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 №5

Название «Конвейеры»

Дисциплина «Анализ алгоритмов»

Студент ИУ7-55Б _____ Бугаенко А.П.
(подпись, дата) (Фамилия И.О.)

Преподователь _____ Волкова Л.Л.
(подпись, дата) (Фамилия И.О.)

Москва, 2021

Содержание

Введение

Конвейер - это способ организации вычислений,, позволяющий увеличить производительность процессов за счёт увеличения числа инструкций выполняемых в единицу времени. В данной лабораторной работе мы рассмотрим конвейерную организацию вычислений на примере трёх операциях на массиве, которые будут обрабатываться в одном случае отдельно, в другом на трёх конвейерных лентах.

Целью данной лабораторной работы является изучение изучение и реализация конвейера на примере трёх операций над массивом. Для того, чтобы достичь поставленной цели, нам необходимо выполнить следующие задачи:

1) провести анализ алгоритмов и теоретические основы конвейерной обработки данных;

2) описать используемые структуры данных;

3) привести схемы рассматриваемого алгоритма для конвейерной и неконвейерной реализации;

4) программно реализовать описанные выше алгоритмы;

5) провести сравнительный анализ каждого алгоритма по затрачиваемому в процессе работы времени.

1 Аналитический раздел

В данном разделе будут рассмотрены теоретически основы работы конвейера при применении его к трём алгоритмам: алгоритму поиска максимума, алгоритму поиска минимума, алгоритму поиска суммы всех элементов.

1.1 Алгоритм поиска максимума

Алгоритм поиска максимума будет реализован в виде простейшего алгоритма поиска максимума с перебором всех элементов массива. То есть каждый элемент сравнивается с переменной, хранящей значение локального максимума и если эта переменная больше, чем максимум, то её значение записывается в переменную максимума.

1.2 Алгоритм поиска минимума

Алгоритм поиска минимума будет реализован в виде простейшего алгоритма поиска минимума с перебором всех элементов массива. То есть каждый элемент сравнивается с переменной, хранящей значение локального минимума и если эта переменная больше, чем минимум, то её значение записывается в переменную минимума.

1.3 Алгоритм поиска суммы

Алгоритм поиска минимума будет реализован в виде простейшего алгоритма поиска минимума с перебором всех элементов массива. То есть каждый элемент добавляется к переменной, хранящей значение суммы всех элементов.

1.4 Конвейерная обработка данных

При конвейерной обработке данных используется принцип конвейерного производства, заключающийся в том, что некая крупная задача разделяется на множество коротких операций, которое делается параллельно. Данный способ организации решения каких-либо задач имеет наибольшее преимущество при потоковом решении множества однотипных подзадач, которые разбиваются так, что каждая операция может быть выполнена на одной из стадий конвейера, в результате мы получаем систему, в которой параллельно выполняется сразу несколько подзадач нескольких разных задач. В данной лабораторной работе мы выделили три задачи, которые последовательно обрабатываются на конвейерной ленте.

1.5 Вывод

В данном разделе были рассмотрены основные теоретические сведения об алгоритмах поиска максимума, минимума и суммы элементов в массиве. В результате были сделаны выводы о том, что на вход алгоритму массив целых чисел произвольных размеров, на выходе программа максимум, минимум и сумму всех элементов в массиве. Алгоритмы работают на массивах

с размерностями от 0 до физически возможного предела для используемой машины. В качестве критерия для сравнения эффективности алгоритмов будет использоваться время работы на матрицах различного размера. Также была рассмотрена конвейерная обработка данных и сделаны выводы о том, как именно будет организован данный процесс.

2 Конструкторский раздел

В данном разделе будут рассмотрены схемы, структуры данных, способы тестирования, описания памяти для следующих алгоритмов:

1) алгоритм поиска максимума;
2) алгоритм поиска минимума.
3) алгоритм поиска суммы всех элементов;
4) алгоритм решения задачи без конвейера;
5) алгоритм решения задачи с конвейером.

2.1 Тестирование алгоритмов

Описание классов эквивалентности:

1) проверка работы на общем случае.

Описание тестов:

1) тест на общем случае - на вход подаётся массив размерама n, возвращаемый результат сравнивается с заранее известным правильным результатом.

2.2 Алгоритм поиска максимума

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов массива, размера массива;
2) array, массив целых чисел - используется для хранения серии целых чисел.

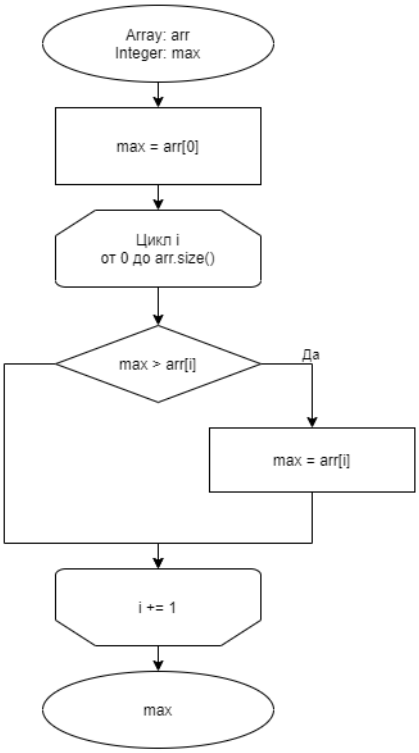

Рисунок 2.1 — Схема алгоритма поиска максимума

2.3 Алгоритм поиска минимума

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов массива, размера массива;
2) array, массив целых чисел - используется для хранения серии целых чисел.

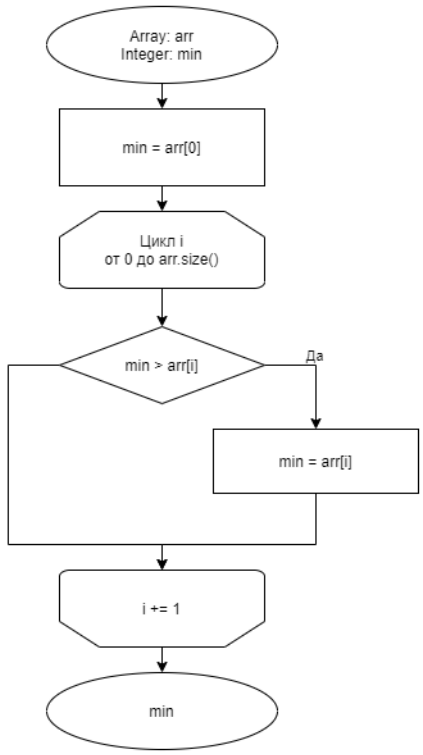

Рисунок 2.2 — Схема алгоритма поиска минимума

2.4 Алгоритм поиска суммы всех элементов

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов массива, размера массива;
2) array, массив целых чисел - используется для хранения серии целых чисел.

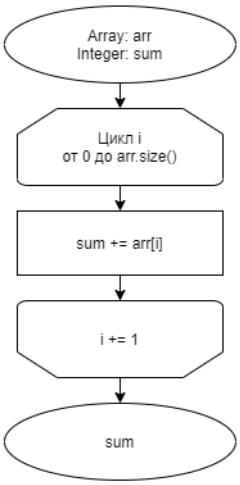

Рисунок 2.3 — Схема алгоритма поиска суммы всех элементов

2.5 Алгоритм решения задачи без конвейера

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов массива, размера массива;
2) array, массив целых чисел - используется для хранения серии целых чисел.

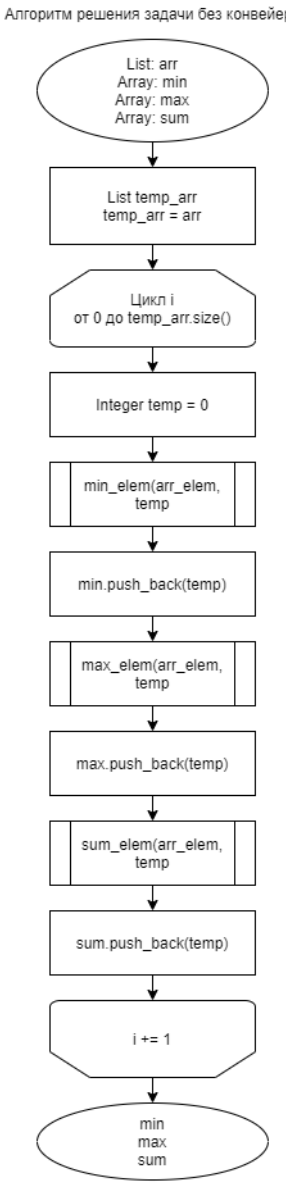

Рисунок 2.4 — Схема алгоритма решения задачи без конвейера

2.6 Алгоритм решения задачи с конвейером

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов массива, размера массива;
2) array, массив целых чисел - используется для хранения серии целых чисел;
3) matrix, массив массивов целых числел - представление матрицы в программе;
4) vector, вектор - вид связного списка, позволяющий осуществять доступ к элементам по индексу.

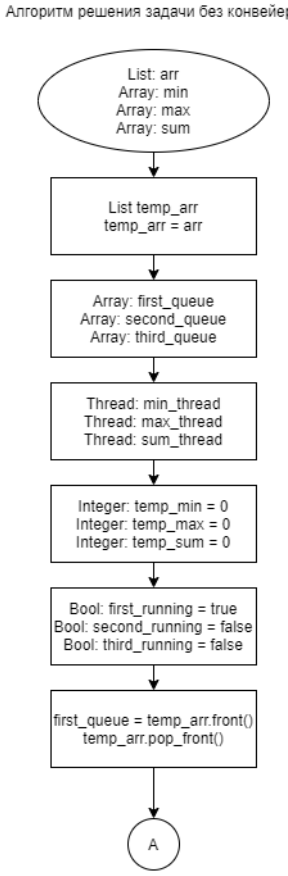

Рисунок 2.5 — Схема алгоритма решения задачи без конвейера часть 1

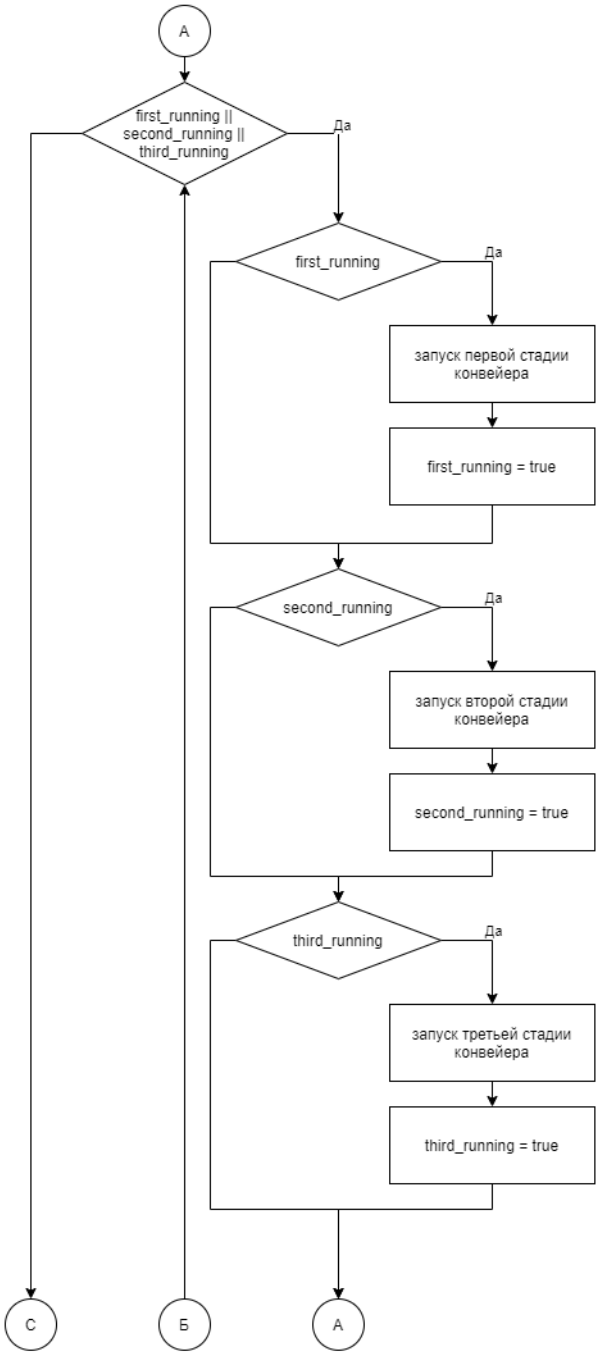

Рисунок 2.6 — Схема алгоритма решения задачи без конвейера часть 2

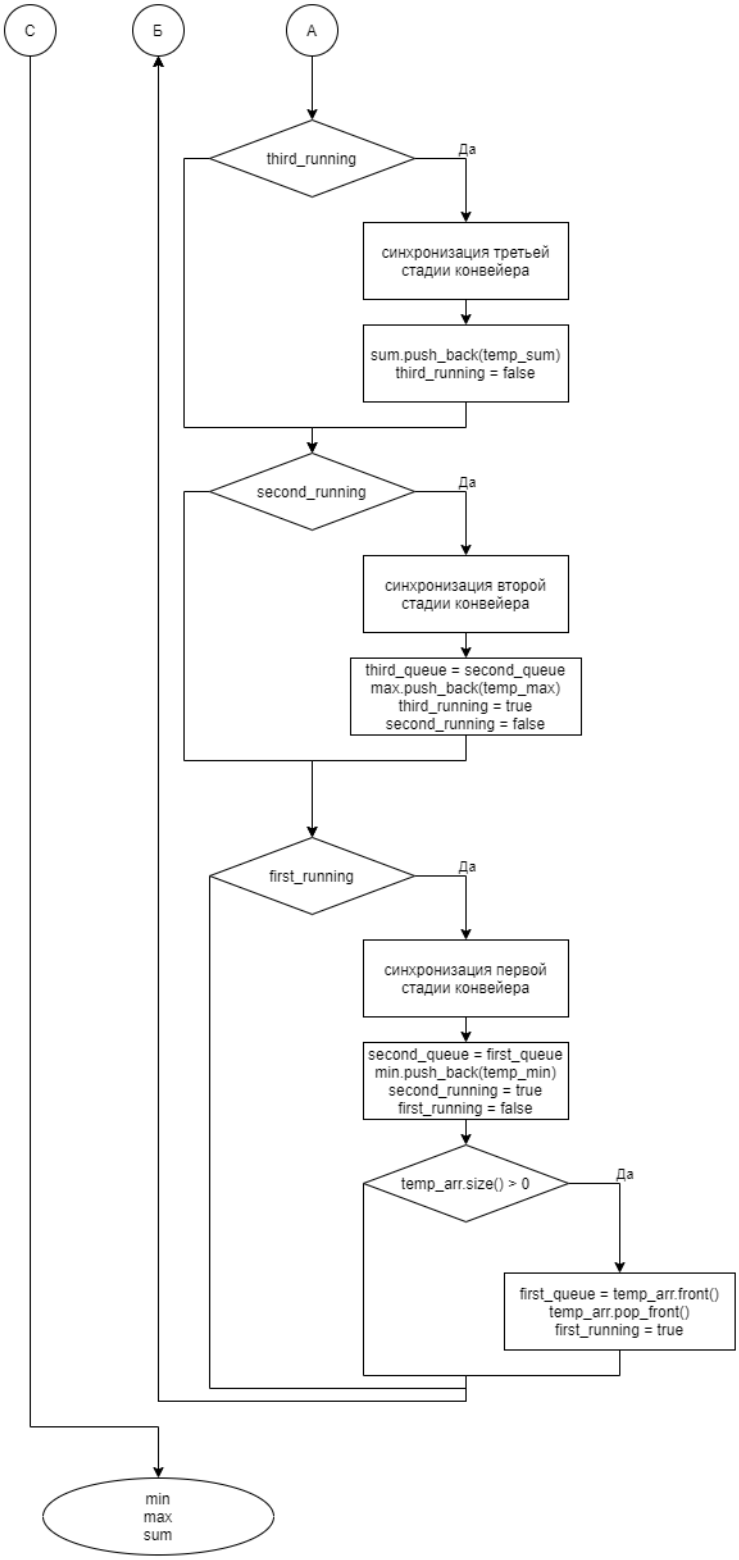

Рисунок 2.7 — Схема алгоритма решения задачи без конвейера часть 3

2.7 Функциональная схема ПО

На изображении ниже представлена функиональная схема разрабатываемого ПО. На вход подаётся массив, заполненный целыми числами и при помощи алгоритмов, реализованных на языке C++ мы получаем в результате работы минимум, максимум и сумму всех элементов.

Рисунок 2.8 — IDEF0 диаграмма разрабатываемой программы

2.8 Вывод

В данном разделе были рассмотрены схемы алгоритмов для поиска максимума, минимума, суммы всех элементов и алгоритмы решения задачи без и с конвейером. Были определены тесты для каждого алгоритма, были описаны типы и структуры данных, использующихся в алгоритмах. Также была приведена функциональная схема разрабатываемого ПО.

3 Технологический раздел

В данном разделе будут рассмотрены подробности реализации описаных выше алгоритмов. Также будут обоснованы выбор языка программирования для реализации, выбор библиотек для проведения экспериментов и представлены важные фрагменты кода написанной в рамках работы программы.

3.1 Выбор языка программирования

В качестве языка программирования для реализации данной лабораторной работы использовался язык программирования C++ поскольку в данном языке есть стандартная библиотека thread в которой реализованы основные методы для работы с потоками. В качестве среды разработки использовалась Microsoft Visual Studio 2019 по причине того, что данная среда имеет встроенные средства отладки и анализа программ.

3.2 Сведения о модулях программы

Реализованное ПО состоит из трёх модулей:

1) conveyor - в данном модуле реализованы алгоритмы;
2) lab_5 - основной файл программы, где находится точка входа;
3) tests - реализация тестов алгоритма;
4) time - реализация замеров времени работы программы.

3.3 Реализация алгоритмов

Листинг 3.1 — Реализация алгоритма поиска минимума

```
1 void min_elem(std::vector<int> arr, int &min)
2 {
3     min = arr[0];
4     for (int i = 0; i < arr.size(); i++)
5     {
6         if (min > arr[i])
7             min = arr[i];
8     }
9 }
```

Листинг 3.2 — Реализация алгоритма поиска максимума

```
1 void max_elem(std::vector<int> arr, int& max)
2 {
3     max = arr[0];
4     for (int i = 0; i < arr.size(); i++)
5     {
6         if (max < arr[i])
```

```
7               max = arr[i];
8       }
9   }
```

Листинг 3.3 — Реализация алгоритма поиска суммы всех элементов

```
1   void sum_elem(std::vector<int> arr, int& sum)
2   {
3       sum = 0;
4       for (int i = 0; i < arr.size(); i++)
5       {
6           sum += arr[i];
7       }
8   }
```

Листинг 3.4 — Реализация алгоритма без конвейера

```
1   void conveyor::count_arr(std::list<std::vector<int>>& arr, std::vector<int>& min,
        std::vector<int>& max, std::vector<int>& sum)
2   {
3       std::list<std::vector<int>> temp_arr;
4       temp_arr = arr;
5       for (auto& arr_elem : temp_arr)
6       {
7           int temp = 0;
8           min_elem(arr_elem, temp);
9           min.push_back(temp);
10          max_elem(arr_elem, temp);
11          max.push_back(temp);
12          sum_elem(arr_elem, temp);
13          sum.push_back(temp);
14      }
15  }
```

Листинг 3.5 — Реализация алгоритма с конвейером

```
1   void conveyor::count_arr_conv(std::list<std::vector<int>>& arr, std::vector<int>&
        min, std::vector<int>& max, std::vector<int>& sum)
2   {
3       std::vector<int> first_queue;
4       std::vector<int> second_queue;
5       std::vector<int> third_queue;
6
7       std::list<std::vector<int>> temp_arr;
8       temp_arr = arr;
9
10      std::thread min_thread;
11      std::thread max_thread;
```

```
12      std::thread sum_thread;
13
14      int temp_min = 0;
15      int temp_max = 0;
16      int temp_sum = 0;
17
18      bool first_running = true;
19      bool second_running = false;
20      bool third_running = false;
21
22      first_queue = temp_arr.front();
23      temp_arr.pop_front();
24
25      while (first_running || second_running || third_running)
26      {
27          if (first_running)
28          {
29              min_thread = std::thread(min_elem, first_queue, std::ref(temp_min));
30              first_running = true;
31          }
32
33          if (second_running)
34          {
35              max_thread = std::thread(max_elem, second_queue, std::ref(temp_max));
36              second_running = true;
37          }
38
39          if (third_running)
40          {
41              sum_thread = std::thread(sum_elem, third_queue, std::ref(temp_sum));
42              third_running = true;
43          }
44
45          if (third_running)
46          {
47              sum_thread.join();
48              sum.push_back(temp_sum);
49              third_running = false;
50          }
51
52          if (second_running)
53          {
54              max_thread.join();
55              third_queue = second_queue;
56              max.push_back(temp_max);
57              third_running = true;
58              second_running = false;
```

```
59          }
60
61          if (first_running)
62          {
63              min_thread.join();
64              second_queue = first_queue;
65              min.push_back(temp_min);
66              second_running = true;
67              first_running = false;
68              if (temp_arr.size() > 0)
69              {
70                  first_queue = temp_arr.front();
71                  temp_arr.pop_front();
72                  first_running = true;
73              }
74          }
75      }
76  }
```

3.4 Реализация тестирования алгоритмов

Для тестирования алгоритмов было реализованы следующие тесты:

1) тест на массивах, заполненных заранее известными значениями;

Листинг 3.6 — Реализация тестов

```
1  void test()
2  {
3      conveyor conv;
4      std::list<std::vector<int>> arrs = { {1, 3, 4, -5}, {2, 4, 1, 4}, {1, 5, 2, 1},
           {1, 3, 4, 5} };
5      std::vector<int> min;
6      std::vector<int> max;
7      std::vector<int> sum;
8
9      std::vector<int> min_true = { -5, 1, 1, 1 };
10     std::vector<int> max_true = { 4, 4, 5, 5 };
11     std::vector<int> sum_true = { 3, 11, 9, 13 };
12
13     conv.count_arr(arrs, min, max, sum);
14
15     std::cout << "Without conveyor result:" << std::endl;
16     std::cout << "Minimun correct: " << (min == min_true) << std::endl;
17     std::cout << "Maximun correct: " << (max == max_true) << std::endl;
18     std::cout << "Sum correct: " << (sum == sum_true) << std::endl;
19
```

```
20      max.clear();
21      min.clear();
22      sum.clear();
23      conv.count_arr_conv(arrs, min, max, sum);
24
25      std::cout << "With conveyor result:" << std::endl;
26      std::cout << "Minimun correct: " << (min == min_true) << std::endl;
27      std::cout << "Maximun correct: " << (max == max_true) << std::endl;
28      std::cout << "Sum correct: " << (sum == sum_true) << std::endl;
29  }
```

3.5 Вывод

В данной разделе были представлены реализации алгоритма свёртки и параллельного алгоритма свёртки и показана реализация модуля тестирования реализованных алгоритмов.

4 Экспериментальный раздел

В данном разделе будут измерены временные характеристики алгоритмов свёртки и сделаны выводы об эффективности применения параллельного программирования для улучшения временных показателей данного алгоритма.

4.1 Технические характеристики

— Операционная система - Windows 10, 64-bit;
— Оперативная память - 16 GiB;
— Процессор - Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz, 6 ядер, 12 потоков.

4.2 Результаты экспериментов

Таблица 4.1 — Время работы алгоритмов часть 1

Размерность	Без конвейера	С конвейером
10000	14410730	36757790
20000	29048790	65201110
30000	41677490	66076070
40000	53177230	71010880
50000	65735330	75928700
60000	79101550	86291220
70000	92885460	99540600
80000	105098080	107917840
90000	118854190	110939920
100000	135586310	131559980
110000	146946320	140116700
120000	160452060	140567430
130000	175246470	147778200
140000	192886470	164722970
150000	222497510	179168950
160000	230255670	182251830
170000	240874730	195043000
180000	247061340	201011500
190000	257129200	207278030
200000	268234690	215100080
210000	284637150	229495320
220000	297554680	233842900
230000	311475670	244595200
240000	389308390	293369750
250000	351293580	272306600

Таблица 4.2 — Время работы алгоритмов часть 2

Размерность	Без конвейера	С конвейером
260000	356309320	274189950
270000	372085030	279380220
280000	381061510	286817280
290000	391780530	290586790
300000	407649700	302525900
310000	416702090	309778420
320000	431093340	315700850
330000	451243070	334894650
340000	464012590	342740610
350000	471855680	344893820
360000	489654290	362364270
370000	494999030	362130930
380000	512794090	379268490
390000	533304620	391316500
400000	545626940	402669500
410000	554232820	403925530
420000	564171810	409611750
430000	581163270	422192700
440000	596754290	431110550
450000	602919380	433952520
460000	630570110	453187860
470000	634635850	459071680
480000	656107760	481584640
490000	661004140	479517890
500000	702121310	504461530

Таблица 4.3 — Лог работы программы

Номер конвейера	Номер заявки	Время начала	Время конца
1	1	2900	20141800
2	1	30015400	35058300
1	2	29820600	42582600
3	1	51763000	59150500
2	2	49758300	65423100
1	3	49545000	71573200
3	2	82941800	98629500
2	3	79447700	104511900
1	4	79232900	108953300
3	3	111954600	114019500
2	4	111278500	115988000
1	5	111163000	118624000
3	4	122693000	124279400
2	5	120866100	126324900
1	6	120666900	127200300
3	5	128527000	130745200
2	6	127979500	131521400
1	7	127898000	132199000
3	6	133430100	135606000
2	7	132917000	136353900
1	8	132869900	138477000
3	7	140910700	143018500
2	8	140807800	144370600
3	8	145263500	146321500

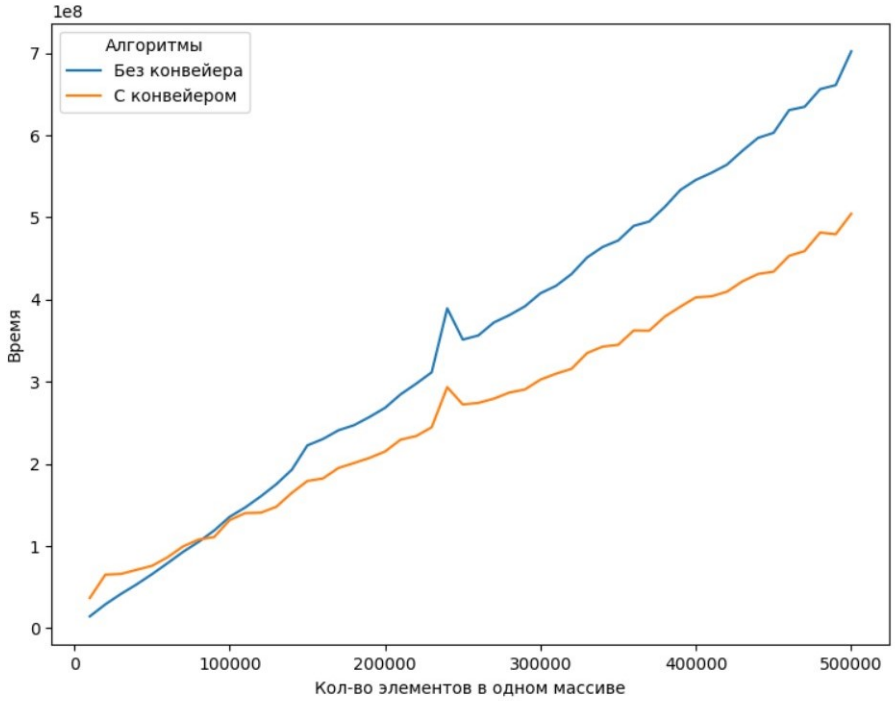

Рисунок 4.1 — График зависимости времени работы от размерности массивов

4.3 Вывод

В результате эксперимента было получено, что на квадратных массивах размерами от 10000 до 500000 использование конвейера позволяет добиться ускорения в 1.39 раз по времени выполнения. В результате можно сделать вывод о том, что использование ковейерного программирования позволяет ускорить обработку больших задач.

Заключение

В процессе выполнения данной лабораторной работы был изучен конвейер на примере алгоритмов обработки массива. Были выполнены анализ алгоритмов и представлены схемы алгоритмов, а также функциональная схема ПО. После чего эти алгоритмы были реализованы при помощи языка C++ в IDE Visual Studio 2019. Помимо этого были произведены эксперименты с целью получить информацию о временной производительности алгоритмов. В результате эксперимента было получено, что на квадратных массивах размерами от 10000 до 500000 использование конвейера позволяет добиться ускорения в 1.39 раз по времени выполнения. В результате можно сделать вывод о том, что использование ковейерного программированния позволяет ускорить обработку больших задач. Целью данной лабораторной работы являлось изучение конвейеров на примере алгоритма обработки массива, что было успешно достигнуто.

Список литературы

[1] Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [электронный ресурс]. Режим доступа: https://www.intuit.ru/studies/courses/4447/983/lecture/14925, свободный (Дата обращения: 1.12.21)

[2] "Документация по языку C++"[электронный ресурс]. Режим доступа: https://docs.microsoft.com/r свободный (Дата обращения 1.12.21)

[3] Процессор Intel® Core™ i7-9750H (12 МБ кэш-памяти, до 4,50 ГГц) [электронный ресурс] https://www.intel.ru/content/www/ru/ru/products/sku/191045/intel-core-i79750h-processor-12m-cache- свободный (Дата обращения: 2.12.21)

[4] Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — БХВ-Петербург, 2002.

[5] <chrono> [электронный ресурс] https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view свободный (Дата обращения: 5.12.21)