Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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»

Лабораторная работа № 4
по дисциплине «Анализ алгоритмов»

Тема Программирование параллельных потоков

Студент Жаринов М. А.

Группа ИУ7-52Б

Преподаватели Волкова Л. Л.

Москва, 2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Аналитическая часть **4**

1.1 Графы 4

1.2 Основные положения последовательного алгоритма 4

1.3 Основные положения параллельного алгоритма 5

2 Конструкторская часть **6**

2.1 Описание работы с семафором 6

2.2 Разработка последовательного алгоритма 6

2.3 Разработка параллельного алгоритма 7

3 Технологическая часть **12**

3.1 Средства реализации 12

3.2 Реализации алгоритмов 12

3.3 Функциональные тесты 15

4 Исследовательская часть **16**

4.1 Технические характеристики 16

4.2 Замеры времени 16

ЗАКЛЮЧЕНИЕ **23**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **24**

ВВЕДЕНИЕ

Цель данной лабораторной работы — разработка и сравнительный анализ последовательного и параллельного алгоритмов. Для достижения данной цели необходимо выполнить задачи:

1) описать последовательный алгоритм решения задачи поиска в ориентированном графе всех вершин с количеством связей большим или меньшим, чем входной параметр (большим или меньшим — параметр поиска);

2) разработать параллельную версию алгоритма;

3) реализовать обе версии алгоритма;

4) выполнить сравнительный анализ зависимостей времени решения задач от размерности входа для реализации последовательного алгоритма и для реализации модифицированного алгоритма, запущенной с единственным вспомогательным (рабочим) потоком;

5) выполнить сравнительный анализ зависимостей времени решения задач от размерности входа для реализации модифицированного алгоритма при k вспомогательных (рабочих) потоках, k принимает значения $1, 2, 4, ..., 8 \cdot q$, где q — количество логических ядер процессора ЭВМ;

6) сформулировать рекомендацию о выборе k для решения задачи на ЭВМ.

1 Аналитическая часть

1.1 Графы

Граф — математический объект, состоящий из двух множеств. Одно из них — любое конечное множество, его элементы называются вершинами графа. Другое множество состоит из пар вершин, эти пары называются рёбрами графа. Если множество вершин графа обозначено буквой V, множество рёбер — буквой E, а сам граф — буквой G, то граф можно обозначить как $G = (V, E)$. Если рёбра — упорядоченные пары, то такой граф называется *ориентированным* (сокращённо орграф), если же неупорядоченные, то *неориентированным*. Ребро (a, b) соединяет вершины a и b. В графе может быть не более одного ребра, соединяющего две данные вершины. Ребро типа (a, a), то есть соединяющее вершину с ней же самой, называют петлей. *Степенью* вершины v называется количество рёбер, инцидентных этой вершине. В случае ориентированного графа различают степень входа (количество входящих дуг) и степень выхода (количество исходящих дуг). Под количеством связей подразумевается степень вершины (сумма входящих и исходящих дуг) [1].

Для представления графа в памяти ЭВМ существует несколько способов:

— матрица смежности — квадратная матрица размера $|V| \times |V|$, где элемент a_{ij} равен 1 (или весу ребра), если существует ребро между вершинами i и j, и 0 в противном случае;

— списки смежности — массив списков, где для каждой вершины i хранится список вершин, смежных с ней.

Для разреженного графа использование матрицы смежности нецелесообразно из-за высоких затрат памяти $O(|V|^2)$. Наиболее подходящей структурой данных являются списки смежности, требующие $O(|V| + |E|)$ памяти.

Для упрощения разработки параллельного алгоритма в качестве структуры данных для представления графа были выбраны два списка смежности — в одном хранятся вершины, соединённые с данной исходящей из неё дугой, а в другом — вершины, дуги из которых входят в данную.

1.2 Основные положения последовательного алгоритма

Алгоритм последовательной обработки выглядит следующим образом:

1) организуется цикл по всем вершинам графа от 0 до $|V| - 1$;

2) для текущей вершины вычисляется количество смежных вершин (сумма размеров списков смежности);

3) полученное значение сравнивается с параметром K в соответствии с выбранным типом сравнения;

4) если условие выполняется, идентификатор вершины добавляется в результирующий список;

5) по завершении цикла возвращается список найденных вершин;

1.3 Основные положения параллельного алгоритма

Для реализации параллельных вычислений используются следующие базовые понятия:

— поток — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Потоки в рамках одного процесса разделяют общую память, но имеют собственный стек и регистровый контекст [2];

— семафор — примитив синхронизации, представляющий собой целочисленный счётчик. Он позволяет ограничить количество потоков, одновременно выполняющих определённый участок кода, блокируя доступ при достижении счётчиком нулевого значения, а также передавать информацию между потоками [2].

Поскольку обработка каждой вершины (проверка её степени) не зависит от результатов обработки других вершин, итерации основного цикла могут выполняться параллельно. Для объединения результатов работы потоков необходимо решить проблему записи в общий массив. Если организовать к нему монопольный доступ, то преимущество от параллельной обработки будет сильно уменьшено. Поэтому реализован следующий алгоритм работы:

1) главный поток определяет общее количество вершин $|V|$ и количество рабочих потоков T;

2) множество вершин разбивается на T диапазонов. Например, поток с индексом i будет обрабатывать вершины в диапазоне $[start_i, end_i)$;

3) главный поток запускает T вспомогательных потоков, передавая каждому границы его диапазона, указатель на граф и параметры поиска;

4) каждый вспомогательный поток считает количество подходящих вершин в своём диапазоне, передаёт эту информацию главному потоку и ожидает разрешения для перехода на второй этап (блокируется на семафоре);

5) после завершения первого этапа всеми потоками, главный поток рассчитывает смещения для записи в общем массиве для каждого потока, и разрешает потокам перейти на второй этап с помощью другого семафора;

6) найденные вершины записываются вспомогательными потоками в общий массив по рассчитанным смещениям.

Для синхронизации работы потоков (ожидание главным потоком завершения первого этапа, получение разрешения для перехода на второй этап) используются семафоры, так как именно это средство синхронизации служит для обмена информацией между потоками.

Вывод

В аналитической части были рассмотрены определения графа, потока и семафора, описаны основные положения последовательного и параллельного алгоритма.

2 Конструкторская часть

2.1 Описание работы с семафором

Взаимодействие с семафором осуществляется посредством двух основных атомарных операций:

— ожидание (wait или acquire) — операция, которая проверяет состояние счётчика. Если значение больше нуля, оно уменьшается на единицу, и поток продолжает выполнение. Если значение равно нулю, поток блокируется и переходит в режим ожидания до освобождения ресурса;

— сигнал (signal, post или release) — операция, которая увеличивает значение счётчика на единицу. Если при этом существуют потоки, ожидающие на семафоре, один из них перестаёт быть заблокированным и получает доступ к ресурсу.

2.2 Разработка последовательного алгоритма

На рисунке 2.1 показана схема последовательного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр.

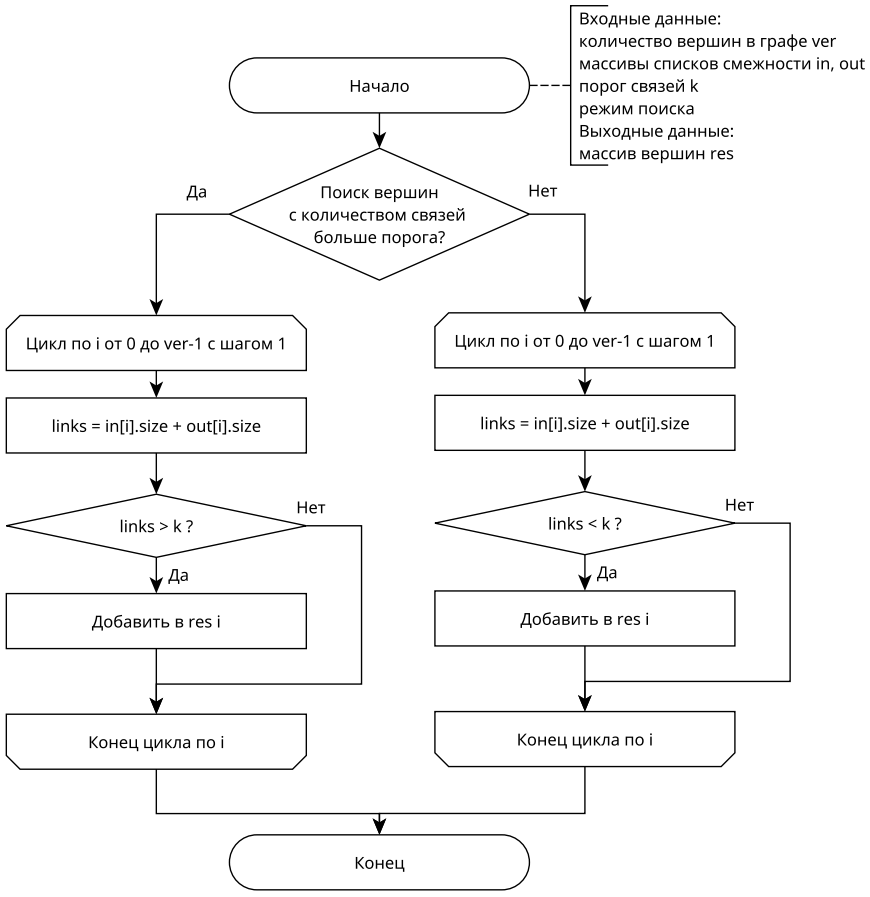

Рисунок 2.1 — Схема последовательного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр

2.3 Разработка параллельного алгоритма

На рисунках 2.2, 2.3 и 2.4 показана схема главного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр.

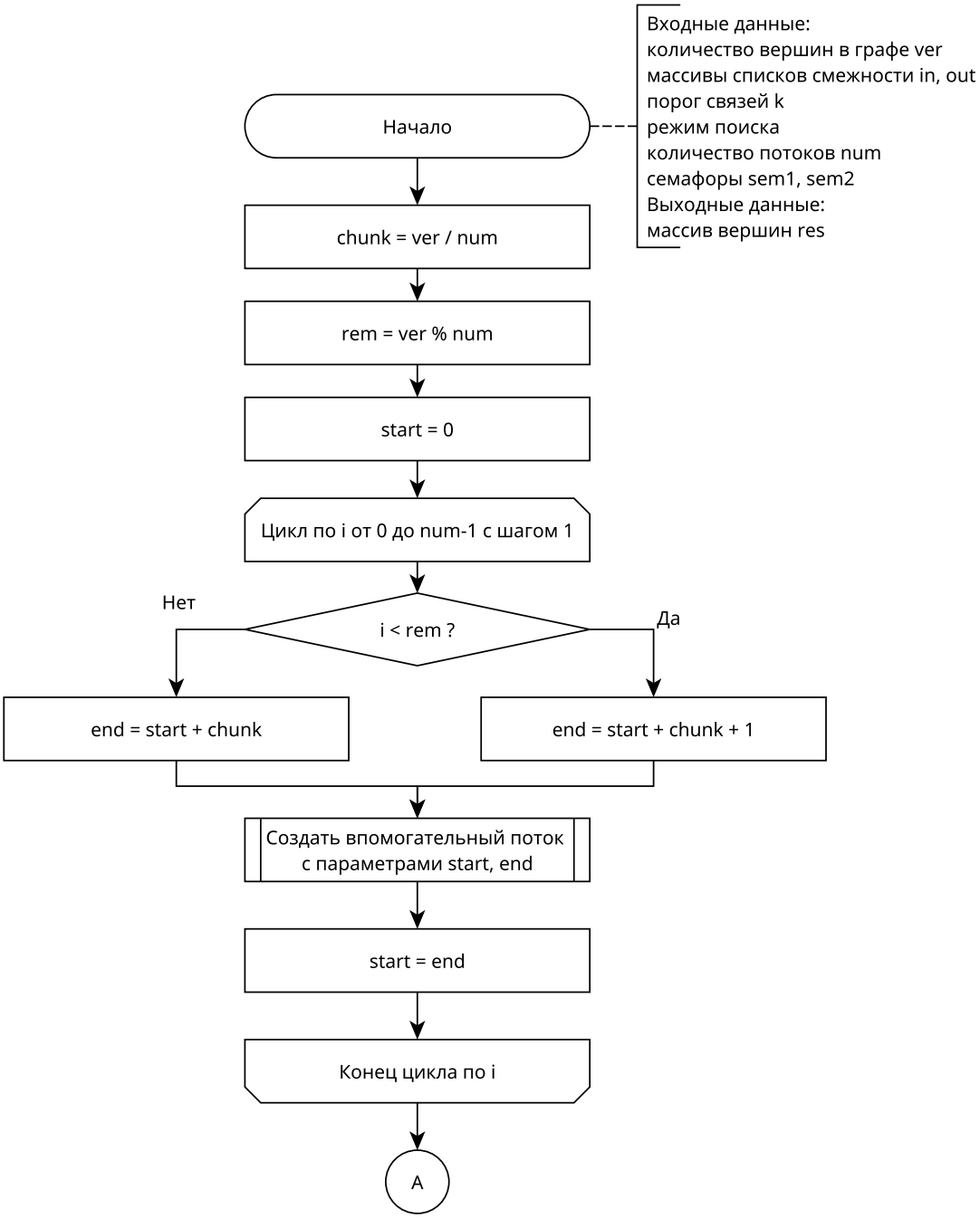

Рисунок 2.2 — Схема главного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр

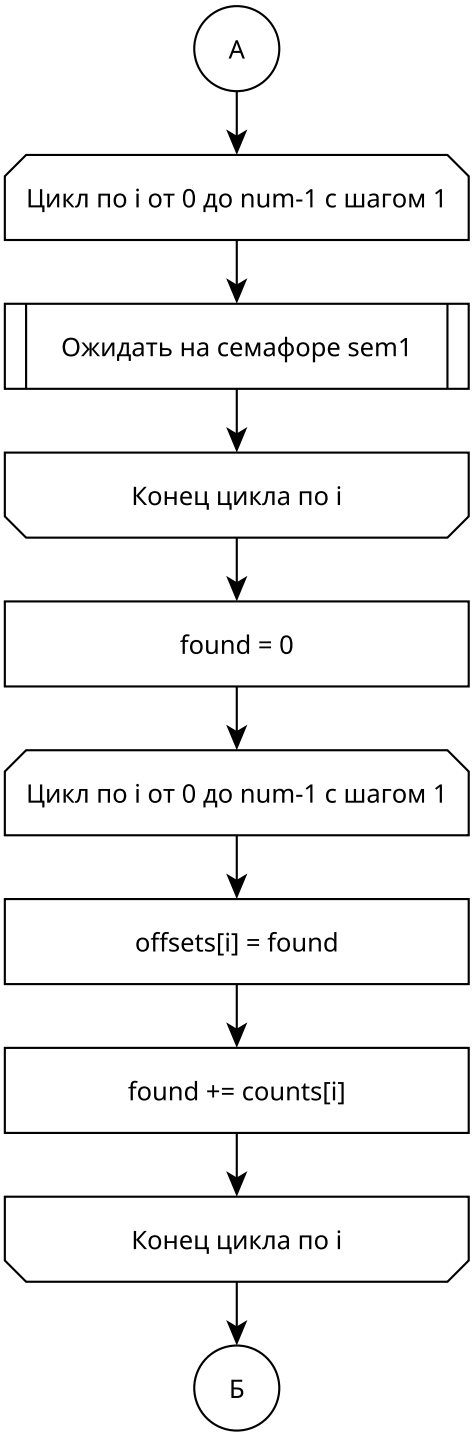

Рисунок 2.3 — Схема главного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр

Рисунок 2.4 — Схема главного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр

На рисунках 2.5 и 2.6 показана схема вспомогательного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр.

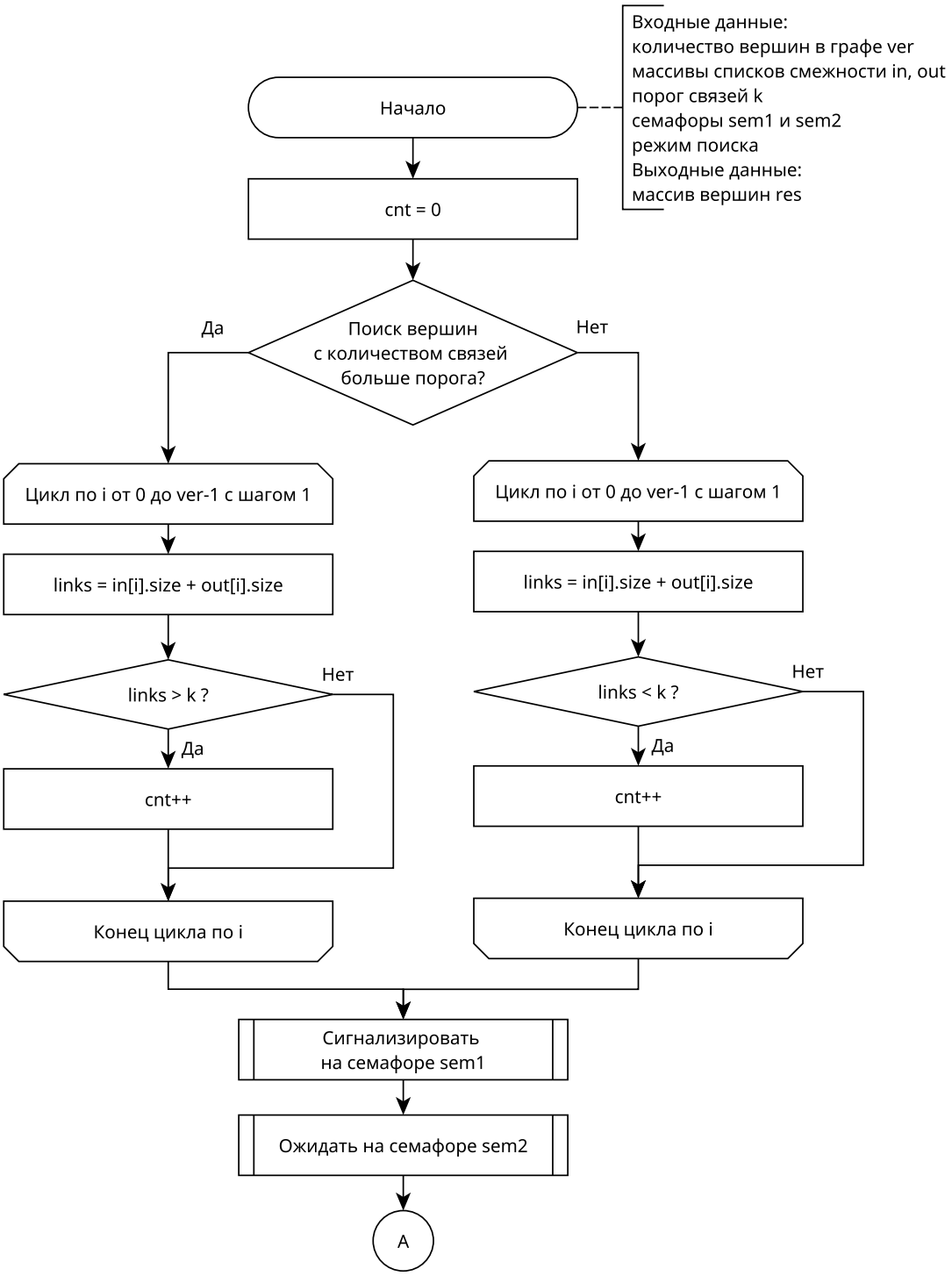

Рисунок 2.5 — Схема вспомогательного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр

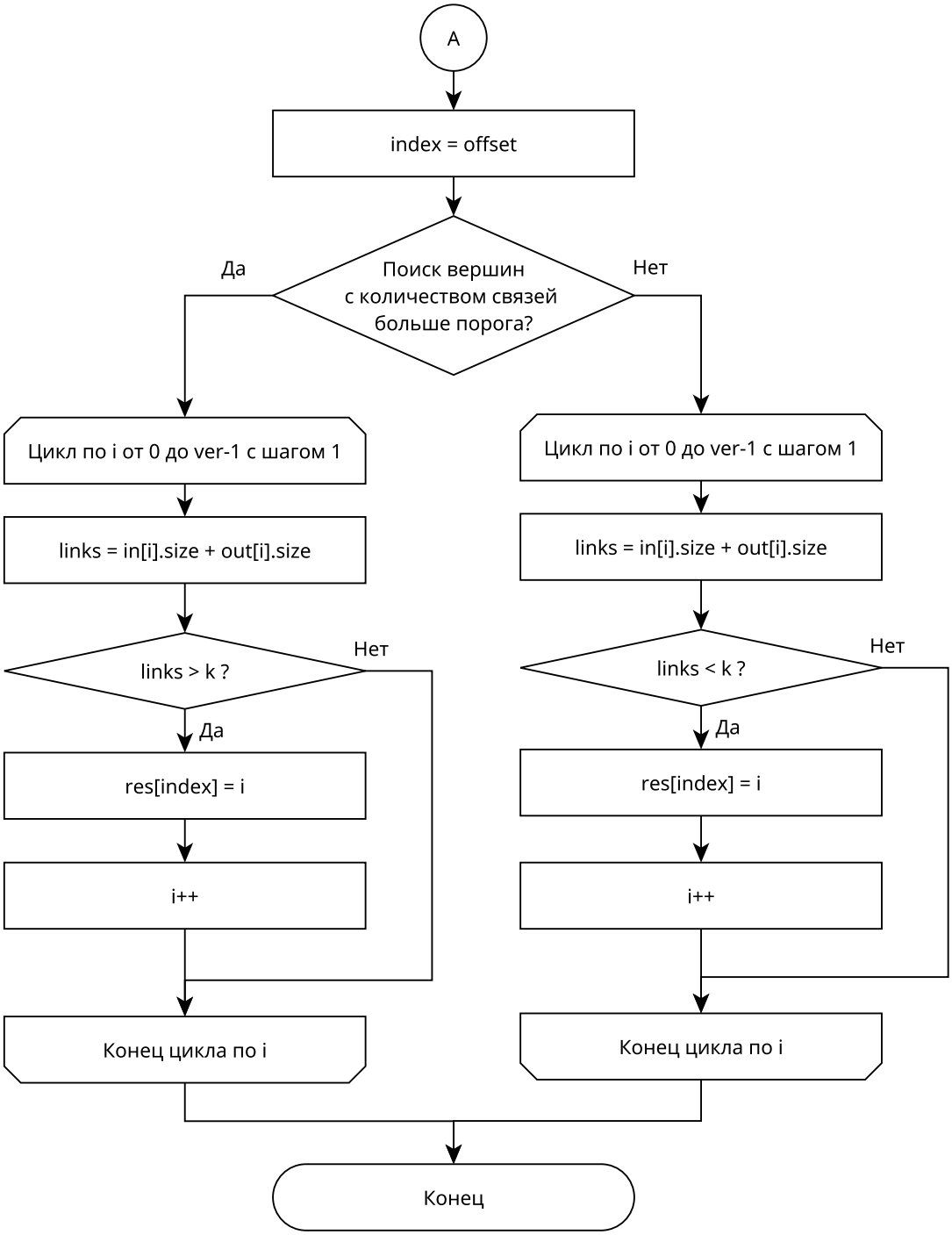

Рисунок 2.6 — Схема вспомогательного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр

Вывод

В конструкторской части были описаны используемые структуры данных (семафоры), разработаны последовательный и параллельный алгоритмы поиска в ориентированном графе всех вершин с количеством связей большим или меньшим, чем входной параметр.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов был выбран язык C++ (стандарт C++20). Для замера реального времени используется стандартная библиотека chrono [5]. Для работы с семафорами используется заголовочный файл semaphore.h [3]. Для создания потоков используется стандартная библиотека thread [6].

Для работы с сущностью вспомогательного потока необходимо соблюдать следующий порядок:

— создать и запустить поток с помощью создания объекта класса std::thread;

— при создании в конструктор передать первым аргументом функцию, которую начнёт исполнять создаваемый поток, дальнейшие аргументы будут переданы данной функции;

— ожидать завершения потока с помощью метода join.

3.2 Реализации алгоритмов

В листинге 3.1 представлен исходный код реализации последовательного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр.

```
void solveSequential(const Graph &g, int k, CompareMode mode, int
   activeVertices, std::vector<int> &resultVector) {
  resultVector.clear();
  const auto *in_ptr = g.in_edges.data();
  const auto *out_ptr = g.out_edges.data();
  size_t k_sz = static_cast<size_t>(k);
  if (mode == MODE_GREATER) {
    for (int i = 0; i < activeVertices; ++i) {
      if (in_ptr[i].size() + out_ptr[i].size() > k_sz) {
        resultVector.push_back(i);
      }
    }
  } else {
    for (int i = 0; i < activeVertices; ++i) {
      if (in_ptr[i].size() + out_ptr[i].size() < k_sz) {
        resultVector.push_back(i);
      }
    }
  }
}
```

Листинг 3.1 — Реализация последовательного алгоритма поиска в графе вершин с необходимым количеством связей

В листинге 3.2 представлен исходный код реализации главного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр.

```
void solveParallel(const Graph &g, int k, CompareMode mode, int
    numThreads, int activeVertices, std::vector<int> &resultVector) {
  resultVector.clear();
  SharedContext ctx(numThreads);
  ctx.graph = &g;
  ctx.K = k;
  ctx.mode = mode;
  ctx.resultVector = &resultVector;
  std::vector<std::thread> threads;
  threads.reserve(numThreads);
  int chunkSize = activeVertices / numThreads;
  int remainder = activeVertices % numThreads;
  int start = 0;

  for (int i = 0; i < numThreads; ++i) {
    int end = start + chunkSize + (i < remainder ? 1 : 0);
    threads.emplace_back(workerThread, std::ref(ctx), i, start, end);
    start = end;
  }

  for (int i = 0; i < numThreads; ++i)
    sem_wait(&ctx.semFinishedStage1);

  int totalFound = 0;
  for (int i = 0; i < numThreads; ++i) {
    ctx.tData[i].offset = totalFound;
    totalFound += ctx.tData[i].count;
  }
  resultVector.resize(totalFound);

  for (int i = 0; i < numThreads; ++i)
    sem_post(&ctx.semStartStage2);
  for (auto &t : threads)
    t.join();
}
```

Листинг 3.2 — Реализация главного потока параллельного алгоритма поиска в графе вершин с необходимым количеством связей

В листинге 3.3 представлен исходный код реализации вспомогательного потока параллельного алгоритма поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр.

```
void workerThread(SharedContext &ctx, int threadId, int startIdx, int
   endIdx) {
  const auto *in_ptr = ctx.graph->in_edges.data();
  const auto *out_ptr = ctx.graph->out_edges.data();
  int localCount = 0;
  size_t k_sz = static_cast<size_t>(ctx.K);
  if (ctx.mode == MODE_GREATER) {
    for (int i = startIdx; i < endIdx; ++i) {
      if (in_ptr[i].size() + out_ptr[i].size() > k_sz)
        localCount++;
    }
  } else {
    for (int i = startIdx; i < endIdx; ++i) {
      if (in_ptr[i].size() + out_ptr[i].size() < k_sz)
        localCount++;
    }
  }
  ctx.tData[threadId].count = localCount;
  sem_post(&ctx.semFinishedStage1);
  sem_wait(&ctx.semStartStage2);
  int writeIndex = ctx.tData[threadId].offset;
  int *res_raw_ptr = ctx.resultVector->data();
  if (ctx.mode == MODE_GREATER) {
    for (int i = startIdx; i < endIdx; ++i) {
      if (in_ptr[i].size() + out_ptr[i].size() > k_sz) {
        res_raw_ptr[writeIndex++] = i;
      }
    }
  } else {
    for (int i = startIdx; i < endIdx; ++i) {
      if (in_ptr[i].size() + out_ptr[i].size() < k_sz) {
        res_raw_ptr[writeIndex++] = i;
      }
    }
  }
}
```

Листинг 3.3 — Реализация вспомогательного потока параллельного алгоритма поиска в графе вершин с необходимым количеством связей

3.3 Функциональные тесты

В таблице 3.1 представлены функциональные тесты для алгоритмов поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр.

Введены обозначения для описания графа: N — количество вершин, список пар (u, v) обозначает наличие дуги из вершины u в вершину v.

Таблица 3.1 — Таблица функционального тестирования реализаций алгоритма

Граф	Запрос	Ожидаемый результат
$N = 4$. Дуги: $(0, 1)$, $(0, 2)$, $(1, 0)$.	Связей > 2	$\{0\}$
$N = 5$. Дуги: $(0, 1)$.	Связей < 1	$\{2, 3, 4\}$
$N = 3$. Дуги: $(0, 1)$, $(1, 2)$, $(2, 3)$, $(3, 0)$.	Связей > 5	$\emptyset$ (пусто)

Все тесты пройдены успешно для реализаций последовательного и параллельного алгоритмов.

Вывод

В технологической части определены необходимые средства реализации, и с их помощью реализованы последовательный и параллельный алгоритмы поиска в графе вершин с количеством связей большим или меньшим, чем входной параметр, описаны функциональные тесты.

4 Исследовательская часть

4.1 Технические характеристики

Замеры времени проводились на ноутбуке с характеристиками:

— операционная система — Debian Unstable GNU/Linux с ядром версии 6.17.12;

— процессор — AMD Ryzen 5 7535HS с 6 физическими ядрами и 12 логическими ядрами с максимальной тактовой частотой 4.60 ГГц [4];

— оперативная память — 16 Гбайт типа LPDDR5 в 4 каналах с тактовой частотой 6400 МГц и задержками CL 19 tRCD 15 tRP 17 tRAS 34 tRC 51.

4.2 Замеры времени

Замеры времени проводились для случайных ориентированных графов размерами от 25000 до 800000 вершин с шагом изменения 25000 для реализации последовательного алгоритма и для реализации параллельного алгоритма с k вспомогательными потоками, k принимает значения $1, 2, 4, ..., 128$. Для каждого размера производилось $m = 100$ измерений, результатом замера является среднее арифметическое из этих измерений. Время измерялось в миллисекундах. Во время выполнения замеров ноутбук был подключён к сети, никаких прочих программ, кроме системных, запущено не было.

Результаты измерений представлены в таблице 4.1.

Таблица 4.1 — Результаты измерений времени поиска в графе вершин с необходимым количеством связей, мс

Размер	Последоват. алгоритм	Количество вспомогательных потоков в параллельном алгоритме							
		1	2	4	8	16	32	64	128
25000	0.177	0.406	0.232	0.179	0.286	0.495	0.982	1.809	3.913
50000	0.344	0.833	0.397	0.255	0.336	0.559	1.048	1.937	3.815
75000	0.527	1.169	0.613	0.367	0.395	0.693	1.140	2.066	4.173
100000	0.703	1.481	0.790	0.427	0.451	0.633	1.193	2.080	4.270
125000	0.852	1.888	1.022	0.540	0.514	0.707	1.100	2.172	4.298
150000	1.031	2.252	1.224	0.630	0.607	0.735	1.213	2.081	4.285
175000	1.162	2.423	1.473	0.697	0.656	0.769	1.251	2.187	4.320
200000	1.461	2.828	1.618	0.833	0.721	0.841	1.244	2.318	4.312
225000	1.482	3.144	1.817	0.900	0.789	0.873	1.324	2.328	4.178
250000	1.656	3.572	1.974	0.954	0.836	0.961	1.420	2.484	4.149
275000	1.824	3.961	2.192	1.070	0.927	1.086	1.480	2.513	4.177
300000	2.010	4.143	2.358	1.176	1.036	1.161	1.543	2.465	4.090
325000	2.160	4.395	2.501	1.262	1.107	1.220	1.603	2.499	4.238
350000	2.300	4.738	2.719	1.423	1.221	1.271	1.624	2.466	4.361
375000	2.491	5.105	2.853	1.539	1.292	1.343	1.580	2.424	4.305
400000	2.646	5.341	3.046	1.615	1.332	1.313	1.617	2.515	4.289
425000	2.807	5.647	3.228	1.731	1.455	1.398	1.680	2.585	4.480
450000	3.189	5.942	3.429	1.902	1.470	1.511	1.861	2.658	4.611
475000	3.204	6.241	3.596	2.032	1.523	1.569	1.910	2.905	4.644
500000	3.328	6.626	3.687	1.988	1.568	1.569	1.919	2.734	4.747
525000	3.505	6.988	3.998	2.154	1.773	1.753	2.045	3.045	4.805
550000	3.738	7.202	4.056	2.468	1.982	2.224	2.597	3.306	5.330
575000	4.109	7.565	4.281	2.320	1.801	1.805	2.304	2.913	4.941
600000	4.043	7.953	4.478	2.623	1.958	2.438	2.177	2.887	4.727
625000	4.144	8.164	4.658	2.474	1.950	1.986	2.250	3.001	4.912
650000	4.284	8.343	4.701	2.608	2.010	2.169	2.310	3.032	4.965
675000	4.449	8.672	4.833	2.718	2.079	2.183	2.391	3.139	4.909
700000	4.617	8.948	5.076	2.763	2.215	2.218	2.457	3.183	5.012
725000	4.777	9.217	5.245	2.927	2.256	2.356	2.515	3.214	5.012
750000	4.972	9.515	5.417	2.949	2.325	2.444	2.629	3.336	5.417
775000	5.152	9.890	5.675	3.406	2.668	2.694	3.336	3.761	5.092
800000	5.266	10.074	5.677	3.134	2.507	2.576	2.782	3.467	5.073

Графики зависимости времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 1 вспомогательным потоком приведены на рисунке 4.1. Из-за накладных расходов на создание и управление потоками, а также из-за двухэтапной работы алгоритма, реализация параллельного алгоритма с 1 вспомогательным потоком работает медленнее, чем реализация последовательного.

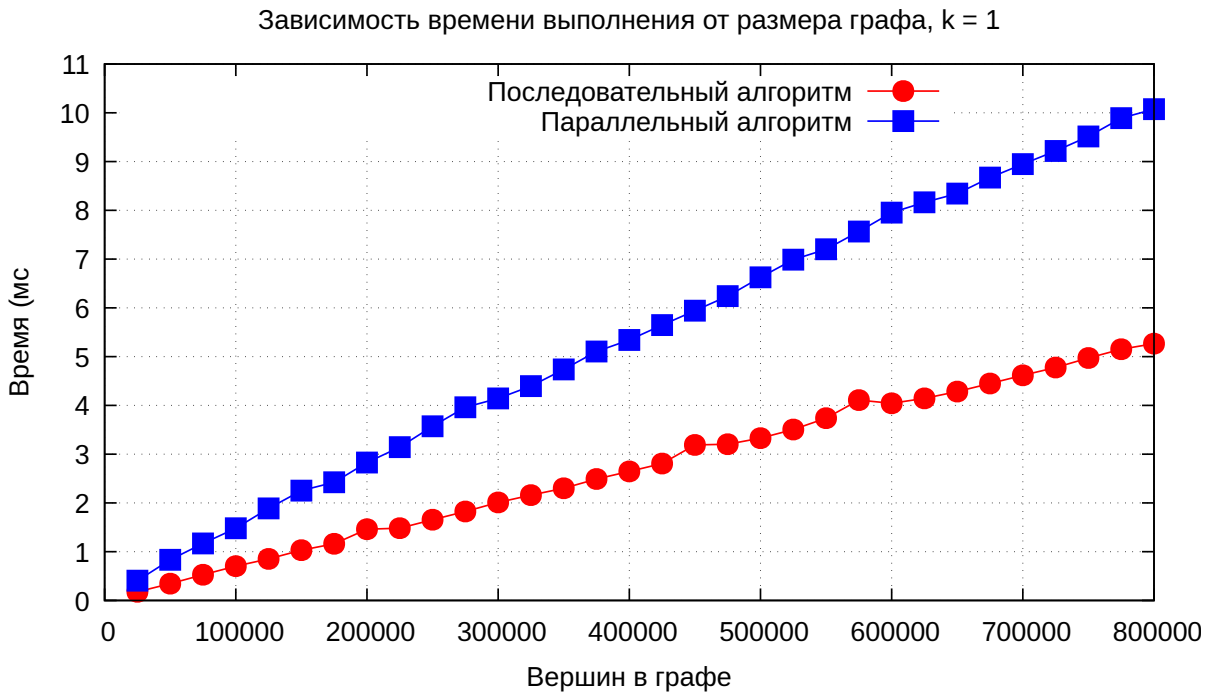

Рисунок 4.1 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 1 вспомогательным потоком

Графики зависимости времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 2 вспомогательными потоками приведены на рисунке 4.2.

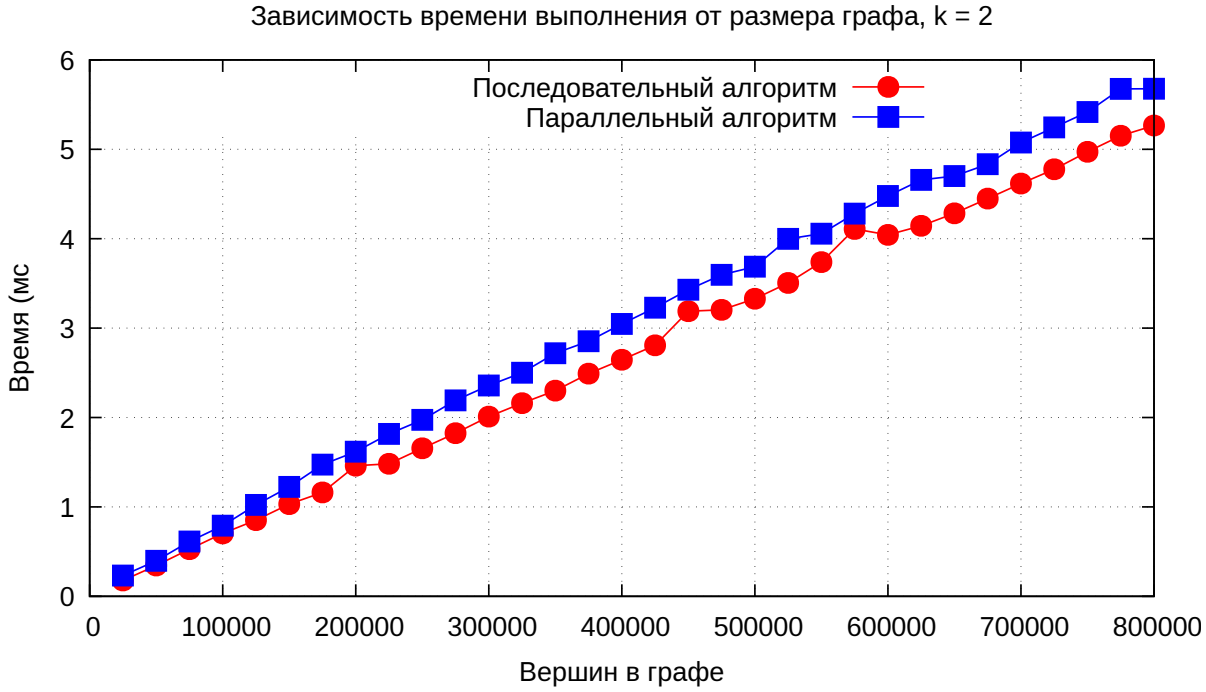

Рисунок 4.2 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 2 вспомогательными потоками

Графики зависимости времени выполнения от размера графа для реализаций последова-

тельного и параллельного алгоритма с 4 вспомогательными потоками приведены на рисунке 4.3.

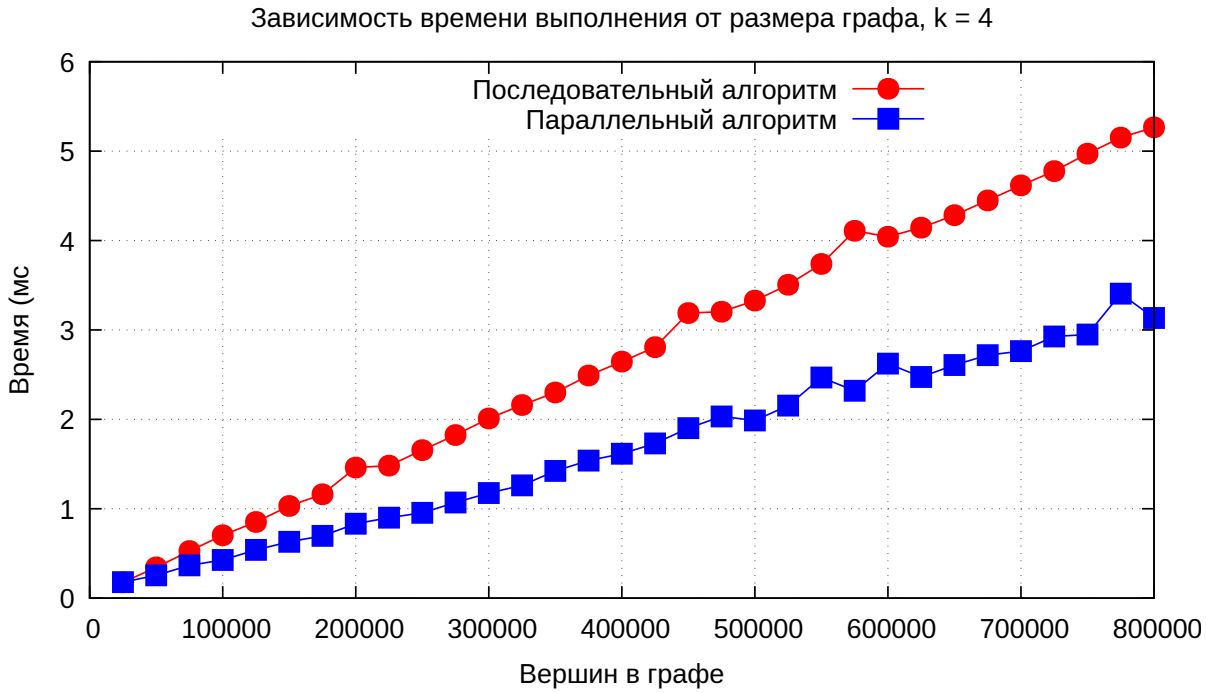

Рисунок 4.3 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 4 вспомогательными потоками

Графики зависимости времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 8 вспомогательными потоками приведены на рисунке 4.4.

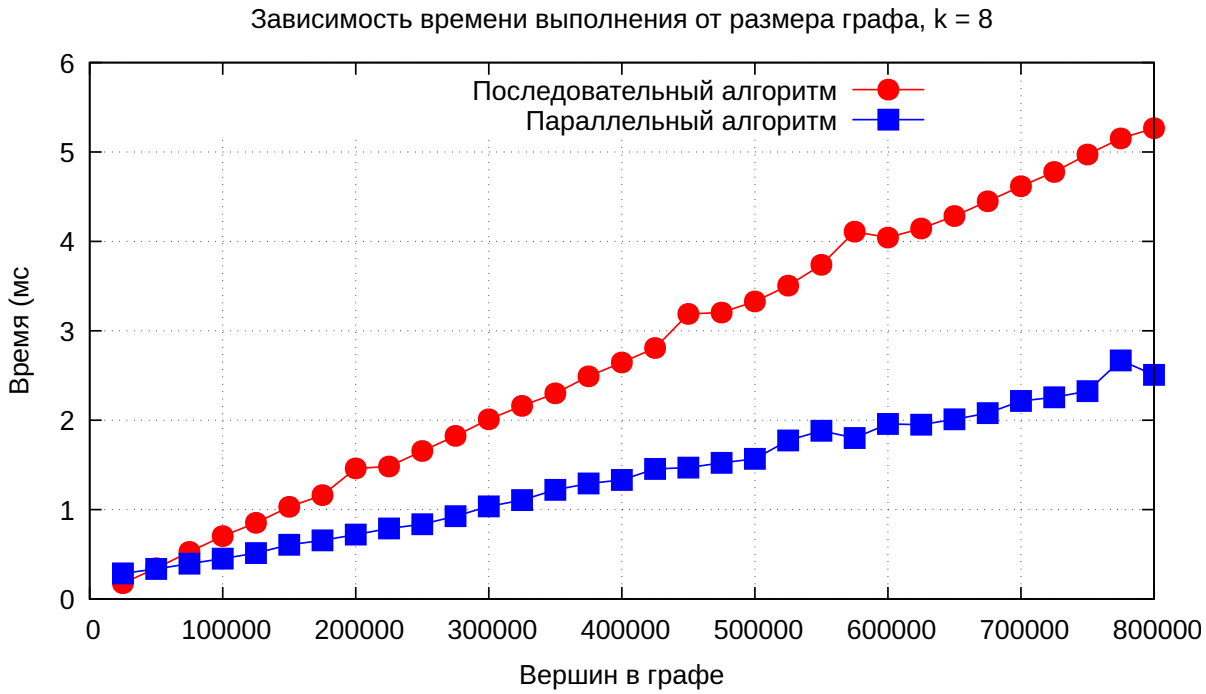

Рисунок 4.4 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 8 вспомогательными потоками

Графики зависимости времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 16 вспомогательными потоками приведены на рисунке 4.5.

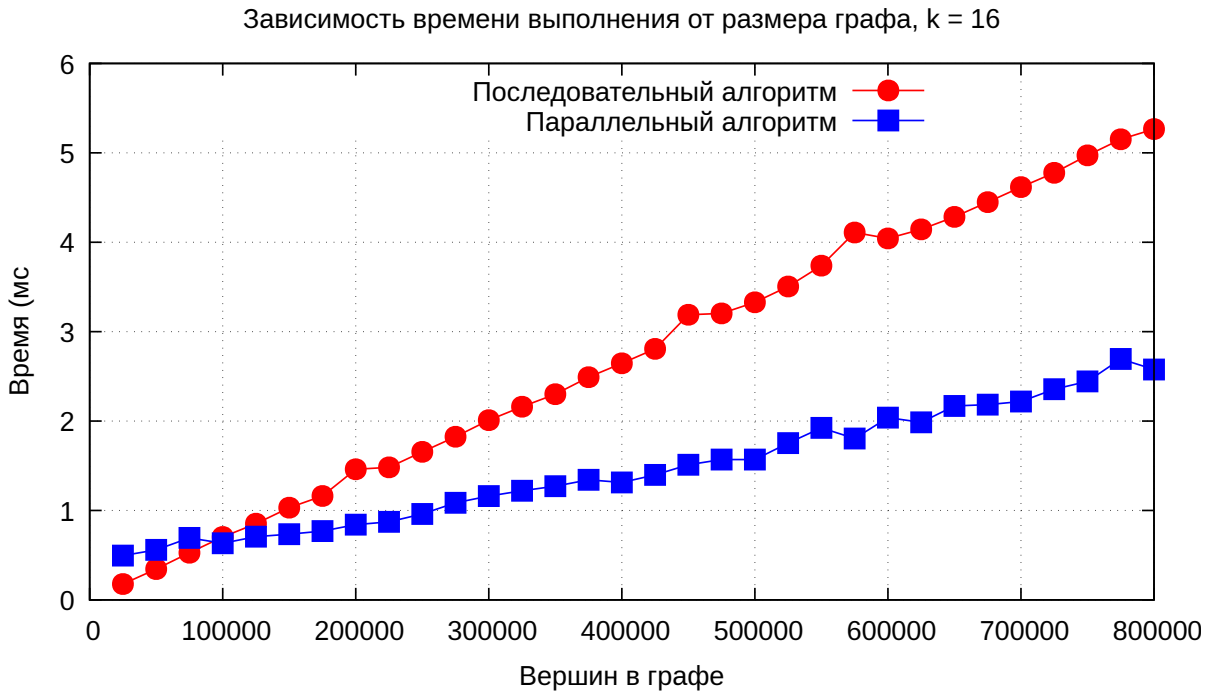

Рисунок 4.5 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 16 вспомогательными потоками

Графики зависимости времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 32 вспомогательными потоками приведены на рисунке 4.6.

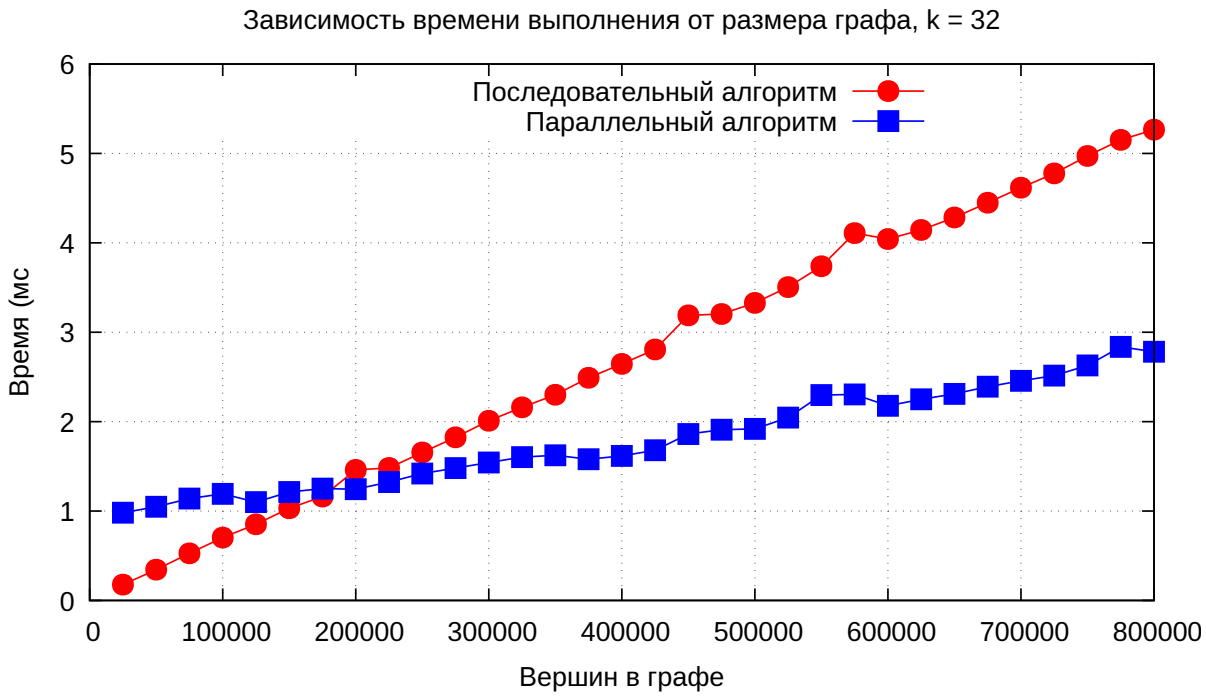

Рисунок 4.6 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 32 вспомогательными потоками

Графики зависимости времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 64 вспомогательными потоками приведены на рисунке 4.7.

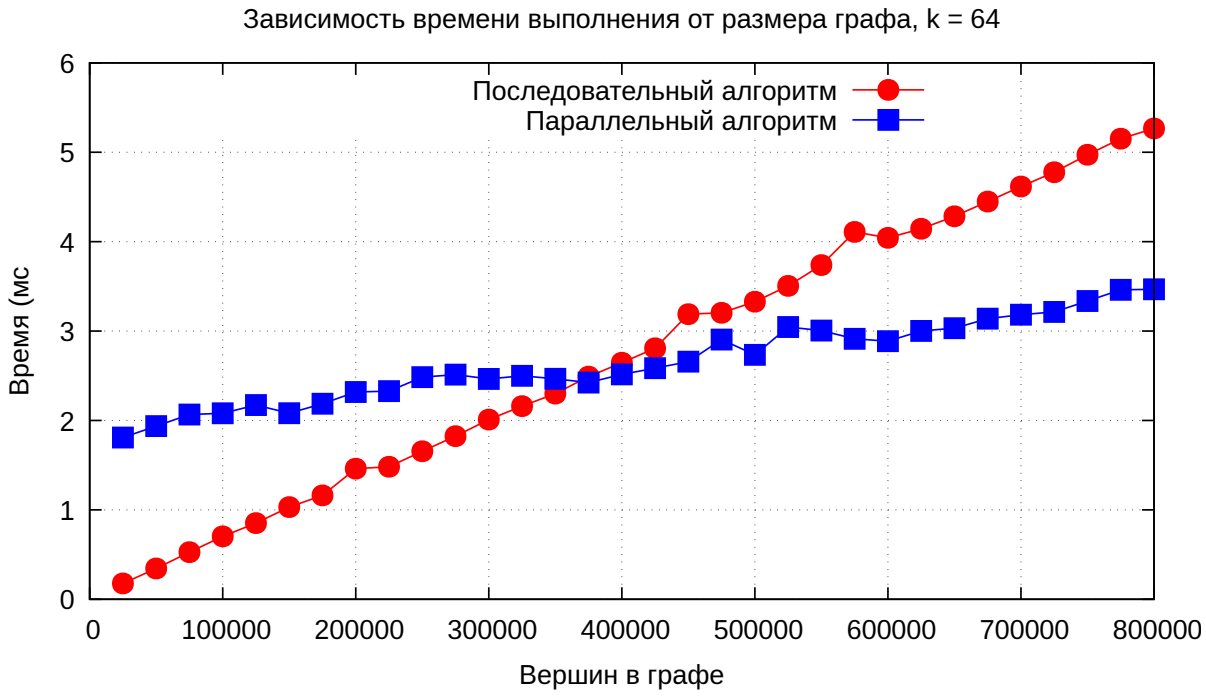

Рисунок 4.7 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 64 вспомогательными потоками

Графики зависимости времени выполнения от размера графа для реализаций последо-

вательного и параллельного алгоритма с 128 вспомогательными потоками приведены на рисунке 4.8.

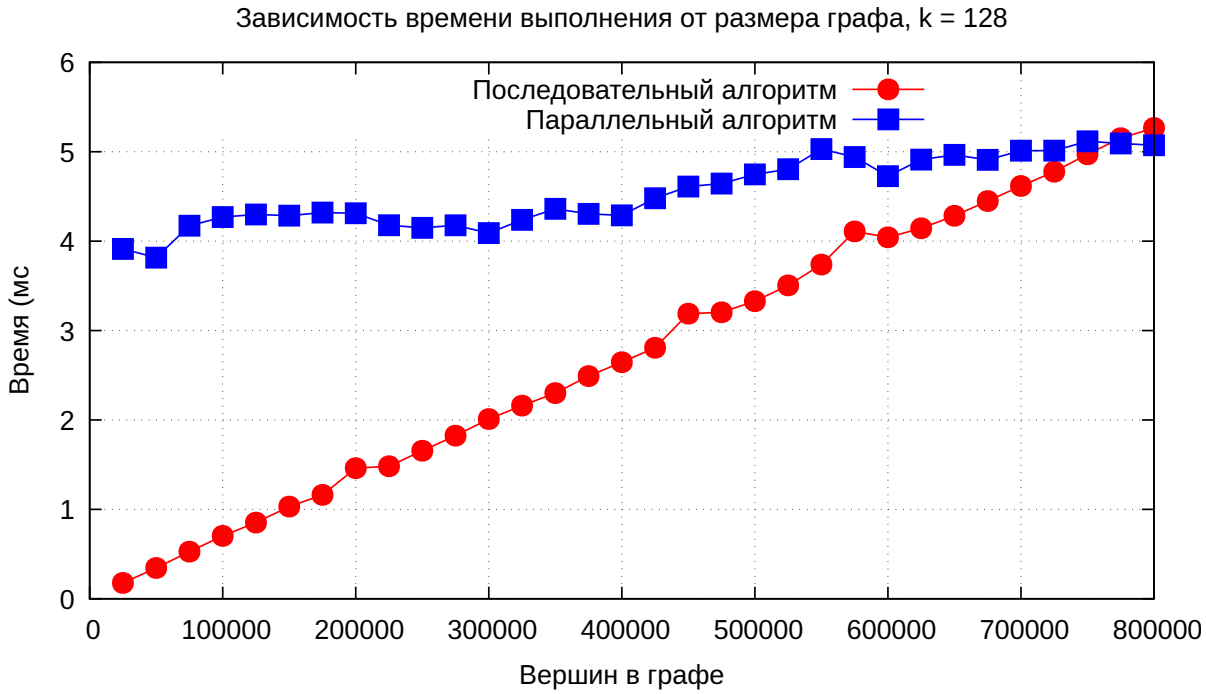

Рисунок 4.8 — Зависимость времени выполнения от размера графа для реализаций последовательного и параллельного алгоритма с 128 вспомогательными потоками

Вывод

В исследовательской части были произведены измерения зависимости времени работы реализаций алгоритмов от размера графа и от количества вспомогательных потоков. В результате для данной ЭВМ оптимальное значение k равно 8, это число обусловлено тем, что оно близко к числу физических ядер данной ЭВМ. При дальнейшем увеличении количества потоков время работы реализации параллельного алгоритма увеличивается из-за увеличения времени на создание и синхронизацию потоков.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута: были разработаны и сравнительно проанализированы последовательный и параллельный алгоритмы. Все задачи решены:

1) описан последовательный алгоритм решения задачи поиска в ориентированном графе всех вершин с количеством связей большим или меньшим, чем входной параметр (большим или меньшим — параметр поиска);

2) разработана параллельная версия алгоритма;

3) реализованы обе версии алгоритма;

4) выполнен сравнительный анализ зависимостей времени решения задач от размерности входа для реализации последовательного алгоритма и для реализации модифицированного алгоритма, запущенной с единственным вспомогательным (рабочим) потоком;

5) выполнен сравнительный анализ зависимостей времени решения задач от размерности входа для реализации модифицированного алгоритма при k вспомогательных (рабочих) потоках, k принимает значения $1, 2, 4, ..., 128$;

6) сформулирована рекомендация о выборе k для решения задачи на ЭВМ — для данной ЭВМ оптимальное значение k равно 8.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алексеев В. Е., Захарова Д. В. ТЕОРИЯ ГРАФОВ: Учебное пособие — Нижний Новгород: Нижегородский госуниверситет, 2017 — 119 с.

2. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. Пер. с англ. Слинкин А. А. — М.: ДМК Пресс, 2012. — 672 с.

3. sem_overview(7) — Linux manual page [Электронный ресурс]. Режим доступа: https://man7.org/linux/man-pages/man7/sem_overview.7.html (Дата обращения: 20.10.2025).

4. AMD Ryzen 5 7535HS [Электронный ресурс]. Режим доступа: https://www.amd.com/en/products/processors/laptop/ryzen/7000-series/amd-ryzen-5-7535hs.html (Дата обращения: 20.11.2025).

5. <chrono> — C++ standard library header files [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/cpp/standard-library/chrono?view=msvc-170 (Дата обращения: 20.10.2025).

6. <thread> — C++ standard library header files [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/cpp/standard-library/thread-class?view=msvc-170 (Дата обращения: 20.10.2025).