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

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

Москва, 2021

Содержание

Введение

Параллельное программирование - способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно). Поскольку ресурсы компьютера ограничены, параллелизация алгоритмов и последующая их реализация в параллельном виде позволит значительно ускорить их выполнение. В данной лабораторной работе мы будем рассматривать параллельное программирование на примере параллелизации алгоритма свёртки.

Целью данной лабораторной работы является изучение изучение и реализация параллельного программирования в рамках решения задачи параллельной и непараллельной реализации алгоритма свёртки. Для того, чтобы достичь поставленной цели, нам необходимо выполнить следующие задачи:

1) провести анализ алгоритма свёртки;

2) описать используемые структуры данных;

3) привести схемы рассматриваемого алгоритма для параллельной и непараллельной реализации;

4) программно реализовать данные выше алгоритмы;

5) провести сравнительный анализ каждого алгоритма по затрачиваемому в процессе работы времени в зависимости от количества потоков.

1 Аналитический раздел

В данном разделе будут рассмотрены теоретически основы работы алгоритма свёртки и основные принципы параллельного программирования.

1.1 Алгоритм свёртки

Алгоритм свёртки является реализацией операцией свёртки, которая используется в свёрточных нейронных сетях. Данный вид нейросетей обрабатывает изображения, которые подаются на вход алгоритму в виде матрицы пикселей. Операция свёртки позволяет находить и максимизировать признаки по которым нейросеть осуществляет какие-либо действия с алгоритмом.

Алгоритм свёртки работает следующим образом: На вход алгоритма подаётся некая матрица A размера M x N. За тем к каждому элементу матрицы применяется ядро, представляющее из себя матрицу небольших размеров. В нашем случае будет использоваться матрица размером 3 на 3, выглядящая следующим образом:

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \tag{1.1}$$

Для каждой области 3 на 3 в матрице применяется данный фильтр. Новое значение вычисляется путём суммирования произведений наложившихся элементов. Результат записывается в новую матрицу, размерности которой уменьшается на два.

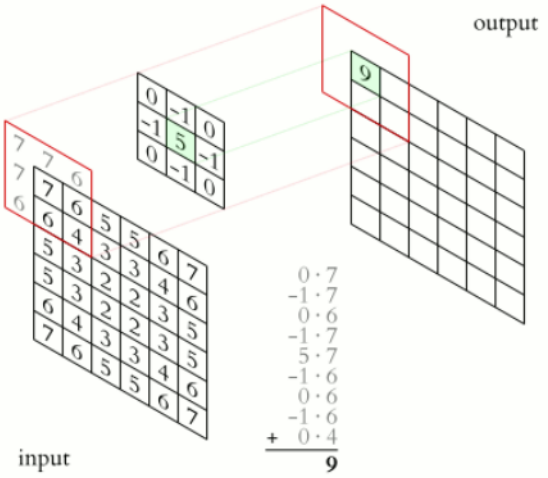

Рисунок 1.1 — Схема работы операции свёртки

1.2 Параллельный алгоритм свёртки

Так как каждая операция применения ядра происходит независимо от друг друга, мы можем выполнять их параллельно. В нашем случае мы будем выполнять их параллельно для каждого ряда матрицы. То есть например если у нас есть два потока, то первый параллельно посчитает чётные ряды, а второй нечётные.

1.3 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти одинаково (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами. Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функционирующее вычислительное устройство. Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью — создание новых параллельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применятся как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.[1]

1.4 Вывод

В данном разделе были рассмотрены основные теоретические сведения об алгоритме свёртки и параллельном алгоритме свёртки. В результате были сделаны выводы о том, что на вход алгоритму подаётся матрица произвольных размеров, на выходе программа возвращает новую матрицу, полученную путём применения фильтра к старой. Алгоритмы работают на матрицах с размерностями от 3 до физически возможного предела для используемой машины. В качестве критерия для сравнения эффективности алгоритмов будет использоваться время работы на матрицах различного размера.

2 Конструкторский раздел

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

1) алгоритм свёртки;
2) параллельный алгоритм свёртки.

2.1 Тестирование алгоритмов

Описание классов эквивалентности:

1) проверка работы на общем случае.

Описание тестов:

1) тест на общем случае - на вход подаётся матрица размерами m на n, выход сравнивается с заранее известным правильным результатом.

2.2 Алгоритм свёртки

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.

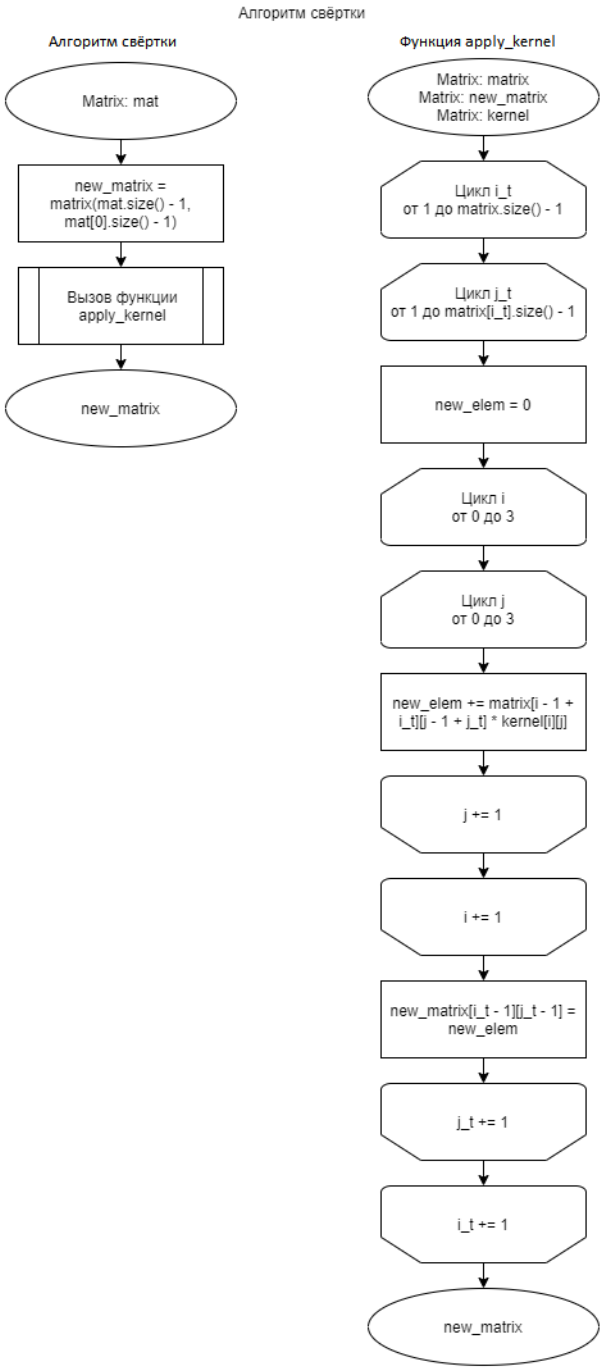

Рисунок 2.1 — Схема алгоритма свёртки

2.3 Параллельный алгоритм свёртки

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.

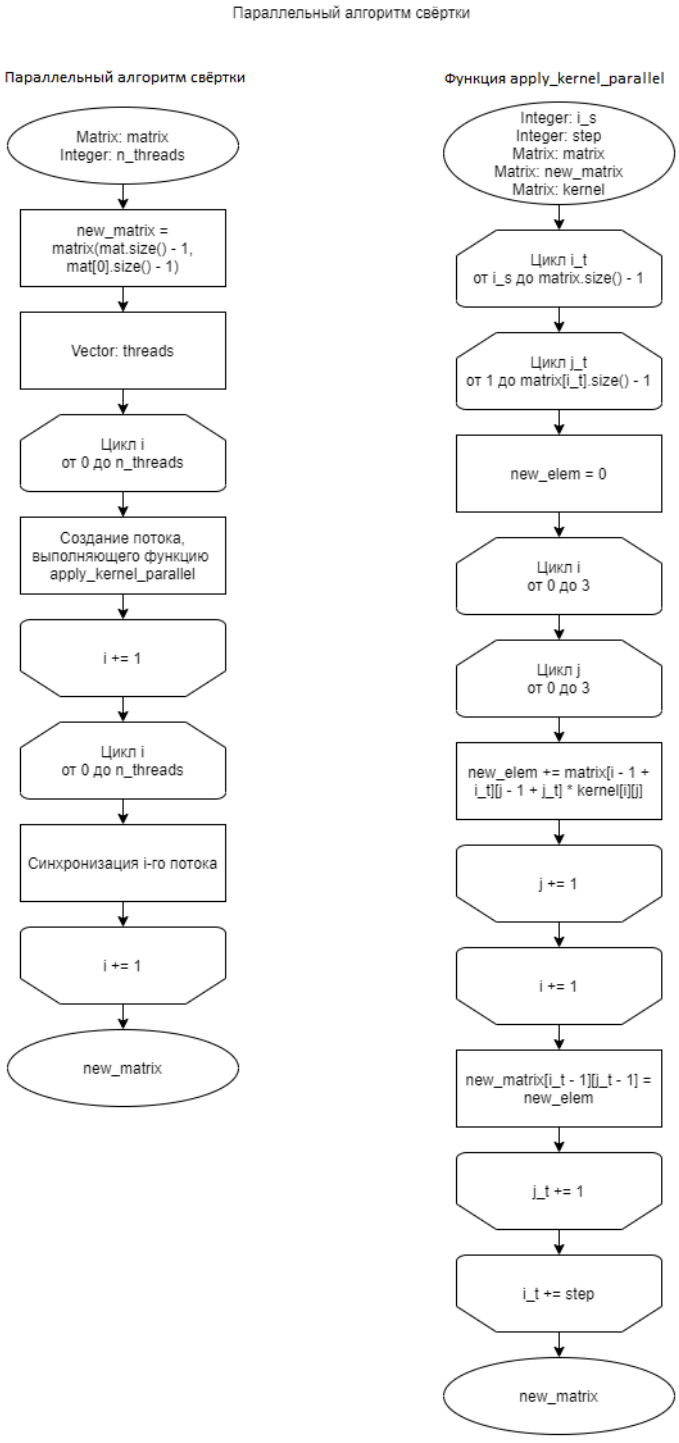

Рисунок 2.2 — Схема параллельного алгоритма свёртки

2.4 Функциональная схема ПО

На изображении ниже представлена функиональная схема разрабатываемого ПО. На вход подаётся матрица, заполненная целыми числами и при помощи алгоритмов, реализованных на языке C++ мы получаем в результате работы новую матрицу, содерждащую в себе результат свёртки.

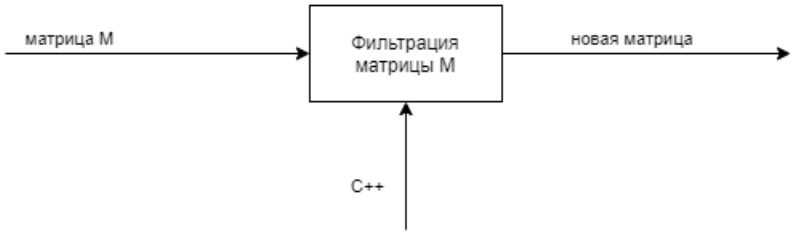

Рисунок 2.3 — IDEF0 диаграмма разрабатываемой программы

2.5 Вывод

В данном разделе были рассмотрены схемы алгоритмов для параллельного не обычного алгоритма свёртки, и 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) Convolutionizer - в данном модуле реализованы алгоритмы свёртки;
2) lab_4 - основной файл программы, где находится точка входа;
3) tests - реализация тестов алгоритма;
4) time - реализация замеров времени работы программы.

3.3 Реализация алгоритма свёртки

Листинг 3.1 — Реализация алгоритма свёртки в виде метода класса Convolutionizer

```
1 std::vector<std::vector<int>>
      Convolutionizer::convolution(std::vector<std::vector<int>> matrix)
2 {
3     std::vector<std::vector<int>> new_matrix(matrix.size() - 2,
          std::vector<int>(matrix[0].size() - 2, 0));
4
5     apply_kernel(matrix, new_matrix);
6     return new_matrix;
7 }
```

Листинг 3.2 — Реализация функции применения ядра к матрице

```
1 void Convolutionizer::apply_kernel(std::vector<std::vector<int>>& matrix,
      std::vector<std::vector<int>>& new_matrix)
2 {
3     for (int i_t = 1; i_t < matrix.size() - 1; i_t++)
4     {
5         for (int j_t = 1; j_t < matrix[i_t].size() - 1; j_t++)
```

```
6           {
7               int new_elem = 0;
8               for (int i = 0; i < 3; i++)
9               {
10                  for (int j = 0; j < 3; j++)
11                  {
12                      new_elem += matrix[i - 1 + i_t][j - 1 + j_t] * kernel[i][j];
13                  }
14              }
15              new_matrix[i_t - 1][j_t - 1] = new_elem;
16          }
17      }
18  }
```

3.4 Реализация параллельного алгоритма свёртки

Листинг 3.3 — Реализация параллельного алгоритма свёртки в виде метода класса Convolutionizer

```
1   std::vector<std::vector<int>>
        Convolutionizer::convolution_parallel(std::vector<std::vector<int>> matrix, int
        n_threads)
2   {
3       std::vector<std::vector<int>> new_matrix(matrix.size() - 2,
            std::vector<int>(matrix[0].size() - 2, 0));
4
5       std::vector<std::thread> threads;
6
7       for (int i = 1; i <= n_threads; i++)
8       {
9           threads.push_back(std::thread(apply_kernel_parallel, i, n_threads,
                std::ref(matrix), std::ref(new_matrix), std::ref(kernel)));
10      }
11
12      for (auto& thread : threads)
13          thread.join();
14      threads.clear();
15
16
17      return new_matrix;
18  }
```

Листинг 3.4 — Реализация функции применения ядра к матрице

```
1   void apply_kernel_parallel(int i_s, int step, std::vector<std::vector<int>>& matrix,
        std::vector<std::vector<int>>& new_matrix, std::vector<std::vector<int>>& kernel)
2   {
```

```
    for (int i_t = i_s; i_t < matrix.size() - 1; i_t += step)
    {
        for (int j_t = 1; j_t < matrix[i_t].size() - 1; j_t++)
        {
            int new_elem = 0;
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    new_elem += matrix[i - 1 + i_t][j - 1 + j_t] * kernel[i][j];
                }
            }
            new_matrix[i_t - 1][j_t - 1] = new_elem;
        }
    }
}
```

3.5 Реализация тестирования алгоритмов

Для тестирования алгоритмов было реализованы следующие тесты:

1) тест на свёртку матрицы 5 на 5, заполненной заранее известными значениями;

Листинг 3.5 — Реализация тестов

```
void test()
{
    Convolutionizer conv;
    std::vector<std::vector<int>> matrix(5, std::vector<int>(5, 0));

    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (i == j)
                matrix[i][j] = i + j;
        }
    }

    std::vector<std::vector<int>> correct_result = {
        {10, -6, 0},
        {-6, 20, -10},
        {0, -10, 30}
    };

    std::vector<std::vector<int>> result;
    bool flag;
```

```
23
24      result = conv.convolution(matrix);
25      flag = true;
26      for (int i = 0; i < 3; i++)
27      {
28          for (int j = 0; j < 3; j++)
29          {
30              if (result[i][j] != correct_result[i][j])
31                  flag = false;
32          }
33      }
34      std::cout << "TEST CONV SIMPLE: " << flag << std::endl;
35
36      result = conv.convolution_parallel(matrix, 1);
37      flag = true;
38      for (int i = 0; i < 3; i++)
39      {
40          for (int j = 0; j < 3; j++)
41          {
42              if (result[i][j] != correct_result[i][j])
43                  flag = false;
44          }
45      }
46      std::cout << "TEST CONV PARALLEL 1: " << flag << std::endl;
47
48      result = conv.convolution_parallel(matrix, 2);
49      flag = true;
50      for (int i = 0; i < 3; i++)
51      {
52          for (int j = 0; j < 3; j++)
53          {
54              if (result[i][j] != correct_result[i][j])
55                  flag = false;
56          }
57      }
58      std::cout << "TEST CONV PARALLEL 2: " << flag << std::endl;
59
60      result = conv.convolution_parallel(matrix, 3);
61      flag = true;
62      for (int i = 0; i < 3; i++)
63      {
64          for (int j = 0; j < 3; j++)
65          {
66              if (result[i][j] != correct_result[i][j])
67                  flag = false;
68          }
69      }
```

```
70      std::cout << "TEST CONV PARALLEL 3: " << flag << std::endl;
71
72      result = conv.convolution_parallel(matrix, 4);
73      flag = true;
74      for (int i = 0; i < 3; i++)
75      {
76          for (int j = 0; j < 3; j++)
77          {
78              if (result[i][j] != correct_result[i][j])
79                  flag = false;
80          }
81      }
82      std::cout << "TEST CONV PARALLEL 4: " << flag << std::endl;
83  }
```

3.6 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

разм.	без параллел.	1 пот.	2 пот.	4 пот.	8 пот.	16 пот.	32 пот.	64 пот.
100	124254	197843	91950	107883	113683	203370	362230	757957
200	571961	483974	262076	201323	267745	246797	405434	950245
300	1009692	1160469	626281	491241	336894	460936	843504	1615922
400	2087216	1783830	1068245	674139	486513	705944	729709	914305
500	3712886	3566122	1674925	937623	695409	779717	922891	1271875
600	3526846	3513634	1853524	1135061	926665	940431	1123670	1538386
700	4656533	5051931	2739630	1797084	1148615	1206905	1160595	1459891
800	6670162	6919744	3299232	2074498	1539406	1587812	1466115	1668359
900	8162026	8338385	5350798	3396232	2092533	2134784	1708677	2034956
1000	10883751	10330769	5161384	3280373	2272809	2155541	2157472	2421776
1100	13378364	12503826	6754120	4009432	2722892	2562020	2667564	2683358
1200	15948354	14565637	8896638	5309428	3196955	3128936	3634144	3008889
1300	17172619	15685027	8315751	5159352	3544501	3480708	3510173	3690841
1400	21081695	19692531	11316242	6189062	4332132	4140771	3902364	4147262
1500	22604317	22169477	11080092	7004708	4790231	4193591	4256824	4322949
1600	25236594	26698045	14522262	8164930	5754227	4816110	4985526	5676034
1700	30790852	28845662	16219230	10956174	6725419	5805027	5550661	6287667
1800	36404743	33998653	18231903	9364057	6975351	5999613	6481451	6443922
1900	38265471	37343315	18875545	12534266	7790332	6982887	7089715	7697814
2000	45661973	44772720	23880992	13277630	8820412	7256153	7253400	7698523

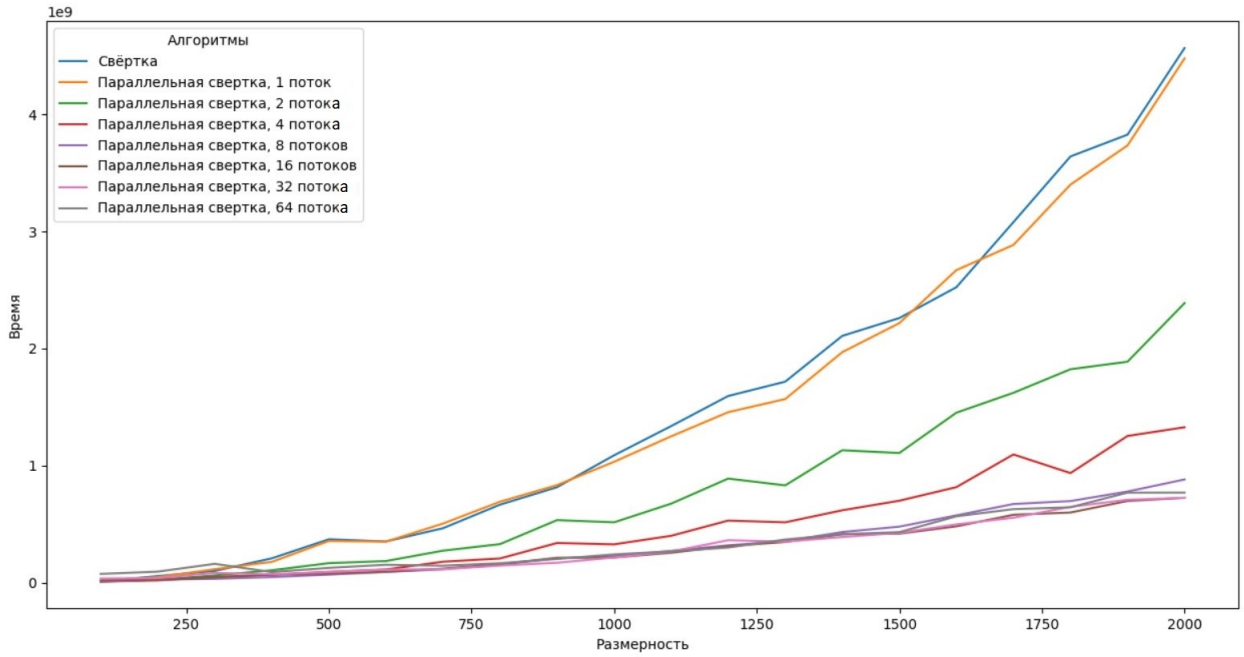

Рисунок 4.1 — График зависимости времени свёртки от размерности матрицы

4.3 Вывод

В результате эксперимента было получено, что на квадратных матрицах размерами от 100 на 100 до 2000 на 2000 распараллеливание позволяет добиться практически шестикратного (5,931) ускорения при использовании 64 потоков. Также можно заметить, что при увеличении количества потоков в два раза, скорость выполнения уменьшается в среднем в два раза относительно предыдущей. В результате можно сделать вывод о том, что использование параллельного программированние позволяет значительно ускорить работу некоторых алгоритмов.

Заключение

В процессе выполнения данной лабораторной работы были изучены алгоритм свёртки и параллельный алгоритм всёртки. 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 матрицах размерами от 100 на 100 до 2000 на 2000 распараллеливание позволяет добиться практически шестикратного (5,931) ускорения при использовании 64 потоков. Также можно заметить, что при увеличении количества потоков в два раза, скорость выполнения уменьшается в среднем в два раза относительно предыдущей. В результате можно сделать вывод о том, что использование параллельного программированние позволяет значительно ускорить работу некоторых алгоритмов. Целью данной лабораторной работы являлось изучение параллельного программирования на примере алгоритма свёртки, 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] Springenberg, Jost Tobias; Dosovitskiy, Alexey; Brox, Thomas & Riedmiller, Martin (2014-12-21), Striving for Simplicity: The All Convolutional Net