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 № 5

по курсу «Анализ Алгоритмов»

на тему: «Организация асинхронного взаимодействия потоков вычисления на примере конвейерных вычислений»

Студент ИУ7-51Б (Группа) _____ (Подпись, дата) А. А. Кузин (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Л. Л. Волкова (И. О. Фамилия)

2023 г.

Содержание

ВВЕДЕНИЕ

Параллельные вычисления позволяют увеличить скорость выполнения программы. Конвейерная обработка является приемом, где использование принципов параллельности помогает ускорить обработку данных. Она позволяет на каждой следующей «линии» конвейера использовать данные, полученные с предыдущего этапа.

Целью данной лабораторной работы является описание конвейерной обработки данных.

Необходимо выполнить следующие **задачи**:

1) изучить основы конвейерной обработки;
2) описать используемые алгоритмы обработки матриц;
3) выполнить замеры затрат реализаций алгоритмов по процессорному времени;
4) провести сравнительный анализ алгоритмов.

1 Аналитическая часть

В этом разделе будут рассмотрена информация, касающаяся основ конвейерной обработки данных.

1.1 Конвейерная обработка данных

Конвейер — организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности [1]. Конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Такая обработка данных в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые лентами, и выделении для каждой из них отдельного блока аппаратуры. Конвейеризация позволяет увеличить пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды.

1.2 Описание алгоритмов

В качестве примера для операции, подвергающейся конвейерной обработке, будет обрабатываться матрица. Всего будет использовано три ленты, которые делают следующее:

1) матрица упаковываниется по схеме Чанга и Густавсона;

2) находится определитель матрицы методом миноров;

3) создается дамп текущей заявки.

Вывод

В этом разделе было рассмотрено понятие конвейрной обработки данных, а также выбраны алгоритмы для обработки матрицы на каждой из трех лент конвейера.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы конвейерной и линейной реализаций алгоритмов обработки матриц.

К программе предъявлен ряд функциональных требований:

— наличие интерфейса для выбора действий;

— возможность выбора линейной или конвейерной реализации алгоритма;

2.1 Разработка алгоритмов

На рисунках 2.1–2.5 представлены схемы линейной и конвейерной реализаций алгоритмов обработки матриц, а также схема трех лент для конвейерной обработки матрицы.

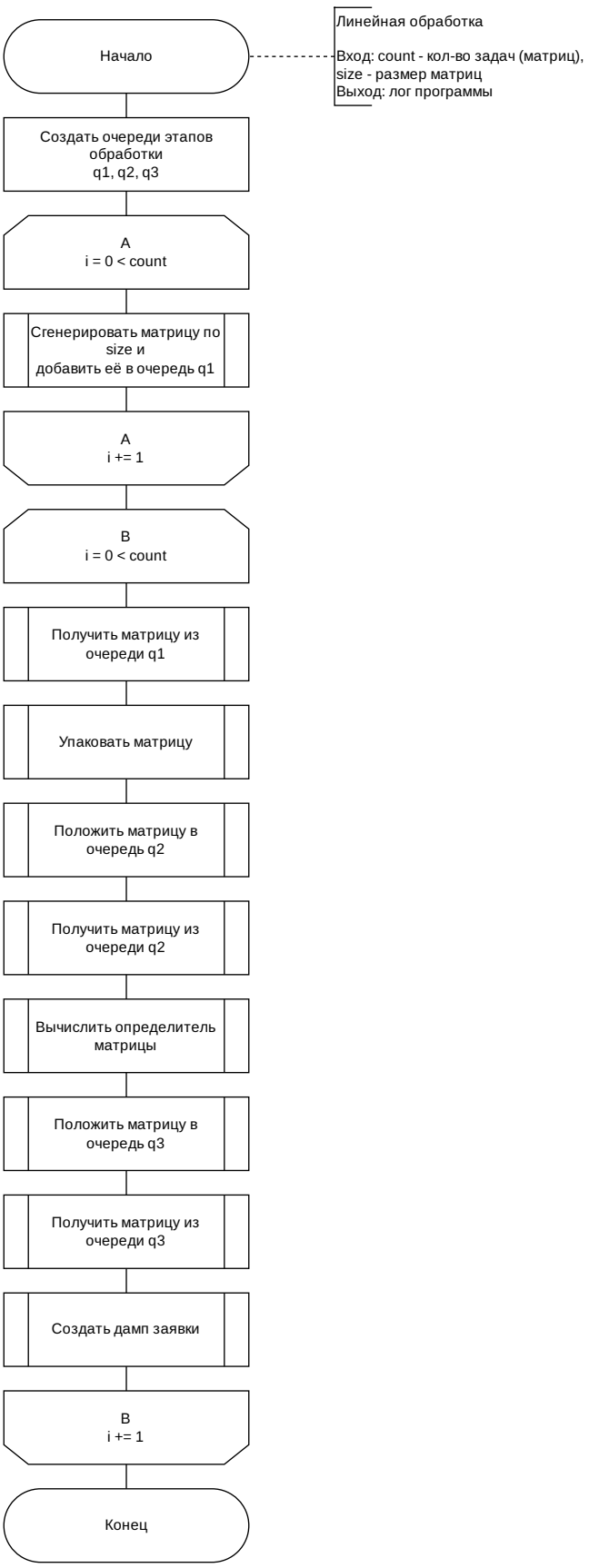

Рисунок 2.1 – Схема алгоритма линейной обработки матрицы

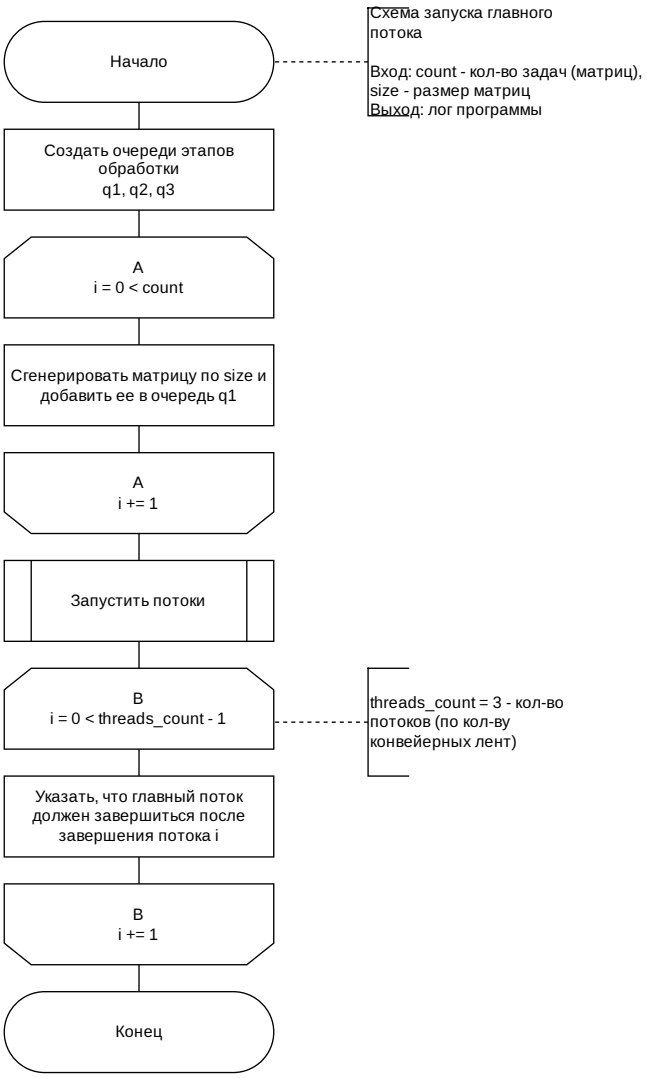

Рисунок 2.2 – Схема алгоритма конвеерной обработки матрицы

Рисунок 2.3 – Схема алгоритма 1 потока обработки матрицы (упаковка матрицы)

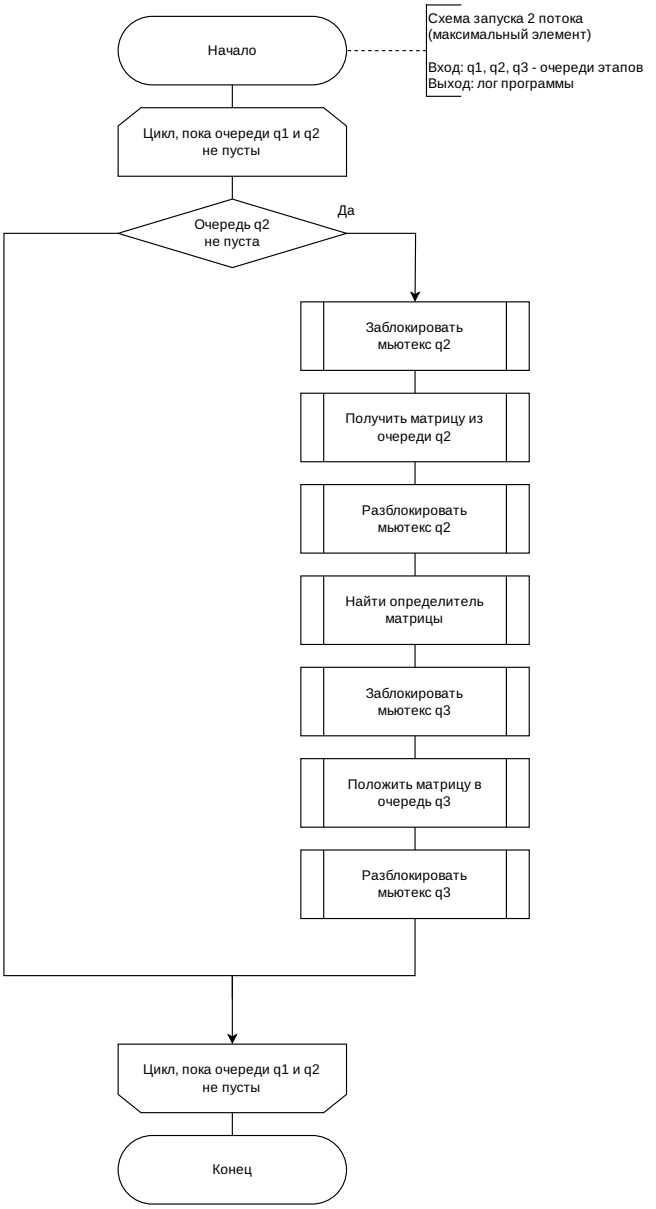

Рисунок 2.4 – Схема алгоритма 2 потока обработки матрицы (вычисление определителя)

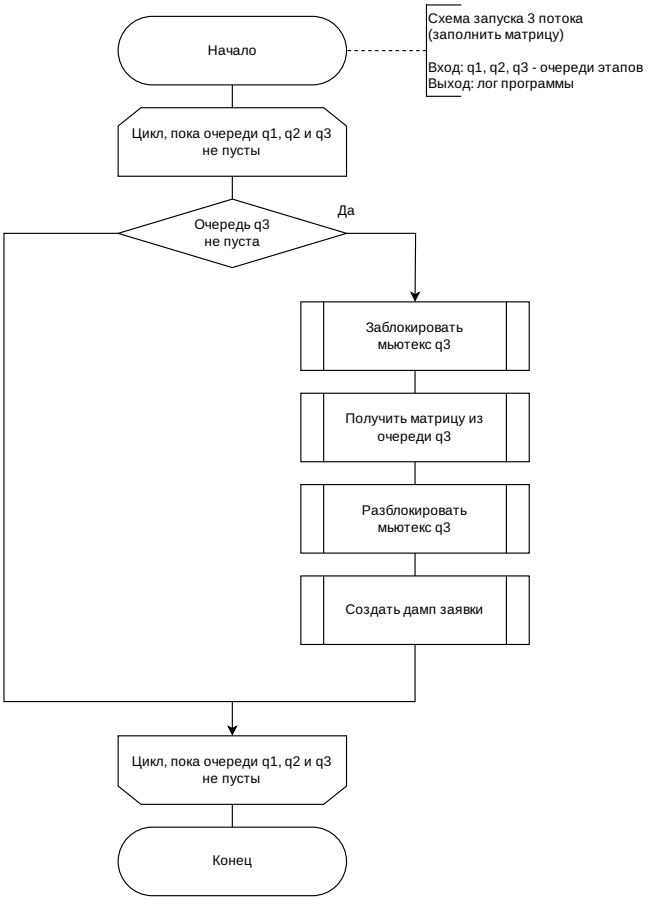

Рисунок 2.5 – Схема алгоритма 3 потока обработки матрицы (создание дампа заявки)

Вывод

В данном разделе были построены схемы алгоритмов требуемых методов обработки матриц (конвейерного и линейного).

3 Технологическая часть

В данном разделе будут приведены средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [2]. Для замеров времени выбрана библиотека `<ctime>` [3], позволяющая производить замеры процессорного времени.

В качестве среды для написания кода был выбран *Visual Studio Code* за счет того, что она предоставляет функционал для проектирования, разработки и отладки ПО.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — точка входа программы, пользовательское меню;

— `general` — модуль с основными конвеерными операциями;

— `csr_matrix` — модуль с реализациями алгоритмов работы над матрицами;

— `measurement` — модуль с реализацией функции подсчета затрачиваемого времени.

3.3 Реализация алгоритмов

Далее будут представлены реализация для линейного и конвейерного алгоритмов обработки матриц. Также представлена реализация запуска 1, 2 и 3 потоков.

Листинг 3.1 – Реализация конвеерных алгоритмов

```
void logLinear(matrix_t &matrix, int task, int stage, \
    void (*func)(matrix_t &), bool needPrinting) {

    std::chrono::time_point<std::chrono::system_clock>
       startTime, endTime;
    double resultStartTime = currentTime, resultTime = 0;

    startTime = std::chrono::system_clock::now();
    func(matrix);
    endTime = std::chrono::system_clock::now();

    resultTime =
       (std::chrono::duration_cast<std::chrono::nanoseconds>
            (endTime - startTime).count()) / 1e9;

    currentTime = resultStartTime + resultTime;

    if (needPrinting)
        printf("Задача: %3d, Этап: %3d, Начало: %.6f, Конец:
           %.6f, Длительность: %.6f\n",
            task, stage, resultStartTime, resultStartTime +
               resultTime, resultTime);
}

void logConway(matrix_t &matrix, int task, int stage, \
    void (*func)(matrix_t &), bool needPrinting) {
    std::chrono::time_point<std::chrono::system_clock>
       startTime, endTime;
    double resultTime = 0;

    startTime = std::chrono::system_clock::now();
    func(matrix);
    endTime = std::chrono::system_clock::now();

```

```
30      resultTime =
           (std::chrono::duration_cast<std::chrono::nanoseconds>
31              (endTime - startTime).count()) / 1e9;
32
33      double resultStartTime;
34
35      if (stage == 1) {
36          resultStartTime = t1[task - 1];
37          t1[task] = resultStartTime + resultTime;
38          t2[task - 1] = t1[task];
39      } else if (stage == 2) {
40          resultStartTime = t2[task - 1];
41          t2[task] = resultStartTime + resultTime;
42          t3[task - 1] = t2[task];
43      } else if (stage == 3) {
44          resultStartTime = t3[task - 1];
45      }
46
47      currentTime = resultStartTime + resultTime;
48      if (needPrinting)
49          printf("Задача: %3d, Этап: %3d, Начало: %.6f, Конец:
               %.6f, Длительность: %.6f\n",
50              task, stage, resultStartTime, resultStartTime +
                   resultTime, resultTime);
51
52  }
53
54
55  void stage1Linear(matrix_t &matrix, int task, bool needPrinting)
       {
56      logLinear(matrix, task, 1, convertToCSRNew, needPrinting);
57  }
58
59  void stage2Linear(matrix_t &matrix, int task, bool needPrinting)
       {
60      logLinear(matrix, task, 2, calcDetermNew, needPrinting);
61  }
62
63
64  void stage3Linear(matrix_t &matrix, int task, bool needPrinting)
       {
```

```
65      logLinear(matrix, task, 3, makeDumpNew, needPrinting);
66  }
67
68  void parseLinear(int count, size_t size, bool needPrinting) {
69      currentTime = 0;
70      std::queue<matrix_t> q1;
71      std::queue<matrix_t> q2;
72      std::queue<matrix_t> q3;
73
74      queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
75
76      for (int i = 0; i < count; i++) {
77          matrix_t res = generateMatrix(size);
78
79          queues.q1.push(res);
80      }
81
82      for (int i = 0; i < count; i++) {
83          matrix_t matrix = queues.q1.front();
84          stage1Linear(matrix, i + 1, needPrinting);
85          queues.q1.pop();
86          queues.q2.push(matrix);
87
88          matrix = queues.q2.front();
89          stage2Linear(matrix, i + 1, needPrinting);
90          queues.q2.pop();
91          queues.q3.push(matrix);
92
93          matrix = queues.q3.front();
94          stage3Linear(matrix, i + 1, needPrinting);
95          queues.q3.pop();
96      }
97  }
98
99
100 void stage1Parallel(std::queue<matrix_t> &q1,
    std::queue<matrix_t> &q2, \
101     std::queue<matrix_t> &q3, bool needPrinting) {
102
103     int task = 1;
104     std::mutex m;
```

```
105     while(!q1.empty()) {
106         m.lock();
107         matrix_t matrix = q1.front();
108         m.unlock();
109 
110         logConway(matrix, task++, 1, convertToCSRNew,
              needPrinting);
111 
112         m.lock();
113         q2.push(matrix);
114         q1.pop();
115         m.unlock();
116     }
117 }
118 
119 
120 void stage2Parallel(std::queue<matrix_t> &q1,
    std::queue<matrix_t> &q2, \
121     std::queue<matrix_t> &q3, bool needPrinting) {
122 
123     int task = 1;
124     std::mutex m;
125     do {
126         m.lock();
127         bool is_q2empty = q2.empty();
128         m.unlock();
129 
130         if (!is_q2empty) {
131             m.lock();
132             matrix_t matrix = q2.front();
133             m.unlock();
134 
135             logConway(matrix, task++, 2, calcDetermNew,
                  needPrinting);
136 
137             m.lock();
138             q3.push(matrix);
139             q2.pop();
140             m.unlock();
141         }
142     } while (!q1.empty() || !q2.empty());
```

```
143 }
144
145
146 void stage3Parallel(std::queue<matrix_t> &q1,
    std::queue<matrix_t> &q2, \
147     std::queue<matrix_t> &q3, bool needPrinting) {
148
149     int task = 1;
150     std::mutex m;
151
152     do {
153         m.lock();
154         bool is_q3empty = q3.empty();
155         m.unlock();
156
157         if (!is_q3empty) {
158             m.lock();
159             matrix_t matrix = q3.front();
160             m.unlock();
161
162             logConway(matrix, task++, 3, makeDumpNew,
                needPrinting);
163
164             m.lock();
165             q3.pop();
166             m.unlock();
167         }
168     } while (!q1.empty() || !q2.empty() || !q3.empty());
169 }
170
171
172 void parseParallel(int count, size_t size, bool needPrinting) {
173     t1.resize(count + 1);
174     t2.resize(count + 1);
175     t3.resize(count + 1);
176
177     for (int i = 0; i < count + 1; i++) {
178         t1[i] = 0;
179         t2[i] = 0;
180         t3[i] = 0;
181     }
```

```
182
183     std::queue<matrix_t> q1;
184     std::queue<matrix_t> q2;
185     std::queue<matrix_t> q3;
186
187     queues_t queues = {.q1 = q1, .q2 = q2, .q3 = q3};
188
189
190     for (int i = 0; i < count; i++) {
191         matrix_t res = generateMatrix(size);
192         q1.push(res);
193     }
194
195     std::thread threads[THREADS];
196
197     threads[0] = std::thread(stage1Parallel, std::ref(q1),
          std::ref(q2), std::ref(q3), needPrinting);
198     threads[1] = std::thread(stage2Parallel, std::ref(q1),
          std::ref(q2), std::ref(q3), needPrinting);
199     threads[2] = std::thread(stage3Parallel, std::ref(q1),
          std::ref(q2), std::ref(q3), needPrinting);
200
201     for (int i = 0; i < THREADS; i++)
202         threads[i].join();
203 }
```

3.4 Функциональные тесты

В данном разделе будут представлены функциональные тесты, проверяющие работу алгоритмов сортировок.

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировок.

Таблица 3.1 – Тестирование функций

Алгоритм	**Кол-во матриц**	**Размер матриц**	**Результат**
Конвейерная	-1	10	Ошибка
Конвейерная	10	-1	Ошибка
Линейная	10	5	Вывод результата
Конвейерная	10	10	Вывод результата

Вывод

Были выбраны язык программирования и среда разработки, приведены сведения о модулях программы, листинги алгоритма, проведено функциональное тестирование.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Intel Core i7 9750H 2.6 ГГц;

— Оперативная память: 16 ГБ;

— Операционная система: Kubuntu 22.04.3 LTS x86_64 Kernel: 6.2.0-36-generic

Во время проведения измерений времени ноутбук был подключен к сети электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 показан пример работы с программой.

```
po4ti@Po4ti-PC:~/Документы/GitHub/Lab_05_AA$ ./app.exe
1 - Линейная очередь
2 - Конвеерная очередь
3 - Выполнить замеры
0 - Выход
1

Размер: 5
Количество: 5
Задача:   1, Этап:   1, Начало: 0.000000, Конец: 0.000008, Длительность: 0.000008
Задача:   1, Этап:   2, Начало: 0.000008, Конец: 0.000248, Длительность: 0.000240
Задача:   1, Этап:   3, Начало: 0.000248, Конец: 0.000277, Длительность: 0.000029
Задача:   2, Этап:   1, Начало: 0.000277, Конец: 0.000284, Длительность: 0.000007
Задача:   2, Этап:   2, Начало: 0.000284, Конец: 0.000514, Длительность: 0.000230
Задача:   2, Этап:   3, Начало: 0.000514, Конец: 0.000541, Длительность: 0.000027
Задача:   3, Этап:   1, Начало: 0.000541, Конец: 0.000547, Длительность: 0.000007
Задача:   3, Этап:   2, Начало: 0.000547, Конец: 0.000779, Длительность: 0.000231
Задача:   3, Этап:   3, Начало: 0.000779, Конец: 0.000805, Длительность: 0.000026
Задача:   4, Этап:   1, Начало: 0.000805, Конец: 0.000811, Длительность: 0.000006
Задача:   4, Этап:   2, Начало: 0.000811, Конец: 0.001040, Длительность: 0.000229
Задача:   4, Этап:   3, Начало: 0.001040, Конец: 0.001067, Длительность: 0.000027
Задача:   5, Этап:   1, Начало: 0.001067, Конец: 0.001073, Длительность: 0.000006
Задача:   5, Этап:   2, Начало: 0.001073, Конец: 0.001305, Длительность: 0.000231
Задача:   5, Этап:   3, Начало: 0.001305, Конец: 0.001332, Длительность: 0.000027
1 - Линейная очередь
2 - Конвеерная очередь
3 - Выполнить замеры
0 - Выход
```

Рисунок 4.1 – Демонстрация работы программы.

4.3 Временные характеристики

Исследование временных характеристик реализованных алгоритмов производилось на массивах размером 1 – 8 с шагом 1.

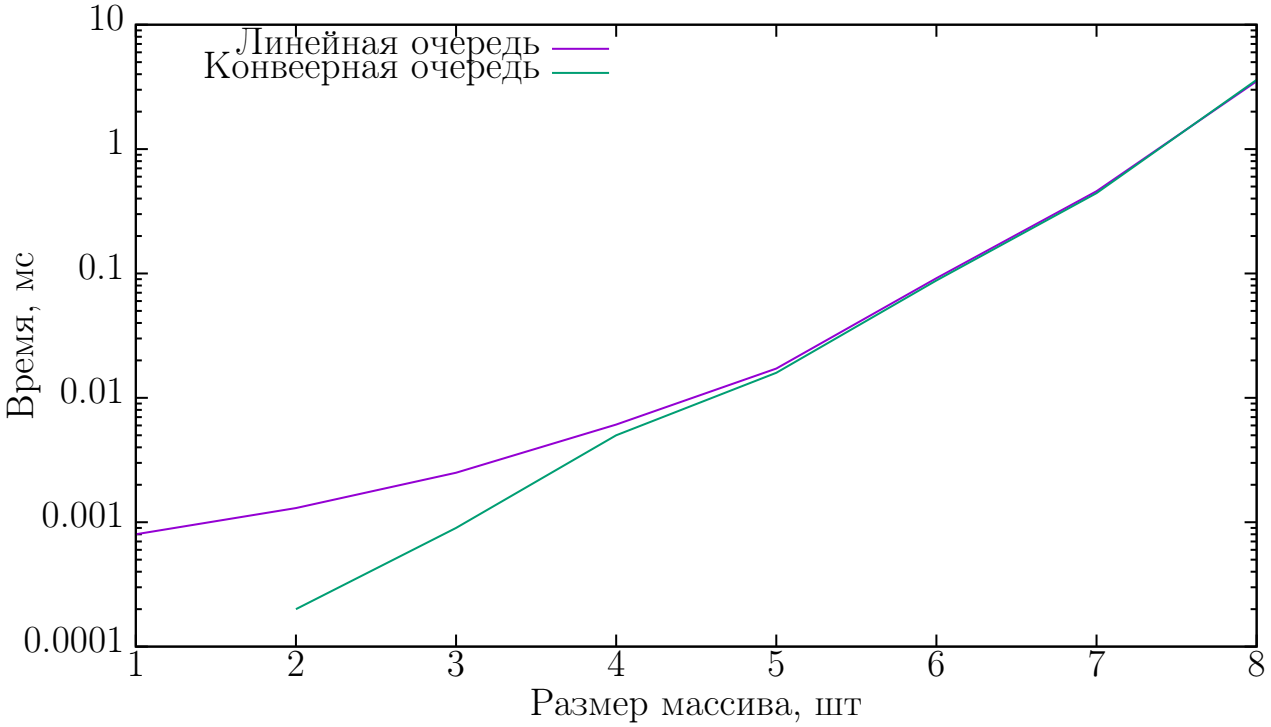

Рисунок 4.2 – Результат измерений времени работы (в мс) алгоритмов при разных размерах матриц

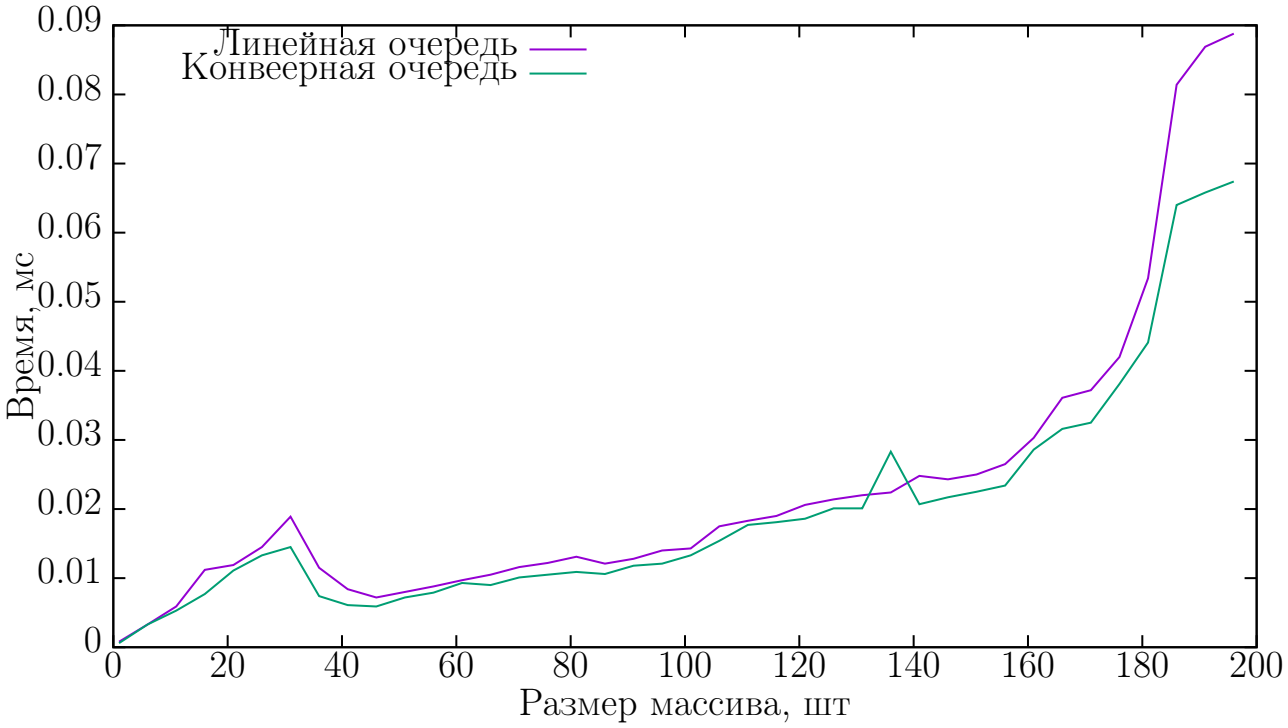

Рисунок 4.3 – Результат измерений времени работы (в мс) алгоритмов при разном кол-ве матриц

4.4 Вывод

При разных размерах матриц, в связи с разной сложностью заявок, при больших размерах матриц возникает проблема горлышка бутылки. Из-за экспоненциального роста второй заявки разница между конвеерным и линейным алгоритмом стремится к нулю.

При одинаковых размерах матриц, конвеерная обработка заявок занимает меньше времени, чем линейная обработка.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы поставленная ранее цель была достигнута: были изучены принципы конвейерной обработки данных на примере работы с матрицами.

В ходе выполнения лабораторной работы были решены следующие задачи:

1) изучены основы конвейерной обработки;
2) описаны используемые алгоритмы обработки матриц;
3) выполнены замеры затрат реализаций алгоритмов по процессорному времени;
4) проведены сравнительный анализ алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конвейерная обработка данных [Электронный ресурс]. — — Режим доступа: hhttps://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh.
2. Документация по Microsoft C++ [Электронный ресурс]. — — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017 (дата обращения: 25.09.2022).
3. Standard library header <ctime> [Электронный ресурс]. — — Режим доступа: https://en.cppreference.com/w/cpp/header/ctime.