Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ В.В. Шайдуров
« 29» июня 2020 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01Математика и компьютерные науки

ЭФФЕКТИВНАЯПАРАЛЛЕЛИЗАЦИЯКОДА В ОПЕРАЦИЯХС РАЗРЕЖЕННЫМИ МАТРИЦАМИ

Научный руководитель,
кандидат физико-математических наук,
доцент _____ 29.06.2020/Е.Д.Карепова

Выпускник _____ 29.06.2020/Н.А.Назаров

Красноярск 2020

РЕФЕРАТ

Выпускная квалификационная работа на тему «ЭФФЕКТИВНАЯ ПАРАЛЛЕЛИЗАЦИЯ КОДА В ОПЕРАЦИЯХ С РАЗРЕЖЕННЫМИ МАТРИЦАМИ» содержит 49 страниц текстового документа, состоит из Введения, трех глав, заключения, 26 рисунков, 31 таблицы и списка литературы; список литературы содержит 6 источников.

Ключевые слова: РАЗРЕЖЕННЫЕ МАТРИЦЫ, ФОРМАТ ХРАНЕНИЯ МАТРИЦЫ, ОПЕРАЦИИ НАД МАТРИЦАМИ, OPENMP, УСКОРЕНИЕ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ

Цель данной работы – программная реализация и сравнительный анализ эффективности распараллеливания с помощью технологии OpenMP алгоритмов основных операций над матрицами, хранящимися в форматах COO и CRS.

Алгебра разреженных матриц – раздел вычислительной математики, который помогает решать задачи, встречающиеся на практике. Проблема работы с разреженными матрицами возникает из-за того, что, несмотря на огромную размерность, эти матрицы содержат большое количество нулевых элементов, которые не вносят вклад в расчеты. Для эффективной работы с такими матрицами были придуманы специальные форматы для их хранения и, соответственно, с учетом особенностей каждого формата изменяется реализация алгоритмов для основных операций алгебры над разреженными матрицами.

В работе рассмотрены и протестированы последовательные и параллельные версии алгоритмов, реализующих операции транспонирования, умножения матрицы на вектор и на другую разреженную матрицу. Для параллельных алгоритмов были рассчитаны такие показатели как ускорение и эффективность параллельных алгоритмов, по которым можно судить о целесообразности использования этих алгоритмов на практике.

Содержание

ВВЕДЕНИЕ

Алгебра разреженных матриц один из важных разделов современной математики, который необходим для многих практических задач. Разреженная матрица – матрица, в которой количество ненулевых элементов много меньше нулевых элементов. Задачи, связанные с разреженными матрицами, встречаются во многих областях наук, таких как математика, физика, биология, экономика и другие. В математике такие задачи довольно часто встречаются при решении дифференциальных уравнений в частных производных и теории графов.

В современных задачах матрицы имеют большие размеры, такие что хранить и работать с матрицей, хранящейся в классическом формате двумерного массива, очень сложно, а иногда и невозможно, так как требуется много памяти для их хранения. Для задач с разреженными матрицами были придуманы специальные форматы хранения матриц, которые решают эту проблему. Такие форматы требуют меньше памяти для хранения матрицы, что позволяет работать с матрицами большего размера.

Несмотря на то, что были придуманы специальные форматы для хранения разреженных матриц, которые направлены на уменьшение занимаемой памяти при хранении матрицы, эти форматы могу потребовать большое количество времени для операций над матрицами, поэтому для более быстрого выполнения операций над матрицами можно воспользоваться технологией OpenMP для распараллеливания алгоритмов операций над матрицами.

Цель работы: сравнительный анализ эффективности распараллеливания алгоритмов основных операций над матрицами, хранящихся в специальных форматах COO и CRS с помощью технологии OpenMP.

Для достижения указанной цели были поставлены следующие задачи.

- Реализовать алгоритмы основных операций над матрицами (умножение матрицы на вектор, транспонирование матрицы, умножение матрицы на матрицу), хранящихся в форматах COO и CRS.
- Реализовать распараллеливание алгоритмов основных операций над матрицами с помощью технологии OpenMP.
- Провести вычислительные эксперименты на многоядерном процессоре на тестовых наборах данных, определить ускорение и эффективность каждого формата в зависимости от размерности.
- Провести вычислительные эксперименты на матрицах, полученных в реальных задачах.
- Провести сравнительный анализ выбранных форматов хранения по объему занимаемой памяти и эффективности распараллеливания.

Благодарность научному руководителю, кандидату физико-математических наук Евгении Дмитриевне Кареповой, за оказанную помощь и наставления в выпускной квалификационной работе.

1 Описание форматов хранения разреженных матриц и операций над матрицами в различных форматах

1.1 Описание форматов

Классический метод хранения матриц представляет из себя хранение всех элементов матрицы в двумерном массиве. Он является простым для понимания и для реализации различных операций над матрицами. Но когда речь заходит о реальных задачах, то такой формат хранения матрицы требует много памяти для хранения разреженной матрицы, большая часть памяти при таком хранении используется впустую, так как нулевые элементы разреженной матрицы не вносят свой вклад при умножении разреженной матрицы на вектор или на другую разреженную матрицу. Поэтому были придуманы другие форматы хранения, специально для разреженных матриц, которые будут описаны ниже.

1.1.1 Формат хранения разреженной матрицы COO

Один из первых и достаточно простой для понимания формат хранения разреженных матриц – координатный формат (COO). Вся информация о матрице хранится в 3 массивах, размерность каждого массива равна количеству ненулевых элементов матрицы.

Массив Value хранит в себе значения ненулевых элементов матрицы.

Массив Row содержит в себе номер строки элемента матрицы, соответствующего элемента из Value.

Массив Col содержит в себе номер столбца элемента матрицы, соответствующего элемента из Value.

Пример хранения матрицы в координатном формате (COO) представлен на рисунке 1.

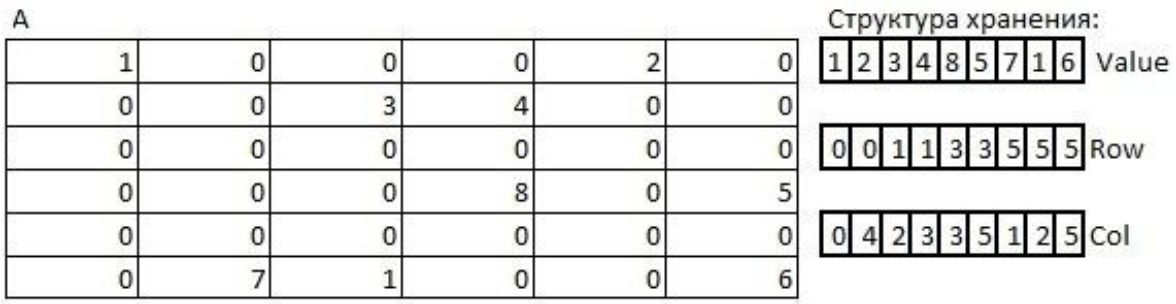

Рисунок 1 – Пример координатного формата хранения разреженной матрицы

На рисунке 1 представлен упорядоченный вид хранения матрицы в формате COO, как мы можем видеть, все элементы разреженной матрицы хранятся последовательно, начиная с верхнего правого угла и заканчивая правым нижним углом матрицы, идя по строкам слева на права. Так же

разреженную матрицу можно хранить в формате COO не упорядоченно. Неупорядоченное хранение разреженной матрицы в формате COO мы можем получить после проделывание некоторых операций над матрицей, например умножения разреженной матрицы в формате COO на разреженную матрицу в формате COO. Для более комфортной дальнейшей работы с разреженной матрицей в формате COO, лучше будет отсортировать массивы Value, Row, Col. Упорядоченное хранение разреженной матрицы в формате COO позволяет находить элементы в нужной строке или столбце быстрее, но также приводит к перепаковкам при вставке или удаление элементов. Формат COO является довольно простым для понимания и использования, но, с другой стороны, он малоэффективен с точки зрения алгоритмов обработки и использования памяти. Представляет интерес, насколько эффективно можно распараллелить основные операции над разреженными матрицами, хранящимися в этом формате. Более эффективное по сравнению с распространенным форматом CRS распараллеливание позволило бы рекомендовать этот формат при написании соответствующего ПО.

1.1.2 Формат хранения разреженной матрицы CRS (CSR)

Самый распространенный формат, который широко используется, для хранения разреженных матриц это формат - CompressedSparseRows(CSR) или CompressedRowStorage(CRS). Этот формат устраняет недоработки формата хранения разреженных матриц COO. Вся информация о матрице так же хранится в 3 массивах, два из которых, а именно, массивы Value и Col, будут иметь размерность равную количеству ненулевых элементов, а третий массив, массив RowIndex, будет иметь размерность N+1.

Массив Value содержит в себе значения ненулевых элементов матрицы.

Массив ColCol содержит в себе номер столбца элемента матрицы, соответствующего элемента из Value.

Массив RowIndex – главное отличия формата CRS от формата COO. В массиве RowIndexхранится не номер строки, а число элементов в каждой строке с накоплением.

МассивRowIndexимеет размерность N+1. Первый элемент массива RowIndex равен 0. Элементы строки i в массиве Valueможно найти по индексам массиваотRowIndex[i] до RowIndex[i+1]–1.

Так же можем заметить, что последний элемент массива RowIndex хранит число равное количеству ненулевых элементов в матрице.

Пример хранения матрицы в формате CRS представлен на рисунке 2.

У CSR формата есть множество различных модификаций.

Также как и COO, формат CRS может быть упорядоченным и неупорядоченным. Обычно формат CRS представлен в упорядоченном формате, где элементы массива Value и Col упорядочены по строкам, но формат CRS может быть не упорядоченным по столбцам.

A

1	0	0	0	2	0
0	0	3	4	0	0
0	0	0	0	0	0
0	0	0	8	0	5
0	0	0	0	0	0
0	7	1	0	0	6

Структура хранения:

1	2	3	4	8	5	7	1	6	Value

0	4	2	3	3	5	1	2	5	Col

0	2	4	4	6	6	9	RowIndex

Рисунок 2 – пример CRS формата хранения разреженной матрицы

По такому же принципу, который описан выше, работает и формат CSR, с одни лишь исключением – массив RowIndex будет храниться не число элементов в каждой строке, а число элементов в каждом столбце. В некоторых ситуациях выгодней и проще использовать формат CRS, а в других формат CSR.

1.2 Последовательные алгоритмы умножения матрицы на вектор

По определению, результатом умножения матрицы A размером NxN на вектор b длинной N, является вектор r длинной N, в котором каждый элемент есть результат скалярного умножения строки матрицы A и вектора b. На рисунке 3 можно наглядно увидеть, как проводятся вычисления. По формуле (1) можно найти все элементы из результирующего вектора r.

$$A * \vec{b} = \vec{c}$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} * \begin{pmatrix} b_1 \\ . \\ . \\ . \\ b_n \end{pmatrix} = \begin{pmatrix} a_{11} * b_1 + a_{12} * b_2 + \cdots + a_{1n} * b_n \\ . \\ . \\ . \\ a_{n1} * b_1 + a_{n2} * b_2 + \cdots + a_{nn} * b_n \end{pmatrix}$$

Рисунок 3 – общая схема умножение матрицы на вектор

$$c_j = a_{i1} * b_1 + a_{i2} * b_2 + \cdots + a_{in} * b_n, \quad (1)$$

где $a_{i\,j}$ – элемент из строки номерi,столбца номер j матрицы A;

b_j – j-ый элемент вектораb;

c_i – i-ый элемент результирующего вектора C.

Для подсчета c_iнеобходимо умножить каждый элемент i-ой строки на соответствующий элемент из вектора b, а затем просуммировать результат,

таки образом количество необходимых операций можно найти по формуле (2).

$$T = n(2n - 1), \quad (2)$$

где T – количество операций;
n – размер матрицы.

1.2.1 Умножение матрицы на вектор в формате COO.

Хранение разреженной матрицы в формате COO благоприятно влияет на количество операция и время необходимое для умножения матрицы на вектор, так как в формате COO не хранятся нулевые элементы, то есть не придется умножать нулевые элементы матрицы при скалярном произведении, также не придется суммировать их.

Для умножения матрицы A в формате COO на вектор b воспользуемся Алгоритмом 1.

Алгоритм 1

Имеем матрицу A, записанную в виде массивов Value, Row, Col, размерность этих массивов равна количеству ненулевых элементов size,и массив b, в котором записан вектор b.

Алгоритм перемножение Матрицы A на вектор b состоит в том, что для i-го элемента массива Value определяем номер строки (Row[i]) и номер столбца (Col[i]), затем высчитываем результирующий массив Result по формуле:

1.Для всех i от 0 до size выполнить (пробегаем все ненулевые элементы матрицы):

1.1. Result[Row[i]] += Value[i] * b[Col[i]];

Таким образом в массиве Resultбудет хранится результат умножения матрицы A на вектор b.

1.2.2 Умножение матрицы на вектор в формате CRS.

Умножение матрицы в формате CRS на вектор имеет те же преимущества перед классическим умножением матрицы на вектор что и формат COO.

Для умножения матрицы A в формате CRS на вектор b воспользуемся следующим Алгоритмом 2.

Алгоритм 2

Имеем матрицу A, записанную в виде массивов Value, Col, RowIndex, размерность массивовValueи Colравно количеству ненулевых элементов, а

размерность массива RowIndexравно n+1, и массив b, в котором записан ветор b.

2.Для всех i от 0 до n (пробегаем все строки матрицы) выполнить:

2.1.Для всех j от RowIndex[i] до RowIndex[i+1](пробегаем все ненулевые элементы в i-ой строке) выполнить:

2.1.1.Result[i] += Value[j] * b[Col[j]](высчитываем результирующий массив)

Таким образом в массиве Resultбудет хранится результат умножения матрицы A на вектор b.

1.3 Последовательные алгоритмы транспонирования матрицы.

По определению транспонированная матрица – матрица, полученная из исходной матрицы, заменой строк на столбцы. На рисунке 4 можно наглядно видеть транспонирование матрицы. По формуле (3) можно найти все элементы из A^T.

$$A=\begin{pmatrix} a_{1\,1} & a_{1\,2} & a_{1\,3} & \cdots & a_{1\,n-1} & a_{1\,n} \\ a_{2\,1} & a_{2\,2} & \cdots & \cdots & \vdots & a_{2\,n} \\ a_{3\,1} & \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ a_{n-1\,1} & \vdots & \cdots & \cdots & a_{n-1\,n-1} & a_{n-1\,n} \\ a_{n\,1} & a_{n\,2} & \cdots & \cdots & a_{n-1\,n} & a_{n\,n} \end{pmatrix} \Rightarrow A^T= \begin{pmatrix} a_{1\,1} & a_{2\,1} & a_{3\,1} & \cdots & a_{n-1\,1} & a_{n\,1} \\ a_{1\,2} & a_{2\,2} & \cdots & \cdots & \vdots & a_{n\,2} \\ a_{1\,3} & \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ a_{1\,n-1} & \vdots & \cdots & \cdots & a_{n-1\,n-1} & a_{n-1\,n} \\ a_{1\,n} & a_{2\,n} & \cdots & \cdots & a_{n-1\,n} & a_{n\,n} \end{pmatrix}$$

Рисунок 4 – общая схема транспонирования матрицы

$$a_{ij}^{T} = a_{ji}, \qquad (3)$$

где $a_{i\,j}^{T}$ элемент из матрицы A^T;

$a_{i\,j}$ - элемент из матрицы A.

Для транспонирования матрицы необходимо сохранить каждый элемент матрицы Ас места (i,j), в матрицу A^T на место (j,i). Таким образом, количество операций необходимое для транспонирования матрицы можно найти по формуле (4).

$$T = n^2, \qquad (4)$$

где T – количество операций;

n- размер матрицы.

1.3.1 Транспонирование матрицы в формате COO

Транспонирование разреженной матрицы в формате COO не сильно отличается от транспонирования матрицы в классическом формате, за тем лишь исключением, что нам не требуется переносить нулевые элементы, в этом заключается преимущество формата COO перед классическим форматом хранения матриц.

Для транспонирования матрицы A воспользуемся Алгоритмом 3.

Алгоритм 3

Имеем матрицу A, записанную с помощью массивов Value, Cow, Row.

Используем переменную buff как буфер.

Будем хранить транспонированную матрицу с помощью массивов TCol, TRow, TValue.

3.1. Копировать массив Row в TCol.

3.2. Копировать массив Col в TRow.

3.3. Копировать массив Value в TValue.

Таким образом, транспонированная матрица будет храниться в массивах TValue, TRow, TCol.

Следует заметить, что в результате транспонирования матрицы в формате COO, мы получаем транспонированную матрицу в неупорядоченном формате COO, что не очень удобно для дальнейшей работы с ней. Ниже, в Алгоритме 4 представлена сортировка матрицы в формате COO для более комфортной работы с ней.

Для дальнейшей работы удобно упорядочить по номерам строк и столбцов хранение транспонированной матрицы, т.е. перезаписать массивы TCol, TRow, TValue в порядке увеличения номеров строк и столбцов (см. Алгоритма 4).

Алгоритм 4

4.1. Пусть num – номер элемента при упорядоченном хранении. Начальное значение num=0.

4.2. Для всех i от 0 до n (пробегает строки матрицы) выполнить:

4.2.1. Для всех j от 0 до size (пробегает массив TRow) выполнить:

4.2.1.1. Если номер строки TRow[j] транспонированной матрицы совпадает с текущим i, меняем местами элементы с номерами num и j в массивах TCol, TRow, TValue.

4.2.1.2. num++

Таким образом, после алгоритма 4, транспонированная матрица будет храниться в массивах TValue, TRow, TCol упорядоченно.

1.3.2 Транспонирование матрицы в формате CRS

Транспонирование разреженной матрицы в формате CRS имеет преимущество перед транспонированием разреженной матрицы в классическом формате, так как в формате CRS нам не надо переносить нулевые элементы разреженной матрицы.

Для транспонирования матрицы A воспользуемся Алгоритмом 5.

Алгоритм 5

Имеем матрицу A, записанную с помощью массивов Value, Col, RowIndex, размерность массивов Valueи Colравна size, а размерность массива RowIndex равна n+1.

Будем хранить транспонированную матрицу с помощью массивов TCol, TRowIndex, TValue.

Пусть num – номер элемента в транспонированной матрице при упорядоченном хранении,k – счетчик для определения номера столбца в транспонированной матрице. Начальное значение num=0, k=0.

5.1.Для всех i от 0 до n (пробегаем все строки матрицы) выполнить:

5.1.1.Для всех j от 0 до size (пробегаем все ненулевые элементы матрицы) выполнить:

5.1.1.2.Если номер столбца Col[j] совпадает с текущим i, то записываем элементValue[j] в TValue[num]

5.1.1.2.1Пока j больше или равно массиву RowIndex[k].

5.1.1.2.1.1.Инкрементируем переменную k.

5.1.1.2.2.Присваиваем значение TCol[num]=number-1.

5.1.1.2.3.Инкрементируем num.

5.1.1.2.4.Обнуляем переменную k.

5.1.2.Массиву TRowIndex[i+1] присваиваем значение num.

Таким образом, транспонированная матрица будет храниться в массивах TValue, TRowIndex, TCol.

1.4 Последовательный алгоритм умножение матрицы на матрицу

По определению результатом умножения матрицы A размером NxN на матрицу B размером NxN есть матрица C размером NxN.

Наглядно увидеть алгоритм умножения матрицы на матрицу можно на рисунке 5. Каждый элемент матрицы C можно найти по формуле (5).

$$A * B = C$$

$$\begin{pmatrix} a_{1\,1} & \cdots & a_{1\,n} \\ \vdots & \ddots & \vdots \\ a_{n\,1} & \cdots & a_{n\,n} \end{pmatrix} * \begin{pmatrix} b_{1\,1} & \cdots & b_{1\,n} \\ \vdots & \ddots & \vdots \\ b_{n\,1} & \cdots & b_{n\,n} \end{pmatrix} = \begin{pmatrix} c_{1\,1} & \cdots & c_{1\,n} \\ \vdots & \ddots & \vdots \\ c_{n\,1} & \cdots & c_{n\,n} \end{pmatrix}$$

Рисунок 5 – Общая схема умножения матрицы на матрицу

$$c_{i\,j} = a_{i\,1} * b_{1\,j} + a_{i\,2} * b_{2\,j} + \cdots + a_{i\,n} * b_{n\,j}, \quad (5)$$

где $a_{i\,j}$ - элемент из матрицы A;

$b_{i\,j}$ - элемент из матрицы B;

$c_{i\,j}$ - элемент из результирующей матрицы C.

Для нахождения всех элементов из матрицы C, необходимо умножить элементы из i-ой строки матрицы A, на элементы из j-ой строки матрицы B, затем просуммировать их, и сделать так для каждого элемента из матрицы C. Таким образом, количество операций можно оценить по формуле (6).

$$\mathrm{T} = \mathrm{n}^2 * (\mathrm{n} - 1), \quad (6)$$

где T – количество операций;

n- размер матрицы.

1.4.1 Умножение матрицы на матрицу в формате COO

Умножение разреженной матрицы в формате COO на разреженную матрицу в формате COO, имеет преимущество перед умножением в классическом формате, так как в формате COO мы не учитываем нулевые элементы.

Для умножения матрицы A на матрицу B воспользуемся Алгоритмом 6.

Алгоритм 6

Имеем две матрицы A и B, которые заданы в виде массивов Value1, Row1, Col1, CV1 для матрицы A, и массивов Value2, Row2, Col2, CV2 – для матрицы B.

Результат будем хранить в двумерном массиве RMatrix.

6.1.Для i от 0 до CV1 (пробегаем все ненулевые элементы в первой матрице) выполнить:

6.1.1.Для j от 0 до CV2 (пробегаем все ненулевые элементы во второй матрице) выполнить:

6.1.1.1.Если значение массива Col1[i] равно значению массива Row2[j], то высчитываем результат по формуле:

RMatrix[Row[i]][Col2[j]] += Value1[i] * Value2[i]

Таким образом, результирующая матрица будет храниться в двумерном массиве RMatrix.

1.4.2 Умножение матрицы на матрицу в формате CRS

Для умножения матрицы A на матрицу B воспользуемся Алгоритмом 7.

Алгоритм 7

Имеем две матрицы A и B, которые заданы в виде массивов Value1, RowIndex1, Col1, CV1 для матрицы A, и массивов Value2, RowIndex2, Col2, CV2 – для матрицы B.

Результат будем хранить в RMatrix.

7.1.Транспонируем матрицу B при помощи Алгоритма 5 в параграфе 1.1.3 для эффективного доступа к ее элементам.

7.2.Для i от 0 до n(пробегаем все строки первой матрицы) выполнить:

7.2.1.Для j от RowIndex[i] до RowIndex[i+1](пробегаем все ненулевые элементы первой матрицы в i-ой строке) выполнить:

7.2.1.1.Для всех q от 0 до n (пробегаем все строки второй матрицы) выполнить:

7.2.1.1.1. Для всех z от RowIndex2[q] до RowIndex2[q+1] (пробегаем все ненулевые элементы второй транспонированной матрицы в q-ой строке) выполнить:

7.2.1.1.1.1. Если элемент массиваCol1[j] равен элементу массива Col2[z],то высчитываем результат по формуле:

RMatrix[i][q] += Value[j] * Value2[z].

Таким образом, результирующая матрица будет храниться в двумерном массиве RMatrix.

2 Реализация с помощью технологии OpenMP операций с разреженными матрицами.

В этом пункте рассмотрим реализации операций над матрицами с помощью технологии OpenMP. Так как алгоритмы операций над матрицами итерационные и состоят в основном из циклов, то технология OpenMP хорошо подойдет для параллельной реализации алгоритмов.

2.1 Особенности реализации с помощью технологии OpenMP операций с разреженными матрицами для формата COO

Алгоритм умножения матрицы на вектор в формате COO состоит из одного цикла по одной переменной (Алгоритм 1 из пункта 1.2.1), поэтому для реализации этого алгоритма с помощью технологии OpenMP достаточно использовать парадигму #pragmaompparallelfor перед циклом. Таким образом, все операции по умножению элементов из матрицы на элементы из вектора, будут выполняться параллельно независимо друг от друга.

Алгоритм транспонирования матрицы в формате COO состоит в том, что мы переписываем элементы из массива Row в массив транспонированной матрицы TCol, а элементы из массива Col, в массив транспонированной матрицы TRow (Алгоритм 3 из пункта 1.3.1). Этот алгоритм реализован с помощью одного цикла по одной переменной, поэтому для реализации этого алгоритма с помощью технологии OpenMP достаточно использовать парадигму #pragmaompparallelfor перед циклом. Таким образом, все элементы из массивов Row и Col перепишутся в массивы TCol и TRow соответственно, параллельно независимо друг от друга.

Алгоритм умножения матрицы на матрицу в формате COO состоит из двух циклов, для ненулевых элементов первой матрицы и ненулевых элементов второй матрицы (Алгоритм 5 из пункта 1.4.1). Для реализации алгоритма умножения матрицы на матрицу с помощью технологии OpenMP воспользуемся парадигмой OpenMP #pragmaompparallelfor и условие private. Условие private указывает на то, что каждый поток должен иметь свою копию переменной на всем протяжении своего исполнения. Перед внешним циклом для ненулевых элементов из первой матрицы необходимо добавить парадигму #pragmaompparallelfor, и условие private(j), где j–это переменная для ненулевых элементов из второй матрицы. Таким образом, все потоки будут выполнять свою работу независимо друг от друга.

2.2 Особенности реализации с помощью технологии OpenMP операций с разреженными матрицами для формата CRS

Алгоритм умножения матрицы на вектор в формате CRS состоит из двух циклов, первый цикл проходит все строки, а второй проходит все ненулевые элементы в каждой строке (Алгоритм 2 из пункта 1.2.2). Для реализации этого алгоритма с помощью технологии OpenMP необходимо

использовать парадигму #pragmaompparallelfor и условие private(j), где j– переменная для ненулевых элементов в каждой строке. Таким образом, все операции по умножению элементов из матрицы на элементы из вектора, будут выполняться параллельно независимо друг от друга.

Для реализации алгоритма транспонирования матрицы в формате CRS потребуется дополнительно создать двумерный массив, размерности N x $size_i$, где N – количество строк матрицы, $size_i$ –количество ненулевых элементов в i-ой строке матрицы. Этот двумерный массив необходим для того, что бы определить номер столбца элемента в транспонированной матрице независимо от других потоков. Для транспонирования матрицы в формате CRS воспользуемся алгоритмом 8 с использованием парадигмы #pragmaompparallelforи условия private(j).

Алгоритм 8

Имеем матрицу A, записанную с помощью массивов Value, Col, RowIndex, размерность массивов Valueи Colравна size, а размерность массива RowIndexравна n+1.

Будем хранить транспонированную матрицу с помощью массивов TCol, TRowIndex, TValue.

Пусть num – номер элемента в транспонированной матрице при упорядоченном хранении,k–двумерный массив размерности N x $size_i$,переменная qотвечает за номер ненулевого элемента в каждой строке, переменная сномер элемента в транспонированной матрице.

8.1.Для всех i от 0 до n выполнить:

8.1.1. Обнуляем переменную q

8.1.2. Для всех j от 0 до size выполнить:

8.1.2.1. Если Col[j] совпадает с текущим i, то выполнить:

8.1.2.1.1. Пока j больше или равно RowIndex[k[i][q]]инкрементируем счетчик k[i][q].

8.1.2.2. Инкрементируем счетчик q.

8.2. Для всех iот 0 до n

8.2.1. Для всехjот 0 до RowIndex[i+1]-RowIndex[i]

8.2.1.1.Записываем значение Value[j] в TValue[c]

8.2.1.2. Записываем значение k[i][j]-1 в TCol[c]

8.2.1.3. Считаем количество элементов в строке транспонированной матрице с помощью TRowIndex[num]++

8.2.2. Преобразуем массив TRowIndex следующим образом: TRowIndex[num] += TRowIndex[num-1]

8.2.3. Инкрементируем счетчик num.

Таким образом, транспонированная матрица будет храниться в массивах TValue, TRowIndex, TCol.

Алгоритм умножения матрицы на матрицу состоит из 4 циклов, первый цикл пробегает все строки первой матрицы, второй цикл пробегает все

ненулевые элементы для каждой строки, третий цикл пробегает все строки для второй матрицы, четвертый цикл пробегает все ненулевые элементы для строки второй матрицы (Алгоритм 6 из пункта 1.4.2). Для реализации этого алгоритма с помощью технологии OpenMP необходимо использовать парадигму #pragmaompparallelfor и условие private(j,q,z), где j-переменная для каждого ненулевого элемента i-ой строки из первой матрицы, q-переменная для строк второй матрицы, z-переменная для каждого ненулевого элемента q-ой строки из второй матрицы. Таким образом, все операции по умножению матрицы на матрицу, будут выполняться параллельно независимо друг от друга.

2.3 Численное исследование работы алгоритмов с помощью технологии OpenMP

В этом пункте рассмотрим время работы алгоритмов, реализованных с помощью технологии OpenMP, на известной матрице, формат которой представлен на рисунке 6. Матрица является разреженной матрицей диагонального вида, с количество ненулевых элементов $5n + 42$. Так же рассмотрим ускорение и эффективность алгоритмов в зависимости от количества нитей и размера матрицы. Рассчитывать ускорение алгоритма будем по формуле (7), эффективность будем рассчитывать по формуле (8).

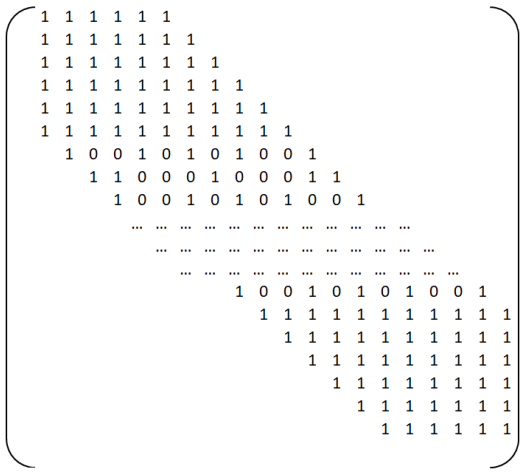

Рисунок 6 – схематичное изображение матрицы.

$$a = \frac{T}{T_p}, \tag{7}$$

где a – ускорение;
T – время последовательного алгоритма;
T_p – время параллельного алгоритма.

$$w = \frac{a}{k}, \quad (8)$$

где w – эффективность;
k – количество потоков;
a – ускорение.

Рассмотрим характеристики матриц, их размер, количество ненулевых элементов, память, которая необходима для хранения матрицы в разных форматах, время последовательных алгоритмов операций над матрицами, в таблице 1 и 2.

В таблице 1 можно заметить, что операции над матрицами, такие как транспонирование и умножение матрицы на матрицу, в форматах хранения COO и CRS выполняются быстрее, чем те же операции над матрицами в формате двумерного массива. При этом матрицы в форматах COO и CRS требует меньше памяти для хранения.

В таблице 2 можно заметить, что для операции умножения матрицы на вектор, выгодней и по памяти, и по времени использовать специальные форматы хранения матрицы, нежели формат двумерного массива.

Для алгоритмов в формате двумерного массива были взяты матрицы, размер которых не превышает 12000 x 12000, так как тесты проводились на кластере СФУ, оперативная память которого ограничена 4гб. По той же причине для алгоритмов умножения матрицы на матрицу в форматах COO и CRSбыли взяты матрицы, размер которых не превышает 12000 x 12000.

Из таблицы 1 и 2 можно сделать вывод, что форматы для хранения матриц COO и CRS позволяют проводить операции над матрицами большего размера, так как матрицы в этих форматах занимают меньше памяти. Так же операции над матрицами, хранящиеся в форматах COO и CRS, проходят быстрее. Также можно заметить, что алгоритмы для формата COO требуют меньше времени, но при этом матрица в формате CRS занимает меньше памяти.

Таблица 1 - характеристики матриц и время работы последовательных алгоритмов транспонирование матрицы и умножение матрицы на матрицу

Размер матрицы	1500	3000	6000	12000	120000	240000	360000
Количество ненулевых элементов	7542	15042	30042	60042	600042	1200042	1800042
Необходимая память в двумерном массиве (Мб)	2,146	8,583	34,332	137,329	13732,910	54931,641	123596,191
Необходимая память в формате COO(Мб)	0,022	0,043	0,086	0,172	1,717	3,433	5,150
Необходимая память в формате CRS(Мб)	0,016	0,032	0,063	0,126	1,259	2,518	3,777
Транспонирование матрицы в классическом формате	3,128e-2	1,321e-1	5,233e-1	6,440e-0	-	-	-
Транспонирование матрицы в COO формате	9,298e-5	1,760e-4	3,769e-4	9,711e-4	1,034e-2	2,060e-1	3,081e-2
Транспонирование матрицы в CRS формате	1,723e-2	6,792e-2	2,700e-1	1,008e-0	108,000e-0	503,000e-0	1203,000e-0
Умножение матрицы на матрицу в классическом формате	41,300e-0	333,252e-0	2706,693e-0	3.23910,900e-0	-	-	-
Умножение матрицы на матрицу в COO формате	9,886e-2	4,031e-1	1,602e-0	6,391e-0	-	-	-
Умножение матрицы на матрицу в CRS формате	1,050e-1	4,138e-1	1,647e-0	6,536e-0	-	-	-

Таблица 2 – характеристики матриц и время работы последовательных алгоритмов умножения матрицы на вектор

Размер матрицы	1500	3000	6000	12000	120000	240000	360000	480000	720000	960000
Количество ненулевых элементов	7542	15042	30042	60042	600042	1200042	1800042	2400042	3600042	4800042
Необходимая память в двумерном массиве (мб)	2,146	8,583	34,332	137,329	13732,910	54931,641	123596,191	219726,563	494384,766	878906,250
Транспонирование матрицы в COO формате	0,022	0,043	0,086	0,172	1,717	3,433	5,150	6,867	10,300	13,733
Необходимая память в формате CRS(мб)	0,016	0,032	0,063	0,126	1,259	2,518	3,777	5,035	7,553	10,071
Умножение на вектор в классическом формате	1,069e-2	4,272e-2	1.678e-1	3.239e-0	-	-	-	-	-	-
Умножение на вектор в COO	2,658e-4	3,541e-4	4,542e-4	6,680e-4	8,248e-3	1,140e-1	1,120e-1	1,602e-2	1,424e-2	3,293e-2
Умножение на вектор в CRS	2,111e-2	1,014e-1	1,842e-2	2,559e-2	2,177e-2	1,701e-2	1,572e-2	2,520e-2	2,352e-2	2,566e-2

2.3.1. Численные исследования работы алгоритмов с помощью технологии OpenMP для формата COO

Далее рассмотрим время работы, ускорение и эффективность алгоритмов умножения матрицы на вектор в формате COO на 1, 2, 4 и 8 нитях.

Таблица 3 – время работы алгоритма умножения матрицы на вектор в формате COO

		Количество нитей			
Размер матрицы	Количество ненулевых элементов	1	2	4	8
1500	7542	2,658e-4	2,142e-3	9,291e-3	2,801e-4
3000	15042	3,541e-4	2,987e-3	1,003e-2	2,011e-2
6000	30042	4,542e-4	4,183e-3	9,297e-3	1,388e-2
12000	60042	6,680e-4	7,079e-3	1,101e-2	2,442e-2
120000	600042	8,248e-4	4,454e-3	1,397e-2	1,823e-2
240000	1200042	1,140e-4	1,020e-1	2,330e-1	1,141e-1
360000	1800042	1,120e-2	1,091e-2	1,282e-2	1,910e-2
480000	2400042	1,602e-2	1,403e-2	1,592e-2	1,893e-2
720000	3600042	1,424e-2	1,957e-2	1,762e-2	2,468e-2
960000	4800042	3,293e-2	2,401e-2	2,264e-2	2,622e-2

Таблица 4 – ускорение алгоритма умножения матрицы на вектор в формате COO

		Количество нитей		
Размер матрицы	Количество ненулевых элементов	2	4	8
1500	7542	0,124	0,029	0,949
3000	15042	0,119	0,035	0,018
6000	30042	0,109	0,049	0,033
12000	60042	0,094	0,061	0,027
120000	600042	1,852	0,590	0,453
240000	1200042	1,118	0,487	0,996
360000	1800042	1,112	0,951	0,635
480000	2400042	1,141	1,006	0,846
720000	3600042	1,239	1,376	0,982
960000	4800042	1,371	1,455	1,256

Из таблицы 4 видно, что с увеличением размера матрицы и ростом ненулевых элементов ускорение алгоритма увеличивается для каждого количества нитей. Однако, при одинаковом размере матрицы и увеличении количества нитей, ускорение в большинстве уменьшается, лишь иногда увеличиваясь. Наглядно зависимость ускорения от размера матрицы и количества нитей можно увидеть на рисунке 7. Эффективность алгоритмов можно увидеть в таблице 5.

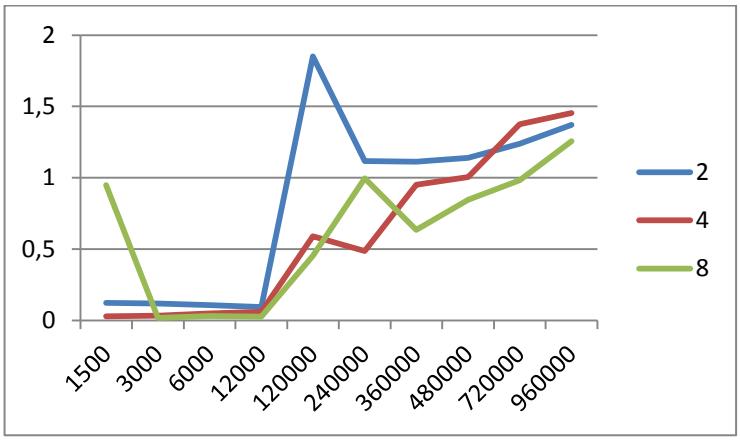

Рисунок 7 – график зависимости ускорения алгоритма умножения матрицы на вектор в формате COO от размера матрицы

Таблица 5 – эффективность работы алгоритма умножения матрицы на вектор в формате COO

		Количество нитей		
Размер матрицы	Количество ненулевых элементов	2	4	8
1500	7542	0,062	0,007	0,119
3000	15042	0,060	0,009	0,002
6000	30042	0,055	0,012	0,004
12000	60042	0,047	0,015	0,003
120000	600042	0,926	0,148	0,057
240000	1200042	0,559	0,122	0,125
360000	1800042	0,556	0,238	0,079
480000	2400042	0,571	0,252	0,106
720000	3600042	0,686	0,364	0,157
960000	4800042	0,062	0,007	0,119

Из таблицы 5 можно сделать вывод, что эффективность алгоритмов не увеличивается от увеличения количества потоков или размером матрицы.

Далее рассмотрим время работы, ускорение и эффективность алгоритмов транспонирования матрицы в формате COO на 1, 2, 4 и 8 нитях.

Из таблицы 6 можно заметить, что алгоритм транспонирования матрицы в формате COO, достаточно быстрый процесс. Увеличение количества нитей дает не большое улучшение времени, а с увеличением размера матрицы время увеличивается.

Из таблицы 7 видно, что ускорения как такового нет. Алгоритм транспонирования матрицы при хранении матрицы в формате COO, работает достаточно быстро и при последовательной реализации, поэтому параллельные версии алгоритмов, реализованных с помощью технологии OpenMP, имеет смысл использовать только при очень большом размере матрицы и большом количестве ненулевых элементов. Наглядно, ускорение алгоритма транспонирования матрицы в формате COO можно увидеть на рисунке 8. Эффективность, ожидаемо будет мала, так как ускорения нет, ее можно увидеть в таблице 8.

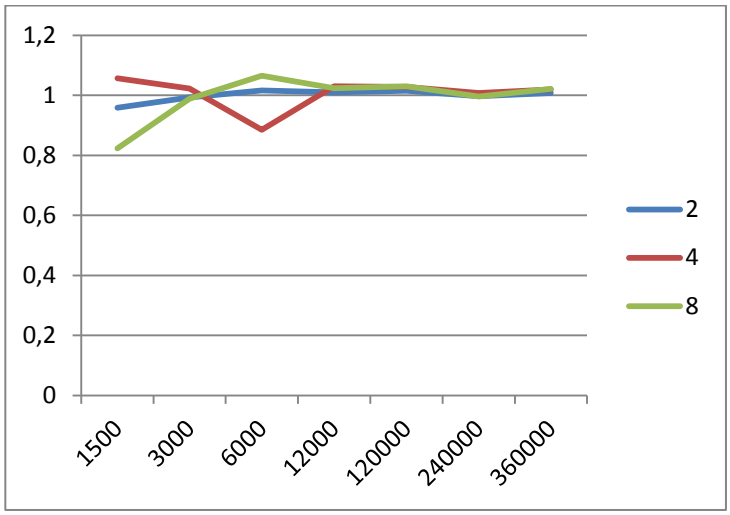

Рисунок 8 – график зависимости ускорения транспонирования матрицы в формате COO от количества нитей и размеров матрицы.

В таблице 8 видно, что эффективность параллельных алгоритмов низкая, как и ожидалось.

Таблица 6 – время работы алгоритма транспонирования матрицы в формате COO

	Кол-во нитей	Размер матрицы						
		1500	3000	6000	12000	120000	240000	360000
Транспонирование матрицы в COO формате	1	9,298e-5	1,760e-4	3,769e-4	9,711e-4	1,034e-2	2,060e-1	3,081e-2
	2	9,701e-5	1,771e-4	3,710e-4	9,610e-4	1,018e-2	2,061e-1	3,059e-2
	4	8,797e-5	1,721e-4	4,260e-4	9,429e-4	1,005e-2	2,041e-1	3,022e-2
	8	1,130e-4	1,780e-4	3,540e-4	9,489e-4	1,003e-2	2,063e-1	3,018e-2

Таблица 7 – ускорения алгоритма транспонирования матрицы в формате COO

		Размер матрицы						
Транспонирование матрицы в COO формате	Количество нитей	1500	3000	6000	12000	120000	240000	360000
	2	0,958	0,993	1,016	1,010	1,015	0,997	1,007
	4	1,057	1,022	0,885	1,030	1,028	1,007	1,019
	8	0,823	0,988	1,065	1,023	1,030	0,996	1,021

Таблица 8 – эффективность алгоритма транспонирования матрицы в формате COO

		Размер матрицы						
Транспонирование матрицы в COO формате	Количество нитей	1500	3000	6000	12000	120000	240000	360000
	2	0,479	0,497	0,508	0,505	0,507	0,498	0,504
	4	0,264	0,256	0,221	0,257	0,257	0,252	0,255
	8	0,103	0,123	0,133	0,128	0,129	0,125	0,128

Далее рассмотрим время работы, ускорение и эффективность алгоритмов умножения матрицы на матрицу в формате COO на 1, 2, 4 и 8 нитях.

Таблица 9 – время работы алгоритма умножения матрицы на матрицу в формате COO

Количество нитей	Размер матрицы			
	1500	3000	6000	12000
1	9,88e-2	4,03e-1	1,60e-0	6,39e-0
2	4,94e-2	2,02e-1	8,02e-1	3,20e-0
4	2,47e-2	1,01e-1	4,01e-1	1,61e-0
8	9,70e-2	5,50e-2	2,01e-1	8,02e-1

Из таблицы 9 видно, что алгоритм умножения матрицы на матрицу в формате COO более трудоемкий процесс, чем умножение на вектор и транспонирование матрицы в формате COO. Время алгоритма растет с увеличением размера матрицы и уменьшается с увеличением количества потоков. Ускорение алгоритмов можно увидеть в таблице 10.

Таблица 10 – ускорение алгоритма умножения матрицы на матрицу в формате COO

Количество нитей	Размер матрицы			
	1500	3000	6000	12000
2	1,977	1,996	1,998	1,997
4	3,954	3,991	3,996	3,994
8	1,019	7,327	7,971	7,988

Из таблицы 10 можно сделать вывод, что ускорение алгоритма увеличивается при увеличении количества нитей, и практически не изменяется при увеличении размера матрицы. Наглядно зависимость ускорения от размера матрицы и количества нитей можно увидеть на рисунке 9. Эффективность алгоритма умножения матрицы на матрицу можно увидеть в таблице 11.

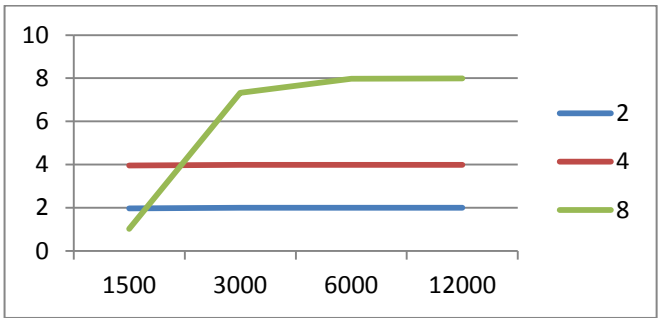

Рисунок 9 – график зависимости ускорения от количества нитей и размеров матрицы

Таблица 11 – эффективность алгоритма умножения матрицы на матрицу в формате COO

		Размер матрицы			
	Количество нитей	1500	3000	6000	12000
Умножение матрицы на матрицу в COO формате	2	0,980	0,998	0,999	0,998
	4	0,999	0,999	0,998	0,998
	8	0,125	0,998	0,998	0,994

Из таблицы 11 можно сделать вывод, что алгоритм умножения матрицы на матрицу в формате COO имеет хорошую эффективность при распараллеливании с помощью технологии OpenMP при 2,4 и 8 нитях.

Из всего выше сказанного можно сделать вывод, что алгоритмы умножения матрицы на вектор в формате COO работает достаточно быстро и при последовательном алгоритме, но при большом размере матрицы и большем количестве ненулевых элементов можно использовать параллельную реализацию алгоритма с помощью технологии OpenMP. Алгоритм транспонирования матрицы не требует использования технологии OpenMP, так как не имеет вычислительной нагрузки. Алгоритм умножения матрицы на матрицу в формате COO показал хорошее ускорение и эффективность.

2.3.2. Численные исследования работы алгоритмов с помощью технологии OpenMP для формата CRS

Далее рассмотрим время работы, ускорение и эффективность алгоритмов умножения матрицы на вектор в формате CRS на 1, 2, 4 и 8 нитях.

Таблица 12 – время работы алгоритма умножения матрицы на вектор в формате CRS

		Количество нитей			
Размер матрицы	Количество ненулевых элементов	1	2	4	8
1500	7542	2,111e-2	1,821e-2	1,773e-2	2,159e-2
3000	15042	1,014e-1	6,730e-2	2,55e-1	1,0432e-1
6000	30042	1,842e-2	2,513e-2	1,759e-2	1,728e-2
12000	60042	2,559e-2	2,746e-2	1,648e-2	2,798e-2
120000	600042	2,177e-2	1,785e-2	2,133e-2	1,606e-2
240000	1200042	1,701e-2	1,901e-2	1,804e-2	1,771e-2
360000	1800042	1,572e-2	1,841e-2	2,090e-2	2,278e-2
480000	2400042	2,520e-2	0,0324e-2	1,872e-2	2,557e-2
720000	3600042	2,352e-2	2,717e-2	2,408e-2	2,419e-2
960000	4800042	2,566e-2	3,205e-2	2,261e-2	2,615e-2

Таблица 13 – ускорения алгоритма умножения матрицы на вектор в формате CRS

		Количество нитей		
Размер матрицы	Количество ненулевых элементов	2	4	8
1500	7542	1,159	1,191	0,978
3000	15042	1,508	3,980	4,825
6000	30042	0,733	1,047	1,066
12000	60042	0,932	1,553	0,915
120000	600042	1,219	1,021	1,355
240000	1200042	0,911	0,961	0,978
360000	1800042	0,854	0,752	0,690
480000	2400042	0,778	1,346	0,985
720000	3600042	0,865	0,977	0,972
960000	4800042	0,801	1,135	0,982

Из таблицы 13 можно сделать вывод, что распараллеливание алгоритма умножения матрицы на вектор в формате CRSне совсем удачное, так как с увеличением количества нитей или с увеличением размера матрицы ускорение ведет себя не предсказуемо. Наглядно увидеть зависимость ускорения от размеров матрицы и количества нитей можно на рисунке 10. Эффективность алгоритма умножения матрицы на вектор можно увидеть в таблице 14.

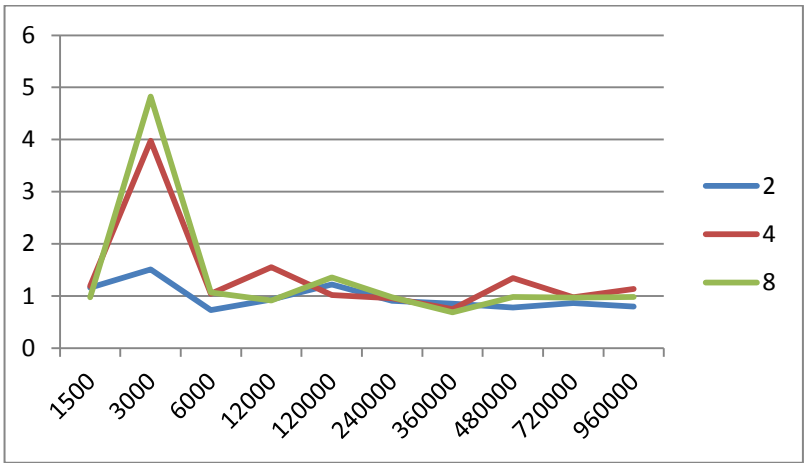

Рисунок 10 – график зависимости ускорения алгоритма умножения матрицы на вектор в формате CRS от количества нитей и размера матрицы.

Таблица 14-эффективность алгоритма умножения матрицы на вектор в формате CRS

		Количество нитей		
Размер матрицы	Количество ненулевых элементов	2	4	8
1500	7542	0,580	0,298	0,122
3000	15042	0,754	0,995	0,603
6000	30042	0,367	0,262	0,133
12000	60042	0,466	0,388	0,114
120000	600042	0,610	0,255	0,169
240000	1200042	0,456	0,240	0,122
360000	1800042	0,427	0,188	0,086
480000	2400042	0,389	0,337	0,123
720000	3600042	0,433	0,244	0,122
960000	4800042	0,580	0,298	0,122

Из таблицы 14 видно, что эффективность алгоритма умножения матрицы на вектор в формате CRS не увеличивается с увеличением количества нитей, а при увеличении размера матрицы находится примерно на одном уровне.

Далее рассмотрим время работы, ускорение и эффективность алгоритмов транспонирования матрицы в формате CRS на 1, 2, 4 и 8 нитях.

Таблица 15 – время работы алгоритма транспонирования матрицы в формате CRS

		Размер матрицы						
	Кол-во нитей	1500	3000	6000	12000	120000	240000	360000
Транспонирование в CRS формате	1	1,723e-2	6,792e-2	2,700e-1	1,008e-0	108,012e-0	503,030e-0	1203,024e-0
	2	1,001e-2	3,796e-2	1,490e-1	5,580e-1	60,678e-0	$291 \cdot 10^0$	701,329e-0
	4	5,010e-3	1,997e-2	7,779e-2	2,931e-1	31,649e-0	$161 \cdot 10^0$	463,141e-0
	8	3,040e-3	9,988e-3	4,000e-2	1,511e-1	16,375e-0	$101 \cdot 10^0$	308,104e-0

Таблица 16– ускорение алгоритма транспонирования матрицы в формате CRS

		Размер матрицы						
	Кол-во нитей	1500	3000	6000	12000	120000	240000	360000
Транспонирование в CRS формате	2	1,700	1,789	1,812	1,806	1,788	1,726	1,716
	4	3,400	3,400	3,462	3,439	3,429	3,131	2,598
	8	5,667	6,800	6,750	6,667	6,627	5,003	3,905

Из таблицы 15 можно сделать вывод, что процесс транспонирования матрицы в формате CRS достаточно трудоемкий и требует много времени в сравнении с умножением матрицы на вектор. Так же заметно, что время растет при увеличении размера матрицы, и уменьшается при увеличении количества нитей. В таблице 16 можно увидеть ускорение алгоритма, что поможет лучше оценить ситуацию.

Из таблицы 16 можно сделать вывод, что ускорение растет вместе с увеличением количества нитей, при этом не сильно изменяется при увеличении размера матрицы. Наглядно зависимость ускорения от размера матрицы и количества нитей можно увидеть на рисунке 11.

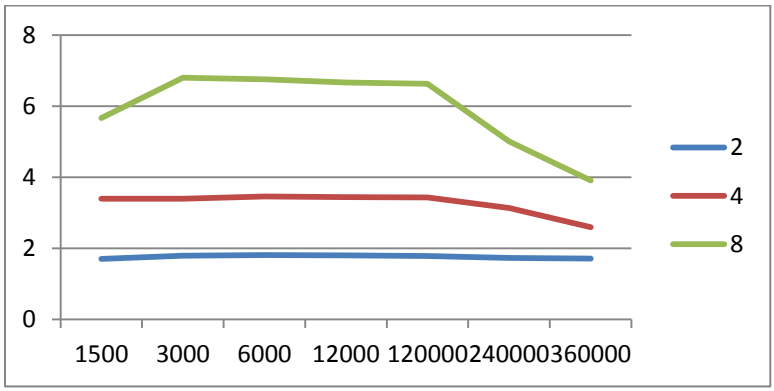

Рисунок 11 – график зависимости ускорения от размера матрицы и количества нитей

Далее рассмотри эффективность алгоритма транспонирования матрицы в формате CRS в таблице 17.

Таблица 17 – эффективность алгоритма транспонирования матрицы

		Размер матрицы						
	Кол-во нитей	1500	3000	6000	12000	120000	240000	360000
Транспонирование в CRS формате	2	0,850	0,895	0,906	0,903	0,894	0,863	0,858
	4	0,850	0,850	0,866	0,860	0,857	0,783	0,650
	8	0,708	0,850	0,844	0,833	0,828	0,625	0,488

Из таблицы 17 можно сделать вывод, что эффективность алгоритма транспонирования матрицы в формате CRS немного снижается при увеличении количества потоков.

Далее рассмотрим время работы, ускорение и эффективность алгоритмов умножения матрицы на матрицу в формате CRS на 1, 2, 4 и 8 нитях.

Из таблицы 18 видно, что алгоритм умножения матрицы на матрицу в формате CRS, выполняется быстрее, чем транспонирование матрицы в том

же формате. Время работы алгоритма увеличивается с увеличением работы матрицы. Так же время алгоритма уменьшается при увеличении количества нитей, что говорит о хорошем ускорении алгоритма. Ускорение алгоритма можно увидеть в таблице 19.

Таблица 18– время работы алгоритма умножения матрицы на матрицу в формате CRS

		Размер матрицы			
	Кол-во нитей	1500	3000	6000	12000
Умножение матрицы на матрицу в CRS формате	1	1,050e-1	4,138e-1	1,647e-0	6,536e-0
	2	5,252e-2	2,076e-1	8,277e-1	3,360e-0
	4	2,646e-2	1,037e-1	4,002e-1	1,638e-0
	8	1,339e-2	5,225e-2	2,063e-1	8,239e-1

Таблица 19 – ускорение алгоритма умножения матрицы на матрицу в формате CRS

		Размер матрицы			
	Количество нитей	1500	3000	6000	12000
Умножение матрицы на матрицу в CRS формате	2	1,982	1,993	1,989	1,945
	4	3,969	3,988	3,906	3,990
	8	7,844	7,920	7,979	7,933

Из таблицы 19 видно, что ускорение алгоритма умножения матрицы на матрицу растет при увеличении числа нитей, при этом находится почти на одном уровне при увеличении размера матрицы. Наглядно ускорение алгоритма можно увидеть на рисунке 12.

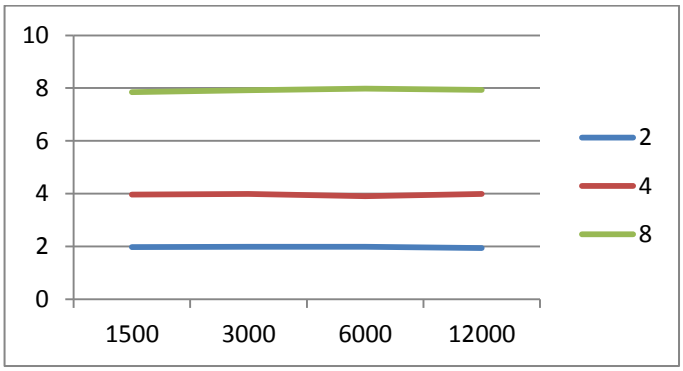

Рисунок 12 – график зависимости ускорения алгоритма умножения матрицы на матрицу в формате CRS от размера матрицы и количества нитей

Далее рассмотри эффективность алгоритма транспонирования матрицы в формате CRS в таблице 20.

Таблица 20 – эффективность алгоритма умножения матрицы на матрицу в формате CRS

		Размер матрицы			
	Количество нитей	1500	3000	6000	12000
Умножение матрицы на матрицу в CRS формате	2	0,991	0,997	0,995	0,973
	4	0,992	0,997	0,977	0,998
	8	0,981	0,990	0,997	0,992

Из таблицы 20 видно, что алгоритм умножения матрицы на матрицу в формате CRS эффективен при любом количестве нитей, и практически не меняется от размера матрицы.

Из всего выше сказанного можно сделать вывод, что алгоритмы умножения матрицы на вектор в формате CRS работает достаточно быстро и при последовательном алгоритме, но при более большом размере матрицы и большем количестве ненулевых элементов можно использовать параллельную реализацию алгоритма с помощью технологии OpenMP. Алгоритм транспонирования матрицы достаточно трудоемкий процесс, который при использовании технологии OpenMP показывает более хорошее время работы, ускорение и эффективность. Алгоритм умножения матрицы на матрицу в формате CRS показал хорошее ускорение и эффективность при использовании OpenMP.

3 Численные эксперименты над коллекцией разреженных матриц возникающих в реальных задачах

Сравнивать время работы алгоритмов будем на матрицах описанных ниже, в пункте 3.1. В пункте 9.3. для параллельных алгоритмов будем сравнивать время работы алгоритма при разных количествах потоков. Для параллельных алгоритмов будем рассчитывать эффективность по формуле (8). Так же рассчитаем ускорение по формуле (7).

3.1 Описание коллекции матриц

Для численных экспериментов постараемся подобрать матрицы со схожим портретом, но разными размерами. Каждый набор таких матриц будет называть коллекцией матриц.

3.1.1. Первая коллекция матриц

В этой коллекции будут матрицы Trefethen_2000 и Trefethen_500, они имеют схожий портрет и отличаются размерностью в 4 раза. Портреты этих матриц совпадают, их портрет можно увидеть на рисунке 13. Более подробная информация о характеристиках этих матриц будет содержаться в таблице 1. Эти матрицы возникают при решении комбинаторной проблемы (CombinatorialProblem).

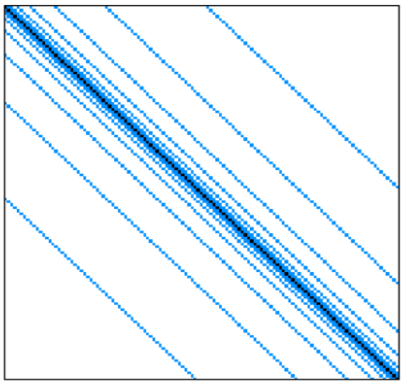

Рисунок 13 – портрет матрицы Trefethen_2000 и матрицы Trefethen_500

3.1.2. Вторая коллекция матриц

В этой коллекции будут матрицы SciMet и SmaGri, они имеют схожий портрет и отличаются размерностью в 3 раза. Портрет матрицы SciMet можно увидеть на рисунке 14, а портрет матрицы SmaGri на рисунке 15. Более подробная информация о характеристиках этих матриц будет содержаться в таблице 1. Эти матрицы возникают при решении проблемы направленного мульти графа (DirectedMultigraph).

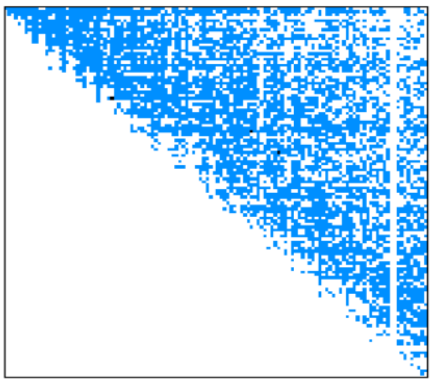

Рисунок 14- потрет матрицы SciMet

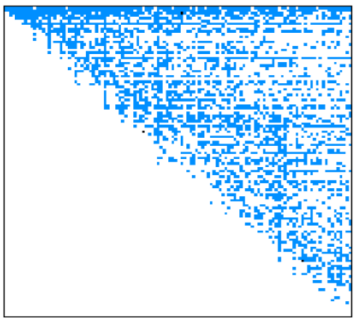

Рисунок 15 – портрет матрицы SmaGri

3.1.3 Третья коллекция матриц

В третью коллекцию матриц возьмем матрицу geom. Она возникает при решении проблемы направленного взвешенного графа(UndirectedWeightedGraph). Потрет матрицы можно увидеть на рисунке 16. Как можно видеть по портрету, ненулевые элементы в этой матрице разбросаны по всей матрице, что и отличает ее от двух предыдущих коллекций. Более подробную информацию о характеристиках матрицы можно найти в таблице 1.

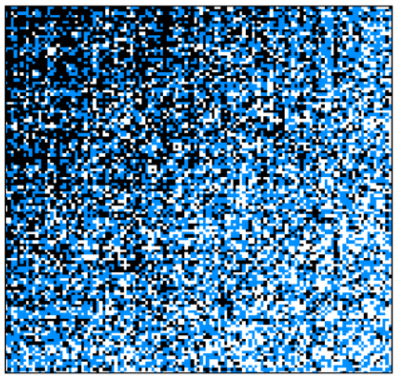

Рисунок 16 – портрет матрицы geom

3.2. Результат расчетов

Будем сравнивать последовательные алгоритмы для разных форматов хранения матриц, на различных матрицах, по времени работе алгоритма.

Таблица 21 – сравнение времени работы последовательных алгоритмов на различных матрицах

Характеристики матриц	Название матриц				
	Thefethen_500	Thefethen_2000	SmaGri	SciMet	geom
Размер матрицы	500x500	2000x2000	1059x1059	3084x3084	7343x7343
Количество ненулевых элементов	8479	41906	4919	10413	23796
Память в формате двумерного массива(мб)	1,90	30,5	8,60	72,6	411,30
Память в формате COO(мб)	0,19	0,96	0,11	0,24	0,54
Память в формате CRS(мб)	0,05	0,65	0,08	0,18	0,42
Название операций	Время операции				
Умножение на вектор в классическом формате	5,39e-4	5,01e-3	2,56e-3	1,97e-2	1,12e-1
Умножение на вектор в COO формате	1,46e-5	8,79e-5	1,59e-5	4,88e-5	6,91e-5
Умножение на вектор в CRS формате	1,31e-5	7,20e-5	2,98e-5	1,18e-4	1,42e-4
Транспонирование в классическом формате	1,66e-3	4,59e-2	1,02e-2	2,12e-1	6,50e-1
Транспонирование в COO формате	5,20e-5	2,39e-4	5,29e-5	1,09e-5	1,25e-4
Транспонирование в CRS формате	9,74e-3	6,49e-2	7,48e-3	4,50e-2	1,08e-1
Умножение матрицы на матрицу в классическом формате	7,84e-1	87,53e-0	10,18e-0	327,87e-0	4503,77e-0
Умножение матрицы на матрицу в COO формате	2,65e-2	6,24e-1	3,22e-2	1,42e-1	1,84e-1

Окончание таблицы 21

Умножение матрицы на матрицу в CRS формате	3,89e-2	8,73e-1	1,17e-1	6,13e-1	1,12e-0

Из таблицы 21 можно сделать вывод, что форматы хранения разреженных матриц на практике занимают значительно меньше памяти при хранении матриц, быстрее выполняют операции над матрицами, как и на подконтрольной матрице.

3.3 Результат расчетов с помощью технологии OpenMP

3.3.1 Результаты расчетов с помощью технологии OpenMP для первой коллекции матриц

Рассмотри численные результаты времени работы алгоритмов на матрице Thefethen_500 в таблице 22 и таблице 23.

Таблица 22 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице Thefethen_500 в формате COO

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате COO на 1 нити	2,40e-4	-	-
Умножение матрицы на вектор в формате COO на 2 нитях	3,70e-3	0,640	0,320
Умножение матрицы на вектор в формате COO на 4 нитях	9,30e-3	0,260	0,065
Умножение матрицы на вектор в формате COO на 8 нитях	2,03e-2	0,011	0,001
Транспонирование матрицы в формате COO на 1 нити	5,20e-5	-	-
Транспонирование матрицы в формате COO на 2 нитях	6,32e-5	0,820	0,410
Транспонирование матрицы в формате COO на 4 нитях	5,70e-5	0,913	0,228
Транспонирование матрицы в формате COO на 8 нитях	4,82e-5	1,080	0,135
Умножение матрицы на матрицу в формате COO на 1 нити	3,52e-2	-	-
Умножение матрицы на матрицу в формате COO на 2 нитях	1,76e-2	1,900	0,95
Умножение матрицы на матрицу в формате COO на 4 нитях	2,65e-2	1,750	0,438

Окончание таблицы 22

Умножение матрицы на матрицу в формате COO на 8нитях	4,42e-3	7,950	0,994

Таблица 23 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице Thefethen_500 в формате CRS

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате CRS на 1 нити	1,22e-2	-	-
Умножение матрицы на вектор в формате CRS на 2 нитях	1,72e-2	0,707	0,354
Умножение матрицы на вектор в формате CRS на 4 нитях	2,50e-2	0,487	0,122
Умножение матрицы на вектор в формате CRS на 8 нитях	2,49e-2	0,491	0,061
Транспонирование матрицы в формате CRS на 1 нити	7,90e-2	-	-
Транспонирование матрицы в формате CRS на 2 нитях	6,16e-3	1,282	0,641
Транспонирование матрицы в формате CRS на 4 нитях	5,04e-3	1,567	0,392
Транспонирование матрицы в формате CRS на 8 нитях	4,38e-3	1,804	0,226
Умножение матрицы на матрицу в формате CRS на 1 нити	3,68e-2	-	-
Умножение матрицы на матрицу в формате CRS на 2 нитях	2,04e-2	1,800	0,900
Умножение матрицы на матрицу в формате CRS на 4 нитях	1,02e-2	3,596	0,899
Умножение матрицы на матрицу в формате CRS на 8нитях	5,19e-3	7,094	0,887

Из таблиц 22 и 23 видно, что формат COO считает быстрее формата CRS, однако формат CRSимеет большее ускорение и эффективность для операций умножения матрицы на вектор и транспонирования, и примерно такие же ускорение и эффективность для операции умножение матрицы на матрицу. Ускорение и эффективность алгоритмов для формата COO на матрице Thefethen_500 схоже с результатами из пункта 2. Ускорение и эффективность алгоритмов для формата CRS немного меньше чем из пункта

2, за исключением алгоритма умножения матрицы на матрицу, но и матрица здесь меньших размеров. Наглядно зависимость ускорения от количества потоков можно увидеть на рисунке17 для формата COOи на рисунке 18 для формата CRS.

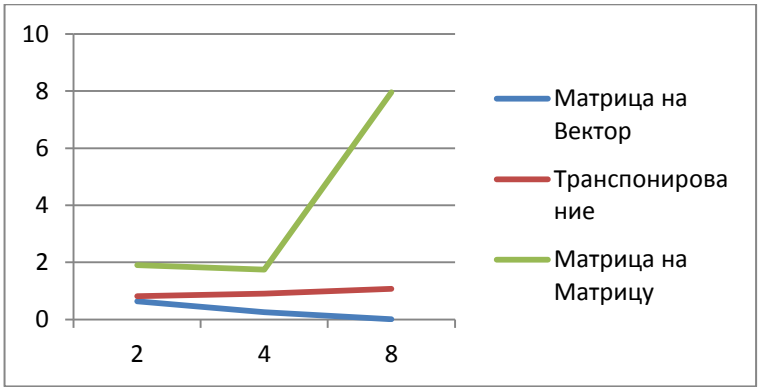

Рисунок 17 – график зависимости ускорения алгоритмов в формате COO от количества нитей

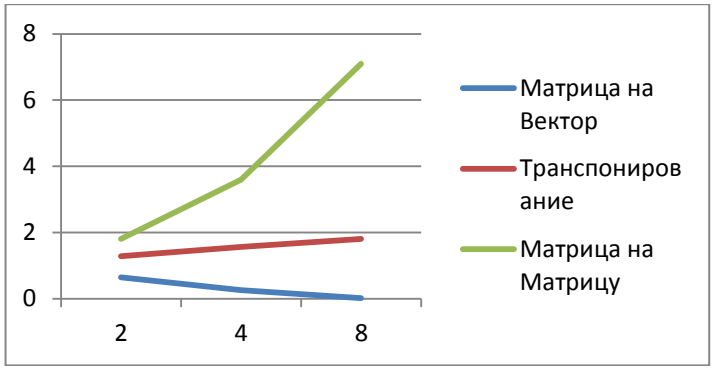

Рисунок 18 – график зависимости ускорения алгоритмов в формате CRS от количества нитей

Рассмотрим численные результаты времени работы алгоритмов на матрице Thefethen_2000 в таблице 24 и таблице 25.

Таблица 24 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице Thefethen_2000 в формате COO

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате COO на 1 нити	3,54e-4	-	-
Умножение матрицы на вектор в формате COO на 2 нитях	3,76e-3	0,094	0,047
Умножение матрицы на вектор в формате COO на 4 нитях	1,05e-2	0,033	0,008

Продолжение таблицы 24

Умножение матрицы на вектор в формате COO на 8 нитях	1,55e-2	0,228	0,029
Транспонирование матрицы в формате COO на 1 нити	2,43e-4	-	-
Транспонирование матрицы в формате COO на 2 нитях	2,57e-4	0,946	0,473
Транспонирование матрицы в формате COO на 4 нитях	2,62e-4	0,927	0,232
Транспонирование матрицы в формате COO на 8 нитях	2,18e-4	1,115	0,139
Умножение матрицы на матрицу в формате COO на 1 нити	8,59e-1	-	-
Умножение матрицы на матрицу в формате COO на 2 нитях	4,31e-1	1,995	0,998
Умножение матрицы на матрицу в формате COO на 4 нитях	2,23e-1	3,811	0,953
Умножение матрицы на матрицу в формате COO на 8нитях	1,13e-1	7,610	0,951

Таблица 25 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице Thefethen_2000 в формате CRS

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате CRS на 1 нити	3,54e-4	-	-
Умножение матрицы на вектор в формате CRS на 2 нитях	1,96e-2	0,018	0,009
Умножение матрицы на вектор в формате CRS на 4 нитях	1,95e-2	0,018	0,005
Умножение матрицы на вектор в формате CRS на 8 нитях	2,16e-2	0,016	0,002
Транспонирование матрицы в формате CRS на 1 нити	1,34e-1	-	-
Транспонирование матрицы в формате CRS на 2 нитях	1,04e-1	1,287	0,644
Транспонирование матрицы в формате CRS на 4 нитях	8,13e-2	1,608	0,402

Продолжение таблицы 25

Транспонирование матрицы в формате CRS на 8 нитях	7,32e-2	1,825	0,228
Умножение матрицы на матрицу в формате CRS на 1 нити	8,59e-1	-	-
Умножение матрицы на матрицу в формате CRS на 2 нитях	4,42e-1	1,942	0,971
Умножение матрицы на матрицу в формате CRS на 4 нитях	2,21e-1	3,891	0,973
Умножение матрицы на матрицу в формате CRS на 8нитях	1,11e-1	7,756	0,969

Из таблиц 24 и 25 видно, что они подтверждают вывод из таблиц 22 и 23. При увеличении размера матрицы операция умножения матрицы на вектор получает более высокие показатели ускорения и эффективности, операция транспонирования так же получает большие показатели ускорения и эффективности, операция умножения матрицы на матрицу получает чуть меньшее значение ускорения и эффективности для 8 нитей. Наглядно увидеть зависимость ускорения от количества потоков можно на рисунке 19 для формата COOи на рисунке 20 для формата CRS.

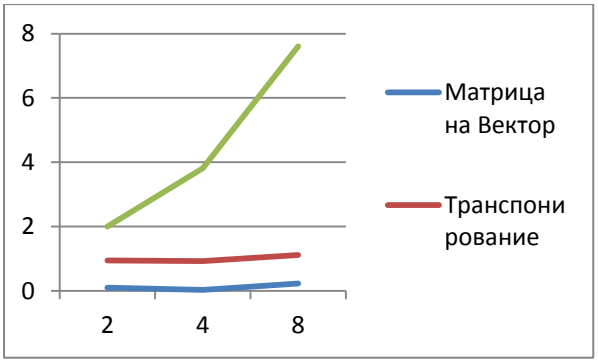

Рисунок 19 - график зависимости ускорения алгоритмов в формате COO от количества нитей

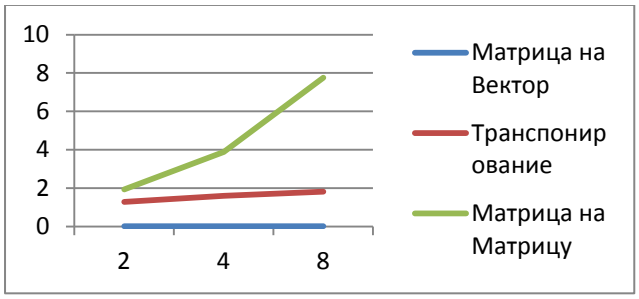

Рисунок 20 - график зависимости ускорения алгоритмов в формате CRS от количества нитей

3.3.2 Результаты расчетов с помощью технологии OpenMP для второй коллекции матриц

Рассмотри численные результаты времени работы алгоритмов на матрице SmaGri в таблице 26 и таблице 27

Таблица 26 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице SmaGri в формате COO

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате COO на 1 нити	2,00e-4	-	-
Умножение матрицы на вектор в формате COO на 2 нитях	3,67e-3	0,056	0,028
Умножение матрицы на вектор в формате COO на 4 нитях	1,24e-2	0,016	0,004
Умножение матрицы на вектор в формате COO на 8 нитях	2,02e-2	0,011	0,001
Транспонирование матрицы в формате COO на 1 нити	7,00e-5	-	-
Транспонирование матрицы в формате COO на 2 нитях	8,20e-5	0,854	0,427
Транспонирование матрицы в формате COO на 4 нитях	3,91e-5	1,790	0,448
Транспонирование матрицы в формате COO на 8 нитях	5,89e-5	1,189	0,149
Умножение матрицы на матрицу в формате COO на 1 нити	4,25e-2	-	-
Умножение матрицы на матрицу в формате COO на 2 нитях	2,13e-2	1,992	0,996
Умножение матрицы на матрицу в формате COO на 4 нитях	1,06e-2	3,988	0,997
Умножение матрицы на матрицу в формате COO на 8нитях	5,34e-3	7,959	0,995

Таблица 27 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице SmaGri в формате CRS

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате CRS на 1 нити	1,69e-2	-	-
Умножение матрицы на вектор в формате CRS на 2 нитях	1,92e-2	0,878	0,439
Умножение матрицы на вектор в формате CRS на 4 нитях	2,06e-2	0,818	0,204
Умножение матрицы на вектор в формате CRS на 8 нитях	2,14e-2	0,788	0,098
Транспонирование матрицы в формате CRS на 1 нити	1,69e-2	-	-
Транспонирование матрицы в формате CRS на 2 нитях	1,33e-2	1,271	0,635
Транспонирование матрицы в формате CRS на 4 нитях	1,09e-2	1,554	0,388
Транспонирование матрицы в формате CRS на 8 нитях	9,69e-3	1,743	0,218
Умножение матрицы на матрицу в формате CRS на 1 нити	1,17e-1	-	-
Умножение матрицы на матрицу в формате CRS на 2 нитя	6,82e-2	1,709	0,854
Умножение матрицы на матрицу в формате CRS на 4 нитях	3,71e-2	3,142	0,785
Умножение матрицы на матрицу в формате CRS на 8нитях	2,05e-2	5,691	0,711

Из таблиц 26 и 27 видно, что формат COO считает быстрее формата CRS, несмотря на изменения структуры матрицы. Формат CRS показывает более высокое значение ускорения и эффективности для алгоритма умножения матрицы на вектор и алгоритма транспонирования матрицы. Для алгоритма умножения матрицы на матрицу формат COO оказался быстрее и показывает более высокие значения ускорения и эффективности. Наглядно увидеть зависимость ускорения от количества нитей можно на рисунке 21 для формата COO и на рисунке 22 для формата CRS.

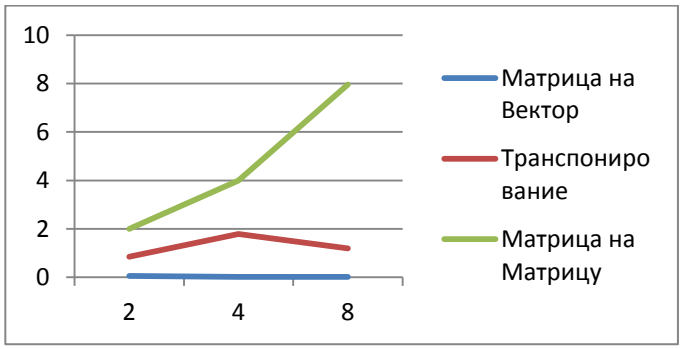

Рисунок 21 - график зависимости ускорения алгоритмов в формате COO от количества нитей

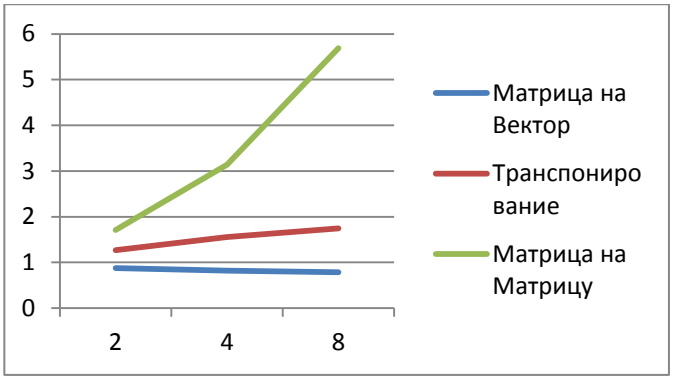

Рисунок 22 - график зависимости ускорения алгоритмов в формате CRS от количества нитей

Рассмотри численные результаты времени работы алгоритмов на матрице SciMet в таблице 28 и таблице 29

Таблица 28 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице SciMet в формате COO

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате COO на 1 нити	2,47e-4	-	-
Умножение матрицы на вектор в формате COO на 2 нитях	5,58e-4	0,443	0,221
Умножение матрицы на вектор в формате COO на 4 нитях	9,34e-3	0,026	0,007
Умножение матрицы на вектор в формате COO на 8 нитях	2,06e-2	0,012	0,002
Транспонирование матрицы в формате COO на 1 нити	1,04e-4	-	-

Продолжение таблицы 28

Транспонирование матрицы в формате COO на 2 нитях	1,05e-4	0,990	0,495
Транспонирование матрицы в формате COO на 4 нитях	1,13e-4	0,919	0,230
Транспонирование матрицы в формате COO на 8 нитях	1,06e-4	0,981	0,123
Умножение матрицы на матрицу в формате COO на 1 нити	1,95e-1	-	-
Умножение матрицы на матрицу в формате COO на 2 нитях	9,82e-2	1,984	0,991
Умножение матрицы на матрицу в формате COO на 4 нитях	4,88e-2	3,986	0,996
Умножение матрицы на матрицу в формате COO на 8нитях	2,44e-2	7,976	0,996

Таблица 29 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице SciMet в формате CRS

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате CRS на 1 нити	2,10e-2	-	-
Умножение матрицы на вектор в формате CRS на 2 нитях	2,13e-2	0,982	0,491
Умножение матрицы на вектор в формате CRS на 4 нитях	2,17e-2	0,969	0,242
Умножение матрицы на вектор в формате CRS на 8 нитях	2,09e-2	1,005	0,126
Транспонирование матрицы в формате CRS на 1 нити	9,72e-2	-	-
Транспонирование матрицы в формате CRS на 2 нитях	7,51e-2	1,294	0,647
Транспонирование матрицы в формате CRS на 4 нитях	5,99e-2	1,621	0,405
Транспонирование матрицы в формате CRS на 8 нитях	5,29e-2	1,837	0,230

Продолжение таблицы 29

Умножение матрицы на матрицу в формате CRS на 1 нити	6,48e-1	-	-
Умножение матрицы на матрицу в формате CRS на 2 нитях	3,80e-1	1,704	0,852
Умножение матрицы на матрицу в формате CRS на 4 нитях	1,99e-1	3,252	0,813
Умножение матрицы на матрицу в формате CRS на 8нитях	1,03e-1	6,294	0,787

Из таблиц 28 и 29 видно, что при увеличении размера матрицы формат COO так же считает быстрее формата CRS. Формат CRS показывает более высокое значение ускорения и эффективности для алгоритма умножения матрицы на вектор и алгоритма транспонирования матрицы. Для алгоритма умножения матрицы на матрицу формат COO оказался быстрее и показывает более высокие значения ускорения и эффективности. Наглядно увидеть зависимость ускорения от количества нитей можно на рисунке 23 для формата COO и на рисунке 24 для формата CRS.

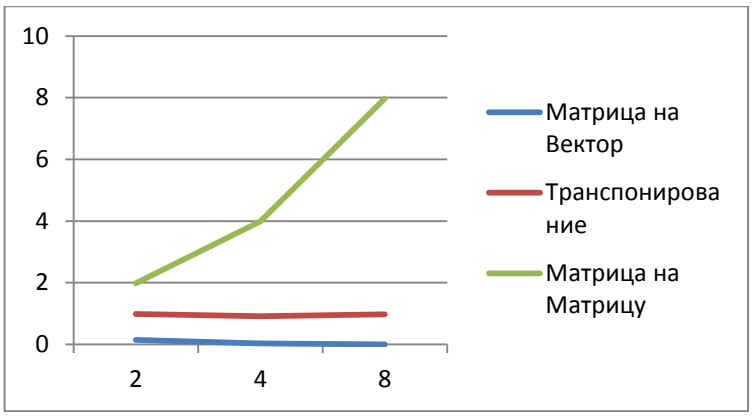

Рисунок 23 - график зависимости ускорения алгоритмов в формате COO от количества нитей

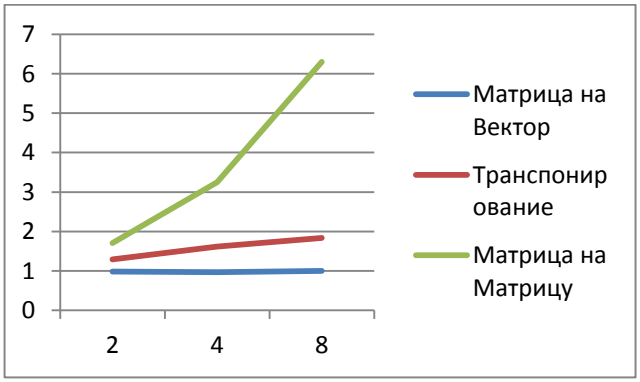

Рисунок 24 - график зависимости ускорения алгоритмов в формате CRS от количества нитей

3.3.3 Результаты расчетов с помощью технологии OpenMP для третей коллекции матриц

Рассмотри численные результаты времени работы алгоритмов на матрице geom в таблице 30 и таблице 31

Таблица 30 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице geom в формате COO

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате COO на 1 нити	3,40e-4	-	-
Умножение матрицы на вектор в формате COO на 2 нитях	2,23e-3	0,153	0,076
Умножение матрицы на вектор в формате COO на 4 нитях	1,08e-2	0,032	0,008
Умножение матрицы на вектор в формате COO на 8 нитях	4,72e-2	0,007	0,001
Транспонирование матрицы в формате COO на 1 нити	1,43e-4	-	-
Транспонирование матрицы в формате COO на 2 нитях	1,38e-4	1,036	0,518
Транспонирование матрицы в формате COO на 4 нитях	1,30e-4	1,101	0,275
Транспонирование матрицы в формате COO на 8 нитях	1,18e-4	1,212	0,152
Умножение матрицы на матрицу в формате COO на 1 нити	2,51e-1	-	-
Умножение матрицы на матрицу в формате COO на 2 нитях	1,35e-1	1,984	0,992
Умножение матрицы на матрицу в формате COO на 4 нитях	6,34e-1	0,396	0,098
Умножение матрицы на матрицу в формате COO на 8нитях	3,16e-2	7,933	0,991

Таблица 31 – Время, ускорение и эффективность алгоритмов с технологией OpenMP на матрице geom в формате CRS

	Время	Ускорение	эффективность
Умножение матрицы на вектор в формате CRS на 1 нити	1,70e-2	-	-
Умножение матрицы на вектор в формате CRS на 2 нитях	1,52e-2	1,122	0,561
Умножение матрицы на вектор в формате CRS на 4 нитях	2,56e-2	0,666	0,167
Умножение матрицы на вектор в формате CRS на 8 нитях	1,70e-2	1,004	0,126
Транспонирование матрицы в формате CRS на 1 нити	2,39e-1	-	-
Транспонирование матрицы в формате CRS на 2 нитях	1,87e-1	1,280	0,640
Транспонирование матрицы в формате CRS на 4 нитях	1,55e-1	1,543	0,386
Транспонирование матрицы в формате CRS на 8 нитях	1,38e-1	1,730	0,216
Умножение матрицы на матрицу в формате CRS на 1 нити	1,17e-1	-	-
Умножение матрицы на матрицу в формате CRS на 2 нитях	9,87e-1	1,202	0,601
Умножение матрицы на матрицу в формате CRS на 4 нитях	6,40e-1	1,853	0,463
Умножение матрицы на матрицу в формате CRS на 8нитях	3,70e-1	3,203	0,401

Из таблиц 30 и 31 видно, что формат COO производит вычисления по-прежнему быстрее формата CRS. Формат CRS показывает большие значения ускорения и эффективности для алгоритмов умножения матрицы на вектор и транспонирования матрицы. Для алгоритма умножения матрицы на матрицу формат COO показывает более высокое значение ускорения и эффективности на всех нитях, за исключением 4. Наглядно увидеть зависимость ускорения от количества нитей можно на рисунке 25 для формата COO и на рисунке 26 для формата CRS.

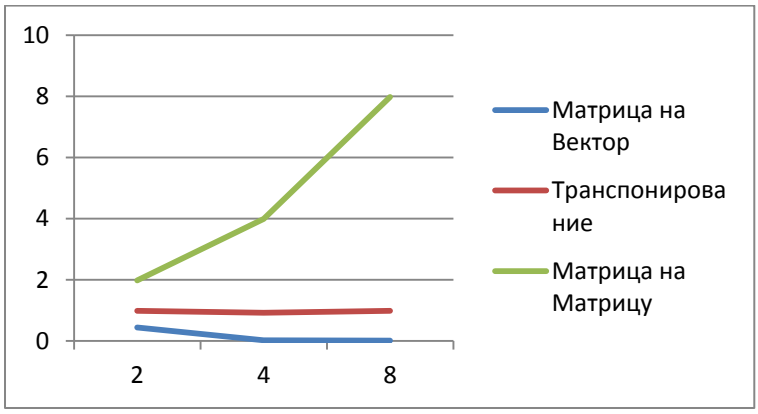

Рисунок 25 - график зависимости ускорения алгоритмов в формате COO от количества нитей

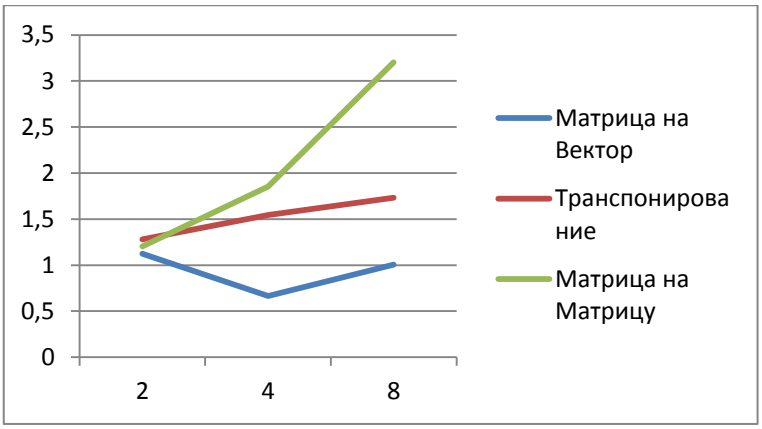

Рисунок 26 - график зависимости ускорения алгоритмов в формате CRS от количества нитей

Из всего выше сказанного можно сделать вывод, что алгоритмы для форматов для хранения разреженных матриц COO и CRS, показывают примерно такие же результаты в реальных задачах, как и для задач с заданной матрицей. Так же можно заметить, что структура матрицы не имеет большого значения для этих форматов.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы были реализованы на языке Си++ алгоритмы умножения матрицы на вектор, транспонирования матрицы, умножения матрицы на матрицу для специальных форматов хранения разреженных матриц COO и CRS. Хранение матриц в форматах COO и CRS требуют меньше памяти для хранения матриц, чем обычное хранение матрицы в виде двумерного массива. Алгоритмы для операций над матрицами в форматах COO и CRS требует меньшее время, чем алгоритмы для формата хранения матрицы в двумерном массиве, поскольку не обрабатывают нулевые элементы.

Произведено исследование об эффективности и ускорении параллельных алгоритмов, реализованных с помощью технологии OpenMP. В результате можно указать следующее.

1. Алгоритм умножения матрицы на вектор при хранении матрицы как в формате COO, так и в формате CRS работает достаточно быстро и при последовательной реализации, поэтому параллельные версии алгоритмов, реализованных с помощью технологии OpenMP, имеет смысл использовать только при очень большом размере матрицы и большом количестве ненулевых элементов. При этом надо понимать, что максимальное ускорение достигается уже при небольшом количестве нитей.

2. Алгоритм транспонирования матрицы при хранении матрицы в формате COO работает достаточно быстро и при последовательной реализации, поэтому параллельные версии алгоритмов, реализованных с помощью технологии OpenMP, имеет смысл использовать только при очень большом размере матрицы и большом количестве ненулевых элементов.

В свою очередь алгоритм транспонирования матрицы, хранящейся в формате CRS, достаточно трудоемкий процесс, который при использовании технологии OpenMP показывает хорошие ускорение и эффективность.

3. Параллельные версии алгоритмов умножения матрицы на матрицу при хранении матриц в обоих форматах COO и CRS показало практически линейное ускорении и эффективность близкую к 1, т.е. оптимальную.

4. Исследование об эффективности и ускорении параллельных алгоритмов, реализованных с помощью технологии OpenMP на матрицах, встречающихся в реальных задачах, подтверждают результаты, полученные при исследовании на заданной матрице.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений – Москва:Издательство «Мир», 1984. – 338 с.

2 Писсанецки С. Технология разреженных матриц. — Москва: Издательство «Мир»,1988.– 410 с.

3Тьюарсон Р.П. Разреженные матрицы – Москва: Издательство «Мир», 1977. – 191 с.

4 Эстербв О., Златев 3. Прямые методы для Разреженных матриц – Москва: Издательство «Мир», 1987.– 120 с.

5 Голуб Дж., Ван Лоун Ч.Матричные вычисления– Москва: Издательство «Мир»,1999. – 278с.

6 SuiteSparseMatrixCollection [Электронный ресурс]: Коллекция разреженных матриц университета Флориды – Режим доступа: https://sparse.tamu.edu.

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ В.В. Шайдуров
«29» июня 2020 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01Математика и компьютерные науки

ЭФФЕКТИВНАЯ ПАРАЛЛЕЛИЗАЦИЯ КОДА В ОПЕРАЦИЯХ С РАЗРЕЖЕННЫМИ МАТРИЦАМИ

Научный руководитель,
кандидат физико-математических наук,
доцент _____ Е.Д.Карепова

Выпускник _____ Н.А.Назаров

Красноярск 2020