МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: Фундаментальная информатика и информационные технологии

**Отчет по лабораторной работе**

Тема:

**«Умножение разреженных матриц. Элементы типа double. Формат хранения матрицы – столбцовый (CCS)»**

**Выполнил:** студент группы 381706-4

Пушкарев Дмитрий Сергеевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

**Проверил:** доцент кафедры МОСТ,

Сысоев Александр Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

Нижний Новгород

2020

Содержание

[Введение 2](#_Toc43126072)

[Постановка задачи 3](#_Toc43126073)

[Формат хранения 4](#_Toc43126074)

[Разреженный столбцовый формат(CCS) 4](#_Toc43126075)

[Разреженный строчный формат(CRS) 5](#_Toc43126076)

[Метод реализации 6](#_Toc43126077)

[Транспонирование матрицы 6](#_Toc43126078)

[Умножение матриц 6](#_Toc43126079)

[Распараллеливание 7](#_Toc43126080)

[Описание программной реализации 8](#_Toc43126081)

[Руководство пользователя 8](#_Toc43126082)

[Руководство программиста 9](#_Toc43126083)

[Результаты экспериментов 11](#_Toc43126084)

[Результаты замеров: 11](#_Toc43126085)

[Ускорение 12](#_Toc43126086)

[Заключение 14](#_Toc43126087)

[Литература 15](#_Toc43126088)

[Приложение 16](#_Toc43126089)

# Введение

Существует много понятий разреженной матрицы. Идея состоит в том, что в разреженной матрице большое число нулевых элементов. Обычно матрица размера N x N называется разреженной, если количество ее ненулевых элементов есть O(N). В противном случае матрица считается плотной. Также разреженной является матрица с ограничением числа ненулевых элементов в одной строке от 1 до k, где k значительно меньше N. Задачи линейной алгебры с разреженными матрицами часто возникают во многих инженерных и научных областях. Например, в теории графов, а также при численном решении дифференциальных уравнений в частных производных и при решении многомерных задач локальной оптимизации.

С любой разреженной матрицей можно работать как с плотной, и наоборот. При реализации алгоритмов для таких матриц в обоих случаях результаты будут корректны, но вычислительные затраты будут различны. Поэтому для многих математических методов необходимы специальные алгоритмы, использующие разреженность и делающие операции эффективнее, чем стандартные алгоритмы. В данной работе рассматривается умножение двух разреженных матриц, как один из распространенных и показательных алгоритмов.

# Постановка задачи

Необходимо программно реализовать параллельный алгоритм умножения разреженных матриц со столбцовым (CCS) форматом хранения с использованием технологий OpenMP и TBB. Затем сравнить эти технологии по времени выполнения реализованного алгоритма, сделав вывод об их эффективности.

# Формат хранения

Важно определиться со структурой хранения – использовать обычные двумерные массивы или сложную структуру, учитывающую разреженность. Иногда плотное представление матрицы попросту не убирается в память. Если же и разреженное, и плотное представление допустимы, а ненулевых элементов немного, необходимо проанализировать, какая структура будет более эффективна в каждом конкретном случае. Существуют различные форматы хранения разреженных матриц. Рассмотрим столбцовый и строчный форматы хранения.

## Разреженный столбцовый формат(CCS)

Для хранения матрицы требуется три одномерных массива:

* - массив ненулевых элементов матрицы , в котором они перечисляются в порядке их появления в столбцах матрицы;
* *rows* - массив номеров строк для соответствующих элементов массива ;
* - массив указателей позиций, с которых начинается описание столбца. Описание столбца хранится в позициях с по массивов и *rows*. Если , то строка пустая. Если матрица состоит из столбцов, то длина массива будет n+1.

Для примера рассмотрим представление матрицы в разреженном столбцовом формате:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | -1 | 0 | -3 | 0 |
|  | 5 | 0 | 0 | 0 |
|  |  | 4 |  | 4 |
|  |  |  | 7 |  |
|  |  |  | 0 | -5 |

Матрица А =

может быть представлена в разреженном столбцовом формате как набор векторов:

values = (1, -1, -3, -1, 5, 4, 6, 4, -3, 6, 7, 4, -5);

rows = (0, 1, 3, 0, 1, 2, 3, 4, 0, 2, 3, 2, 4);

pointer = (0, 3, 5, 8, 11, 12).

Объем памяти, требуемый для хранения вектора , значительно меньше, чем для хранения вектора . Более того, разреженный столбцовый формат обеспечивает эффективный доступ к столбцам матрицы; обращение к строкам матрицы затруднено. Поэтому предпочтительно использовать этот способ хранения в тех алгоритмах, в которых преобладают столбцовые операции.

Так как для реализации умножения требуются матрицы и в столбцовом и в строчном формате хранения, логично будет также рассмотреть ранее неописанный формат хранения – разреженный строчный.

## Разреженный строчный формат(CRS)

В этом случае ненулевые элементы матрицы перечисляются в порядке их появления в строках матрицы, а не в столбцах. Все ненулевые элементы хранятся по строкам в массиве ; индексы столбцов ненулевых элементов – в массиве ; элементы массива указывают на позиции, с которых начинается описание очередной строки.

Строчные представления могут рассматриваться как столбцовые представления транспонированных матриц. Разреженный строчный формат обеспечивает эффективный доступ к строкам матрицы; обращение к столбцам матрицы затруднено. Поэтому предпочтительно использовать этот способ хранения в тех алгоритмах, в которых преобладают строчные операции.

Рассмотрим строчный формат матрицы :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 0 | 0 | -3 | 0 |
|  | 5 | 0 | 0 | 0 |
|  |  | 4 |  | 4 |
|  |  |  | 7 |  |
|  |  |  | 0 | -5 |

Матрица А =

может быть представлена в разреженном строчном формате как набор векторов:

values = (1, -3, 5, 4, 6, 4, 1, 6, 7, 4, -5);

cols = (0, 3, 1, 2, 3, 4, 0, 2, 3, 2, 4);

pointer = (0, 2, 3, 6, 9, 10).

# Метод реализации

## Транспонирование матрицы

Рассмотрим задачу транспонирования разреженной матрицы . Матрица   может быть определена как .

Результирующую матрицу будем формировать построчно. Для этого будем брать столбцы исходной матрицы и создавать из них строки результирующей матрицы. Операция выделения из матрицы столбца является трудоемкой, т.к. данные в векторе хранятся по строкам и для выборки данных по столбцу нужно просмотреть всю матрицу, что приводит к квадратичной (от числа ненулевых элементов) трудоемкости алгоритма.

1. Сформируем вектор векторов пар для хранения целых и вещественных чисел. Размер вектора соответствует числу столбцов исходной матрицы.
2. В цикле просмотрим все строки исходной матрицы, для каждой строки – все ее элементы. Пусть текущий элемент находится в строке , столбце , его значение равно . Тогда добавим пары чисел и в вектора для хранения целых и вещественных чисел (соответственно). Тем самым в векторах сформируются строки транспонированной матрицы.
3. Последовательно скопируем данные из векторов в CRS структуру транспонированной матрицы ( и ), попутно формируя массив .

Алгоритм транспонирует матрицу за линейное время.

## Умножение матриц

Для умножения матриц и в столбцовом формате преобразуем матрицу в строчный формат, используя транспонирование. Заведем указатели и указывающие на матрицы и матрицы . Пусть – номер столбца ненулевого элемента матрицы , а – номер строки ненулевого элемента матрицы .

1. Если то умножаем элементы и накапливаем их результат в соответствующем скалярном произведении;
2. иначе если , то сдвигаем на следующий элемент, переходим к п.1;
3. иначе если то сдвигаем на следующий элемент, переходим к п1;
4. если скалярное произведение не равно , то записываем его в результирующую матрицу;
5. переходим на следующий столбец/строку.

# Распараллеливание

В параллельном алгоритме матрица делится на столбцы. Каждый поток умножает всю матрицу на несколько своих столбцов матрицы и получает часть результата в свою собственную матрицу. В конце все полученные матрицы последовательно объединятся в одну результирующую матрицу.

Каждый поток определяет, необходимые столбцы матрицы с помощью задачи (структура ), в которой указано с какого столбца по какой столбец нужно использовать из матрицы и в какую матрицу нужно записать результат.

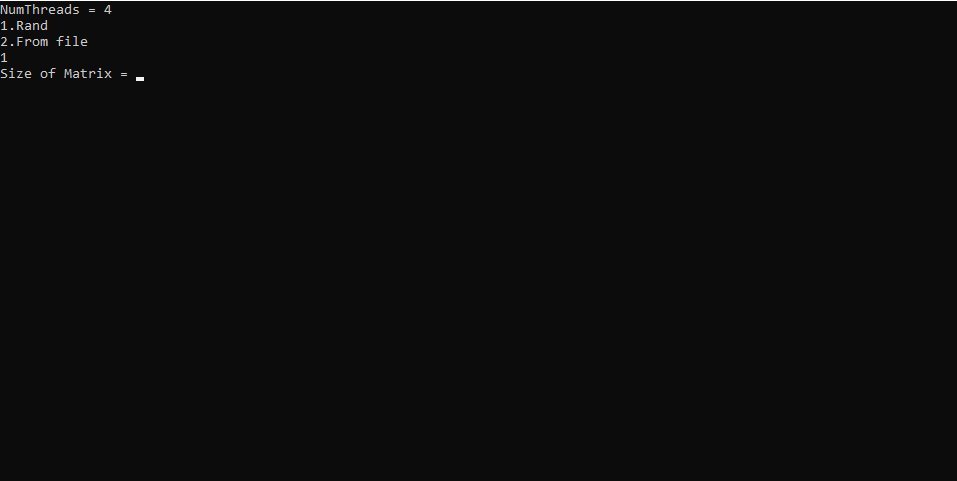
Для TBB версии алгоритма дополнительно реализуется класс-функтор, принимающий на вход две матрицы и оперирующий рабочими векторами, который будет реализовывать основную часть умножения в методе operator().

**Описание программной реализации**

**Руководство пользователя**

При запуске программы появляется консоль, в окне которой пользователь последовательно:

1. задает «NumThreads» – число потоков, на которых будет работать алгоритм;
2. выбирает способ задания матрицы: «1.Rand» - программа сгенерирует матрицу или «2.From file» - матрица будет считана из файла, в который данные заносятся вручную;
3. если выбран вариант «1.Rand», то пользователь должен задать «Size of Matrix» - размерность матрицы.



По окончанию работы программы на экран будет выведено время работы алгоритма умножения с использованием технологии распараллеливания:



**Руководство программиста**

В программе созданы и описаны 2 класса:

*class Matrix*

поля:

vector<type> vv; // содержит все элементы(values) матрицы

методы:

type\* operator [] (int j)

void transposition () //транспонирование матрицы

static void randMatrixdata(Matrix& A, Matrix& B, int N)//генерирует матрицы

…

Класс описывает двумерную матрицу, которая хранится «одномерно» в vector<type> vv. Элементы в матрицу добавляются «по столбцам», например матрица A = будет храниться так: .

Доступ к ячейкам осуществляется следующим образом:

A[номер столбца][номер строки] = (&vv[номер столбца \*кол-во столбцов])[номер строки])

Умножение матриц выполняется тройным обычным циклом по , , и выполняется распараллеливание по первому циклу, для более быстрой работы.

*class CCSMatrix*

поля:

vector<type> values;

vector<int> rows;

vector<int> pointer;

int N; //размерность матрицы

методы:

Matrix CCStoMatrix()// приведение CCS матрицы к виду обычной матрицы

void transposition() // транспонирование матрицы

void unite(const MatrixCCS &m) // объединяет части в результирующую матрицу

MatrixCCS parallelMult(const MatrixCCS& m, const int numThreads)

…

Класс описывает разряженную матрицу, хранящуюся в столбцовом формате CCS. В алгоритме умножения в процессе подсчитывается оставшееся количество ненулевых элементов в текущем столбце и строке. Если в столбце или строке все оставшиеся элементы нулевые, выполняется continue. Данный вариант хорошо себя показывает на сильно разряженных матрицах, но увеличивает объем кода.

**Результаты экспериментов**

Персональный компьютер:

CPU: AMD Ryzen 7 2700X 4.3GHz;

8 ядер, 16 потоков;

RAM: 16 Gb;

OS: Windows 10 x64.

Указано время работы только параллельной части алгоритма умножения матриц. Время, затраченное на транспонирование первой матрицы, не учитывалось из-за линейной сложности, то есть при достаточно большой размерности матрицы его вклад будет незначителен.

Время указано в секундах, 15% ненулевых элементов.

## Результаты замеров:

Размерность матрицы = 700

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 поток | 2 потока | 4 потока | 8 потоков |
| OpenMP | 5,78 | 3,21 | 1,77 | 1,27 |
| TBB | 6,02 | 3,3 | 1,86 | 1,16 |

Размерность матрицы = 1500

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 поток | 2 потока | 4 потока | 8 потоков | 16 потоков |
| OpenMP | 54,53 | 29,35 | 15,75 | 9,57 | 7,16 |
| TBB | 56,15 | 30,19 | 15,92 | 9,87 | 7,36 |

Размерность матрицы = 2000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 поток | 2 потока | 4 потока | 8 потоков | 16 потоков | 32 потока |
| OpenMP | 128,6 | 68,96 | 36,18 | 21,34 | 15,93 | 16,11 |
| TBB | 131,7 | 70,36 | 36, 93 | 21,54 | 16,21 | 16,59 |

## Ускорение

Размерность матрицы = 1500

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 поток | 2 потока | 4 потока | 8 потоков | 16 потоков |
| OpenMP | 1 | 1,85 | 3,46 | 5,69 | 7,61 |
| TBB | 1 | 1,86 | 3,52 | 5,68 | 7,62 |

Размерность матрицы = 2000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 поток | 2 потока | 4 потока | 8 потоков | 16 потоков | 32 потока |
| OpenMP | 1 | 1,86 | 3,55 | 5,97 | 8,07 | 7,98 |
| TBB | 1 | 1,87 | 3,56 | 6,11 | 8,12 | 7,93 |

На основе результатов проведенных экспериментов можно заключить, что обе технологии распараллеливания OpenMP и TBB показывают приблизительно одинаковые (с небольшими отличиями в обе стороны) результаты, и при этом работают эффективно, показывая хорошее ускорение, величина которого зависит от количества потоков и размерности матрицы. Лучший результат был зафиксирован при запуске на 16 потоках.

Также можно сделать вывод о том, что чем больше размерность матрицы, тем больше ускорение. При работе с матрицей размера 2000x2000 OpenMP показывает результат немного лучше, чем TBB. Однако, это различие может быть значимо только при работе с намного бо́льшими объемами данных.

**Заключение**

В ходе выполнения лабораторной работы была написана программа, реализующая параллельный алгоритм умножения разреженных матриц со столбцовым форматом хранения с использованием двух технологий распараллеливания: OpenMP и TBB. Также были проведены эксперименты, показывающие эффективность параллельной схемы.

Рассмотренные в работе технологии распараллеливания на небольших размерностях матриц показывают одинаково хорошее ускорение, зависящее от количества потоков. На больших же матрицах технология OpenMP показывает бо́льшую эффективность.

Различие технологий распараллеливания заключается также в том, что при использовании TBB помимо распределения задач по потокам необходимо описать целый класс-функтор, что усложняет процесс реализации алгоритма.

**Литература**

1. Мееров И.Б., Сысоев А.В. Образовательный комплекс «Параллельные численные методы» Лабораторная работа №7 Разреженное матричное умножение. -Нижний Новгород. -2011. - 82c.
2. Академия Intel: Intel Parallel Programming Professional (Introduction). [Электронный ресурс]. Лекция 7: Умножение разреженных матриц. - URL: https://www.intuit.ru/studies/courses/4447/983/lecture/14931?page=5 (Дата обращения: 31.05.2020).

**Приложение**

MatrixCCS parallelMult(const MatrixCCS& m, const int numThreads)//параллельное умножение

{

int numTask = numThreads;

if (numTask > N)//если потоков больше, чем размерность матрицы

numTask = N;

//this->transposition();

struct Task//задача для каждого потока

{

int pointerStart;

int pointerEnd;

int taskIndex;

Task() :pointerStart(0), pointerEnd(0), taskIndex(0) {}

Task(int a, int b, int c) :pointerStart(a), pointerEnd(b), taskIndex(c) {}

};

vector<MatrixCCS> tmpMatrix(numTask, MatrixCCS(N));

vector<int> elCountM(numTask);

vector<int>\* cols = &rows;

vector<Task> task(numTask);//получаем вектор, в котором numTask задач

//создание задач потокам

{

int sizeTask = m.N / numTask + (bool)(m.N % numTask);//если кратное, то + 0, иначе + 1 (распределяем столбцы потокам)

for (int i = 0; i < numTask; i++)

task[i] = Task(i \* sizeTask, std::min((i + 1) \* sizeTask, m.N), i % numTask);//(начало, конец, индекс)

int lastPointerM = 0;

for (int i = 0; i < numTask; i++)

{

elCountM[i] = lastPointerM;

int jstart = task[i].pointerStart;

const int jend = task[i].pointerEnd;

for (jstart; jstart < jend; jstart++)

{

lastPointerM += m.pointer[jstart + 1] - m.pointer[jstart];//считаем количество ненулевых элементов

}

}

}

#pragma omp parallel for

for (int itask = 0; itask < task.size(); itask++)//каждый itask поток

for (int j = task[itask].pointerStart; j < task[itask].pointerEnd; j++)//проходим по ненулевым элементам столбцов

{

int indexTask = task[itask].taskIndex;//номер потока(номер задачи)

int numElInResCol = 0;//количество элементов в результирующем столбце

const int numElementInCol = m.pointer[j + 1] - m.pointer[j];//количество элементов в столбце

if (numElementInCol == 0)//если столбец только с нулями

{

int size = tmpMatrix[indexTask].pointer.size();

tmpMatrix[indexTask].pointer.push\_back(tmpMatrix[indexTask].pointer[size - 1]);

continue;

}

int elCountThis = 0;

for (int i = 0; i < N; i++)

{

const int numElementInRows = pointer[i + 1] - pointer[i];

if (numElementInRows == 0)

{

continue;

}

int tmpNumElRow = numElementInRows;//для A

int tmpNumElCol = numElementInCol;//для B

double sum = 0;

int tmpElCountM = elCountM[indexTask];

for (int z = 0; z < std::min(tmpNumElCol, tmpNumElRow);)//проходим по столбцу и умножаем

{

int colThis = (\*cols)[elCountThis];//cols==rows(из A)

int rowM = m.rows[tmpElCountM];//индекс rows с которого нужно начинать task(получаем значение rows в B)

if (colThis == rowM)

{

sum += values[elCountThis] \* m.values[tmpElCountM];

tmpNumElCol--;

tmpNumElRow--;

tmpElCountM++;

elCountThis++;

}

else if (colThis < rowM)

{

tmpNumElRow--;//уменьшаем количество элементов

elCountThis++;//передвигаем индекс

}

else

{

tmpNumElCol--;

tmpElCountM++;

}

}

for (int z = 0; z < tmpNumElRow; z++)//двигаем индекс в rows, если у одной матрицы в столбце меньше чисел

elCountThis++;

if (sum != 0)//записываем в результирующую матрицу(по столбцам)

{

tmpMatrix[indexTask].values.push\_back(sum);

tmpMatrix[indexTask].rows.push\_back(i);

numElInResCol++;

}

}

const int size = tmpMatrix[indexTask].pointer.size();

tmpMatrix[indexTask].pointer.push\_back(tmpMatrix[indexTask].pointer[size - 1] + numElInResCol);//записываем предыдущее + количество элементов, которые в столбце

elCountM[indexTask] += numElementInCol;

}

for (int i = 1; i < tmpMatrix.size(); i++)

{

tmpMatrix[0].uniteMatrix(tmpMatrix[i]);//посылаем следующую часть матрицы, объединяем все части с нулевой

}

if (tmpMatrix[0].pointer.size() < N + 1)

tmpMatrix[0].pointer.push\_back(tmpMatrix[0].values.size());

return tmpMatrix[0];

}