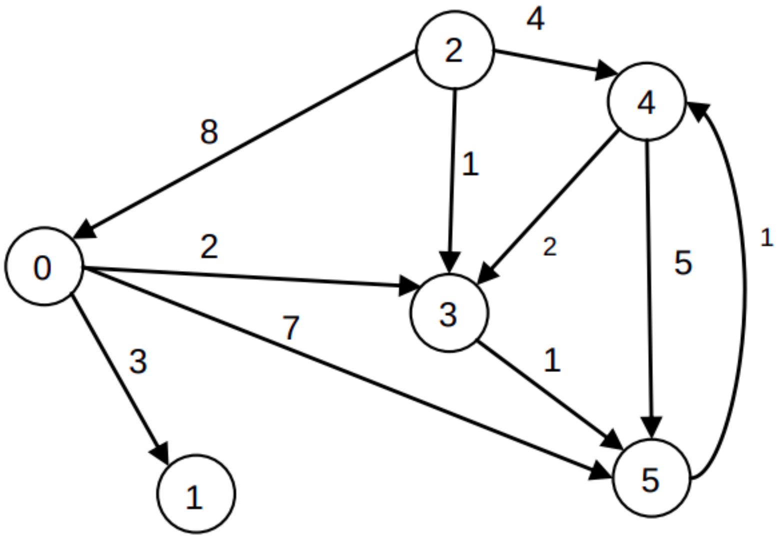

Рисунок 1.1 – Ориентированный взвешенный граф

Списки смежности

При представлении графа списками смежности для каждой вершины i хранится список W[i] смежных с ней вершин, как на рисунке 1.2.

$$
\begin{aligned}
W[1] &= [2, 4, 6] \\
W[2] &= [] \\
W[3] &= [1, 4, 5] \\
W[4] &= [6] \\
W[5] &= [4, 6] \\
W[6] &= [5]
\end{aligned}
$$

Рисунок 1.2 – Матрица смежности

Таким образом, весь граф можно представить одним списком, состоящим из вложенных списков смежности вершин.

Матрица смежности

При представлении взвешенного графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где элемент A[i][j] равен весу ребра, если вершины i и j соединены ребром. Иначе равен определенному символу (для рассматриваемого графа 0):

$$A = \begin{pmatrix} 0 & 3 & 0 & 2 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.1}$$

Если граф не является взвешенным, то элемент матрицы смежности равен 1, если вершины соединены.

Матрица смежности требует $O(n^2)$ памяти и может оказаться неэффективным способом хранения дерева или разреженных графов. Но использование матрицы смежности позволяет применять при реализации вычислительных процедур анализа графов матричные алгоритмы обработки данных, которые можно распараллелить. Поэтому при реализации данной лабораторной работы граф будет представляться именно таким образом.

1.2 Алгоритм Флойда

Алгоритм Флойда позволяет найти кратчайшее расстояние между любыми двумя вершинами в графе [4].

Последовательный алгоритм

Будем считать, что в графе n вершин, пронумерованных числами от 0 до $n-1$. Граф задан матрицей смежности, вес ребра $i-j$ равен w_{ij}. При отсутствии ребра $i-j$ значение $w_{ij}=-1$, также будем считать, что $w_{ii}=0$.

Пусть значение a_{ij}^k равно длине кратчайшего пути из вершины i в вершину j, при этом путь может заходить в промежуточные вершины только с номерами меньшими k (не считая начала и конца пути). То есть a_{ij}^0 - это длина кратчайшего пути из i в j, который вообще не содержит промежуточных вершин, то есть состоит только из одного ребра $i - j$, поэтому $a_{ij}^0 = w_{ij}$. Значение $a_{ij}^1 = w_{ij}$ равно длине кратчайшего пути, который может проходить через промежуточную вершину с номером 0, путь с весом a_{ij}^2 может проходить через промежуточные вершины с номерами 0 и 1 и т. д. Путь с весом a_{ij}^n может проходить через любые промежуточные вершины, поэтому значение a_{ij}^n равно длине кратчайшего пути из i в j.

Алгоритм Флойда последовательно вычисляет a_{ij}^0, a_{ij}^1, a_{ij}^2, $\ldots$, a_{ij}^n, увеличивая значение параметра k. Начальное значение - $a_{ij}^0 = w_{ij}$.

Теперь предполагая, что известны значения a_{ij}^{k-1} вычислим a_{ij}^k. Кратчайший путь из вершины i в вершину j, проходящий через вершины с номерами, меньшими, чем k может либо содержать, либо не содержать вершину с номером $k-1$. Если он не содержит вершину с номером $k-1$, то вес этого пути совпадает с a_{ij}^{k-1}. Если же он содержит вершину $k-1$, то этот путь разбивается на две части: $i-(k-1)$ и $(k-1)-j$. Каждая из этих частей содержит промежуточные вершины только с номерами, меньшими $k-1$, поэтому вес такого пути равен $a_{i,k-1}^{k-1} + a_{k-1,i}^{k-1}$. Из двух рассматриваемых вариантов необходимо выбрать вариант наименьшей стоимости, поэтому:

$$a_{ij}^k = min(a_{ij}^{k-1}, a_{i,k-1}^{k-1} + a_{k-1,i}^{k-1}) \tag{1.2}$$

Параллельный алгоритм

Параллельный алгоритм Флойда заключается в том, что на k-й итерации мы распределяем k-ю строку между всеми процессами(или потоками), а затем каждый процесс(или поток) выполняет последовательный алгоритм Флойда для полосы матрицы смежности, одновременно обновляя значение матрицы.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунке 2.2 представлена схема последовательного алгоритма Флойда, на рисунке 2.3 - параллельного. Схема алгоритма нахождения крайтчайшего расстояния между двумя вершинами показана на рисунке 2.1. Схема организации многопоточности представлена на рисунке 2.4. Семафоры и мьютексы не используются.

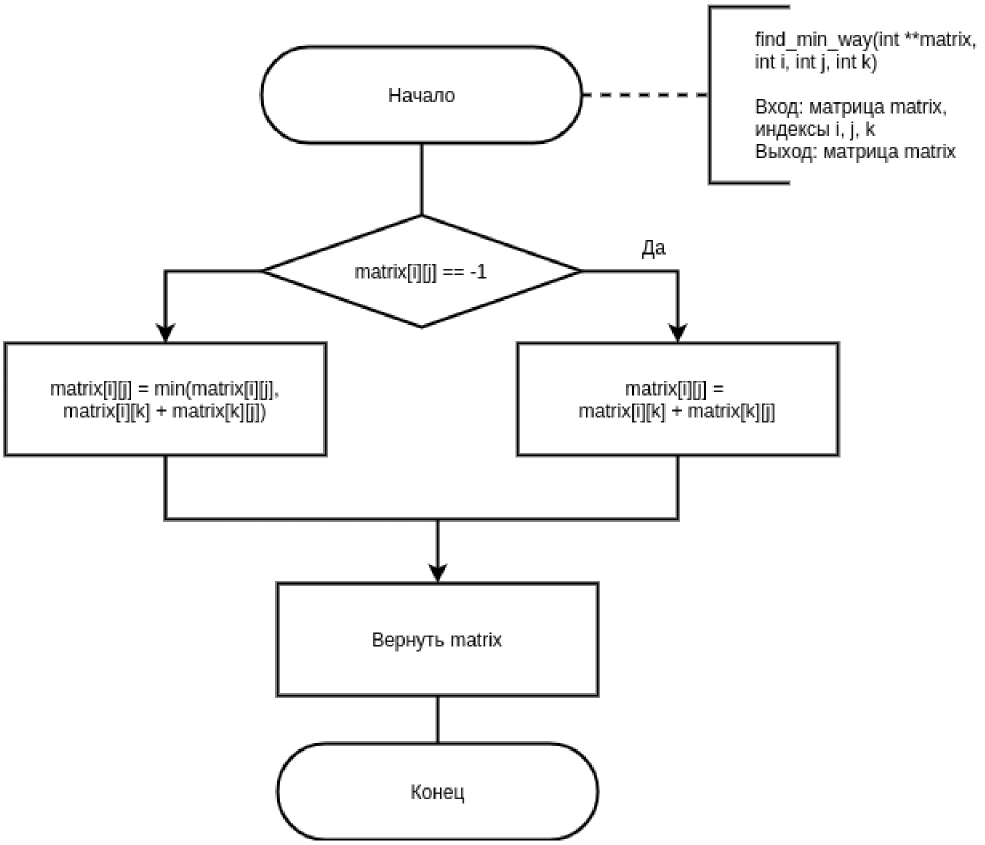

Рисунок 2.1 – Алгоритм нахождения кратчайшего пути между двумя вершинами

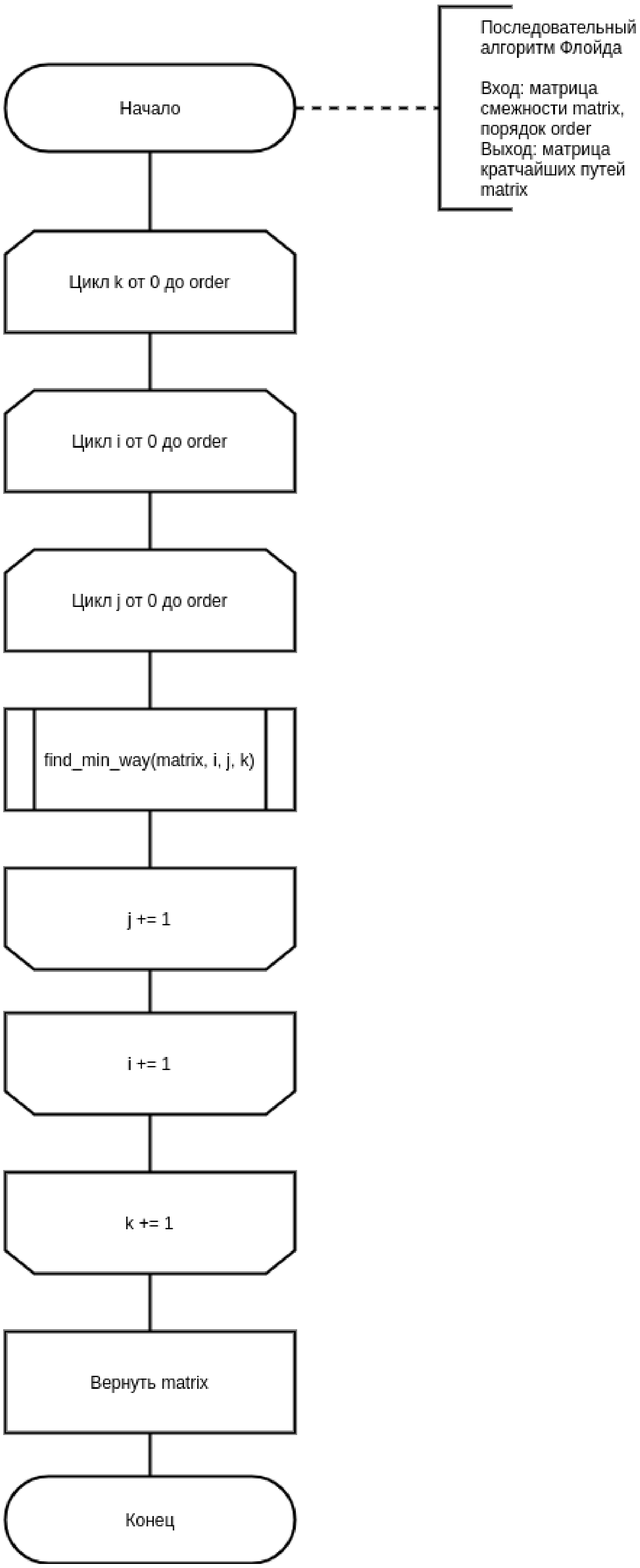

Рисунок 2.2 – Последовательный алгоритм Флойда

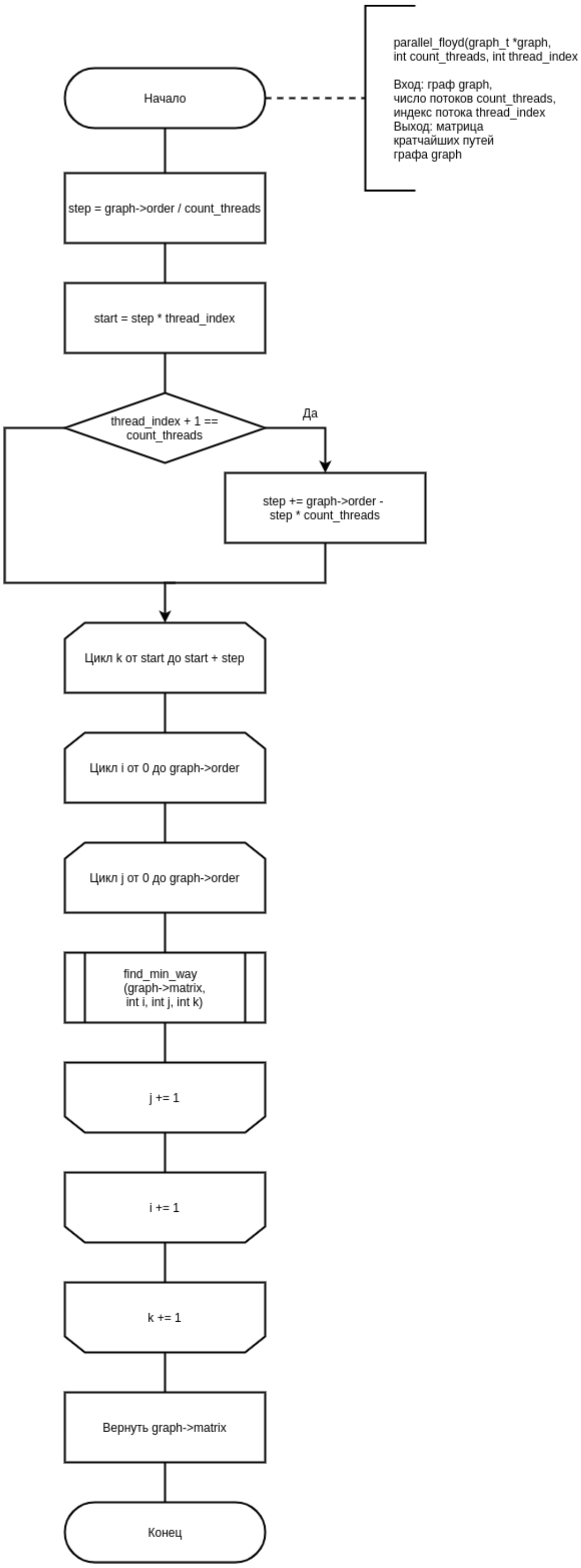

Рисунок 2.3 – Параллельный алгоритм Флойда

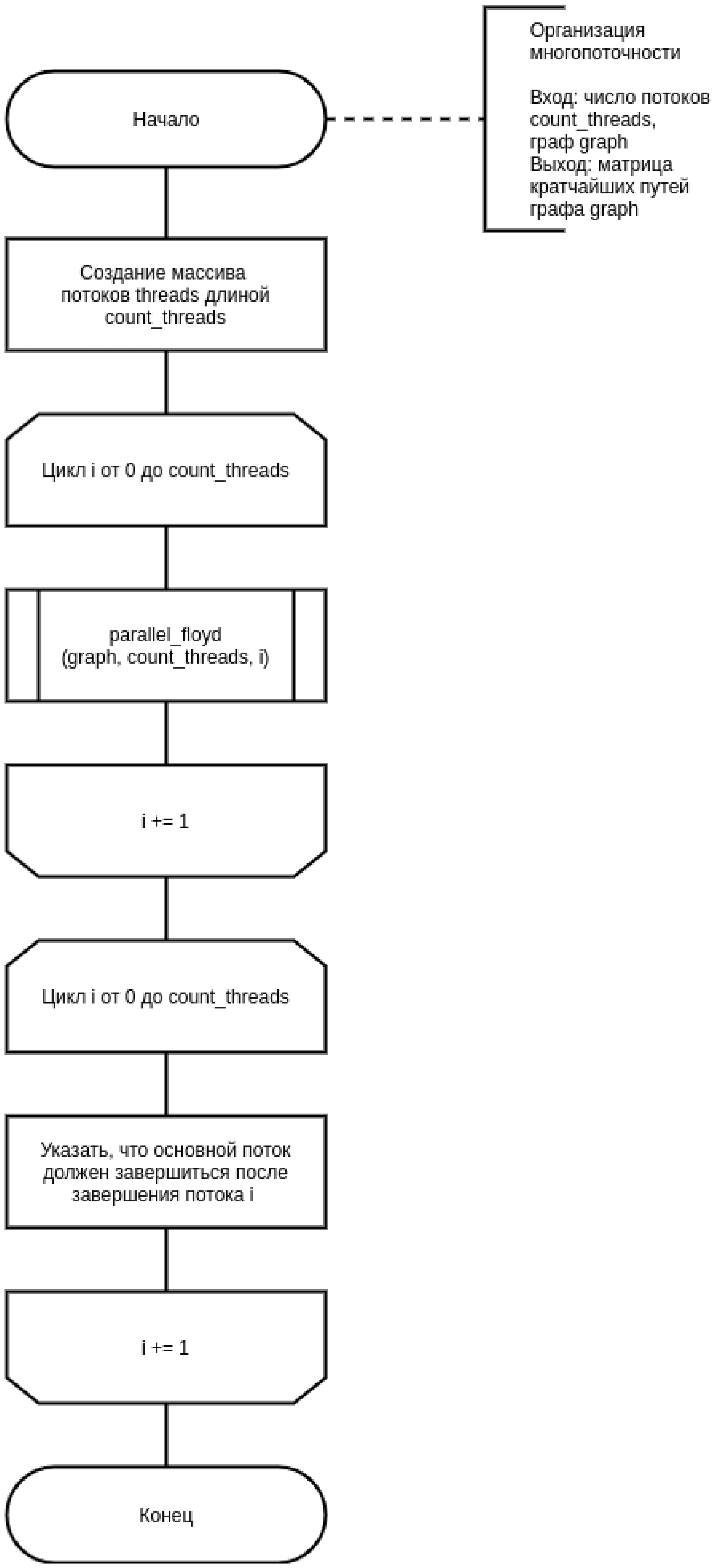

Рисунок 2.4 – Организация многопоточности

3 Технологическая часть

3.1 Требования к ПО

К программе предъявляется ряд требований:

— входными данными являются количество вершин графа и матрица смежности;

— элементы матрицы смежности – натуральные числа и число 0;

— на выходе – результирующая матрица с кратчайшими расстояниями между всеми вершинами.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран C++ – компилируемый, статически типизированный язык программирования общего назначения [5].

Данный выбор обусловлен поддержкой языком парадигмы объектно - ориентированного программирования, возможностью создавать нативные потоки и наличием методов для замера процессорного времени.

Время работы реализованных алгоритмов было замерено с помощью библиотеки chrono [6]. Для реализации многопоточности была использована библиотека thread [7].

3.3 Реализация алгоритмов

В листинге 3.1 приведена реализация последовательного алгоритма Флойда.

Листинг 3.1 – Реализация последовательного алгоритма Флойда

```
void Graph::simpleWA()
{
    for (int i =  0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            for (int k = 0; k < size; k++)
            {
                if (i != j && matrix[i][k] != 0 && matrix[k][j] != 0)
                {
                    if (matrix[i][j] == 0)
                        matrix[i][j] = matrix[i][k] + matrix[k][j];
                    else
                        matrix[i][j] = min(matrix[i][j], matrix[i][k] +
                            matrix[k][j]);
                }
            }
        }
    }
}
```

В листинге 3.2 приведена реализация параллельного алгоритма Флойда.

Листинг 3.2 – Реализация параллельного алгоритма Флойда

```

void Graph::parallelWA(int index)
{
    int step = size / countThreads;
    int start = index * step;

    if (index + 1 == countThreads)
    {
        step += + (size - step * countThreads);
    }

    for (int k = start; k < start + step; k++)
    {
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                if (i != j && matrix[i][k] != 0 && matrix[k][j] != 0)
                {
                    if (matrix[i][j] == 0)
                        matrix[i][j] = matrix[i][k] + matrix[k][j];
                    else
                        matrix[i][j] = min(matrix[i][j], matrix[i][k] +
                           matrix[k][j]);
                }
            }
        }
    }
}

void Graph::doParallel()
{
    std::vector<std::thread> threads(countThreads);

    for (int i = 0; i < countThreads; i++)
    {
        threads[i] = std::thread(&Graph::parallelWA, this, i);
    }

    for (int i = 0; i < countThreads; i++)
    {
        threads[i].join();
    }
}
```

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся замерный эксперимент:

— операционная система Ubuntu 22.04.1 LTS Linux x86_64 [8];

— память 8 ГБ;

— процессор Intel® Core™ i3-7130U;

— количество ядер – 4, количество логических ядер – 8.

Замеры проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно замерным экспериментом.

4.2 Пример работы программы

На рисунке 4.1 представлен пример работы программы. Вводится количество вершин в графе и матрица смежности. Далее выводится матрица кратчайших путей и время выполнения последовательного и параллельного алгоритма в микросекундах. Граф из данного примера изображен на рисунке 4.2. Также при компиляции программы можно указать вывод графа на экран.

```
dashori@fossa ~/P/A/b/l/code (main)> ./a.out
Введите количество вершин графа: 4
Введите 1 строчку матрицы смежности:
0 8 0 1
Введите 2 строчку матрицы смежности:
0 0 1 0
Введите 3 строчку матрицы смежности:
4 0 0 0
Введите 4 строчку матрицы смежности:
0 2 9 0
Матрица смежности:
0 8 0 1
0 0 1 0
4 0 0 0
0 2 9 0

Последовательный алгоритм:
Матрица кратчайших путей:
0 3 4 1
5 0 1 6
4 7 0 5
7 2 3 0
Время: 11 (микросекунды)

Параллельный алгоритм:
Введите количество потоков: 4
Матрица кратчайших путей:
0 3 4 1
5 0 1 6
4 7 0 5
7 2 3 0
Время: 876 (микросекунды)
```

Рисунок 4.1 – Пример работы программы

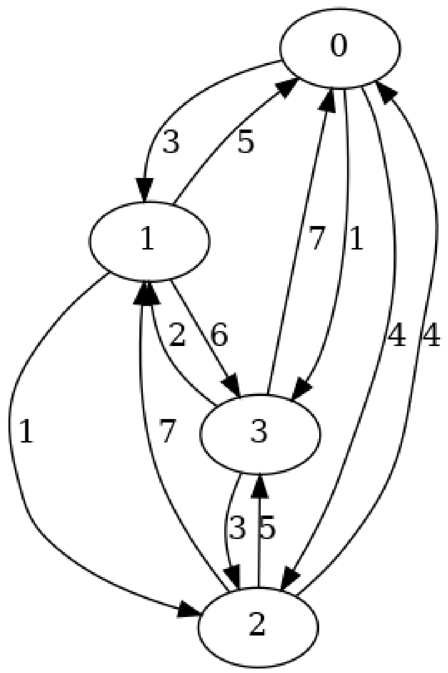

Рисунок 4.2 – Пример графа из тестовой программы

4.3 Время выполнения реализованных алгоритмов

Было проведено два эксперимента по замеру времени. Первый – зависимость времени работы реализованных алгоритмов от количества потоков, при этом количество вершин графа фиксированно и равно 150. Второй – зависимость времени от количества вершин графа, при этом для параллельной реализации алгоритма количество потоков фиксировано и равно 4.

Замеры времени работы реализованных алгоритмов для каждого эксперимента проводились 200 раз. В таблице 4.1 представлен результат зависимости времени работы реализованных алгоритмов от количества вершин в графе, количество потоков для параллельной реализации равно 4. В таблице 4.2 представлен результат зависимости времени работы реализованных алгоритмов от количества потоков, количество вершин графа равно 150.

Таблица 4.1 – Результаты замеров времени реализованных алгоритмов в микросекундах

Количество вершин в графе	Последовательный алгоритм	Параллельный алгоритм
1	1	428
11	99	162
21	548	617
31	1467	963
41	3401	1805
50	5831	3273
100	42865	20937
150	138661	67138
200	331876	156506

Таблица 4.2 – Результаты замеров времени реализованных алгоритмов в микросекундах

Количество потоков	Последовательный алгоритм	Параллельный алгоритм
1	137489	132363
2	274394	203034
3	411187	284118
4	546128	349896
5	681251	420196
6	815748	488302
7	950798	555731
8	1094473	625521
10	1367051	760079
11	1502139	826971
12	1637389	893409
13	1772377	960084
14	1907026	1028013
15	2042003	1093619
16	2176615	1160360

На рисунке 4.3 представлена зависимость времени работы реализованных алгоритмов от количества вершин в графе. На рисунке 4.4 представлена зависимость времени работы реализованных алгоритмов Флойда от количества потоков.

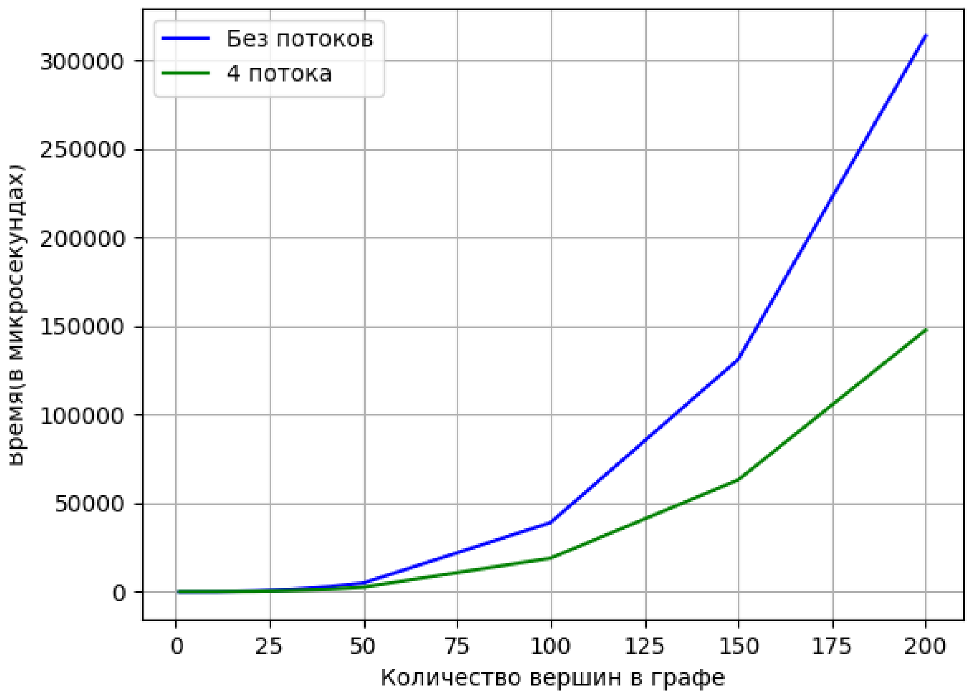

Рисунок 4.3 – Зависимость времени работы реализованных алгоритмов Флойда от количества вершин в графе

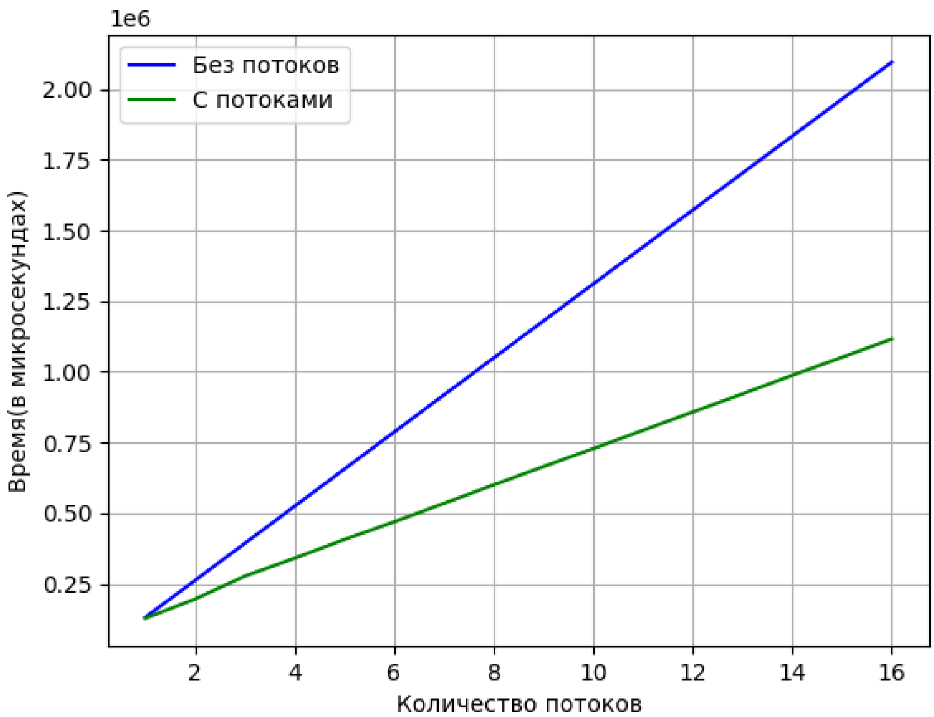

Рисунок 4.4 – Зависимость времени работы реализованных алгоритмов Флойда от количества потоков при фиксированном количестве вершин графа

Реализация параллельного алгоритма Флойда превосходит последовательную реализацию в 1.35 раз на двух потоках и в 1.87 раз при восьми потоках. Данный эксперимент проводился на графе с количеством вершин 150 и показывает разницу во времени между параллельной и последовательной реализации, которая зависит от количества потоков.

При сравнении работы реализованных алгоритмов от количества вершин графа было зафиксированно 4 потока. Параллельная реализация алгоритма Флойда выигрывает последовательную реализацию, когда количество вершин графа превышает 30. Если в графе 200 вершин, то параллельная реализация превосходит последовательную в 2.2 раза.

Заключение

В ходе выполнения лабораторной работы поставленная цель была достигнута: был получен навык организации параллельных вычислений на базе нативных потоков.

В ходе выполнения лабораторной работы были решены все задачи:

— проанализированы последовательный и параллельный варианты алгоритма поиска кратчайших расстояних между всеми парами вершин графа;

— реализованы выбранные алгоритмы;

— проведен анализ затрат работы реализованных алгоритмов по времени;

— на основе полученных в ходе экспериментов данных были сделаны выводы по поводу эффективности всех реализованных алгоритмов по времени;

— был подготовлен отчет по лабораторной работе.

Результат замерных экспериментов реализованных алгоритмов показал, что параллельная реализация алгоритма Флойда выигрывает последовательную реализацию, когда количество вершин графа больше 30. Если сравнивать по количеству потоков при фиксированном количестве вершин графа (и больше 30), то параллельная реализация работает быстрее при двух потоках в 1.35 раз и в 1.87 раз при восьми потоках.

Список использованных источников

[1] Многопоточность [Электронный ресурс]. Режим доступа: https://www.researchgate.net/publication/2468402_Software-Controlled_Multithreading_Using_Informing_Memory_Operations (дата обращения: 11.11.2022).

[2] Белоусов А. И. Ткачев С. Б. Дискретная математика. 2006. с. 744.

[3] Теория графов [Электронный ресурс]. Режим доступа: https://foxford.ru/wiki/informatika/teoriya-grafov (дата обращения: 11.11.2022).

[4] Алгоритм Флойда [Электронный ресурс]. Режим доступа: https://foxford.ru/wiki/informatika/algoritm-floyda (дата обращения: 11.11.2022).

[5] Язык программирования C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/ (дата обращения: 11.11.2022).

[6] Стандарт языка C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/files/papers/N4860.pdf (дата обращения: 11.11.2022).

[7] Стандарт языка C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/files/papers/N4800.pdf (дата обращения: 11.11.2022).

[8] Операционная система Ubuntu 22.04 LTS [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/jammy/ (дата обращения: 11.11.2022).