Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт Информационных Технологий Математики и Механики

Отчет по лабораторной работе

**Использование методов параллельного программирования в умножении разреженных матриц**

Выполнил:

студент группы 381506-2

Цирулев А.С.

Проверил:

Ассистент кафедры МОСТ

Козинов Е.А.

Нижний Новгород

2018 г

Оглавление

[Постановка задачи 3](#_Toc514705444)

[Метод решения 4](#_Toc514705445)

[Схема распараллеливания 6](#_Toc514705446)

[Описание программной реализации 7](#_Toc514705447)

[Подтверждение корректности 13](#_Toc514705448)

[Оценка ускорения при распараллеливании алгоритма 21](#_Toc514705449)

# Постановка задачи

Алгебра разреженных матриц – важный раздел математики, имеющий очевидное практическое применение. Разреженные матрицы возникают естественным образом при постановке и решении задач из различных научных и инженерных областей. В частности, при формулировании оптимизационных задач большой размерности с линейными ограничениями матрица системы ограничений нередко оказывается разреженной. Матрицы с существенно преобладающим количеством нулевых элементов формируются при численном решении дифференциальных уравнений в частных производных. Такие матрицы возникают в теории графов.

Попробуем определиться с понятием *разреженная матрица*. Выясняется, что это не так просто. Один из первых источников по данной тематике, в комплексе рассматривающий проблемы работы с разреженными матрицами, дает такое определение: *разреженной называют матрицу, имеющую малый процент ненулевых элементов*. В ряде других источников встречается такая формулировка: матрица размера N x N называется разреженной, если количество ее ненулевых элементов есть O(N).

Эффективные методы хранения и обработки разреженных матриц на протяжении последних десятилетий вызывают интерес у широкого круга исследователей. Выясняется, что для учета разреженной структуры приходится существенно усложнять как методы хранения, так и алгоритмы обработки. Многие тривиальные с точки зрения программирования алгоритмы в разреженном случае становятся весьма сложными.

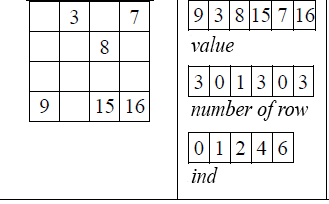
Целью данной работы является построение эффективного алгоритма умножения разреженных матриц и последующее ускорение его работы при помощи средств параллельного программирования.

# Метод решения

В рамках задачи будет использована CCS-структура (Compressed Column Storage) хранения матриц. Структура состоит из трёх массивов:

* Массив значений (**value**)
* Массив номеров строк для каждого элемента из **value** (**numberOfRow**)
* Массив индексов начала каждой строки в столбцах value и **numberOfRow** (**ind**)

Отметим, что количество элементов массива **ind** равно N + 1. При этом элементы строки *i* в массиве **value** находятся по индексам от **ind[i]** до **ind[i+1]-1** включительно (*i*-й элемент массива **ind** указывает на начало *i*-й строки). Таким образом обрабатывается случай пустых строк. Последний элемент массива **ind** содержит число ненулевых элементов матрицы (NZ). Соответственно, размер массивов value и number of row равны NZ.



***Рис.1 представление формата CCS***

Поскольку при перемножении матриц А и В элементы i-й строки А перемножаются с элементами из i-го столбца В, то в рамках задачи возникает вопрос о предварительной подготовке матрицы А для облегчения работы. К примеру – можно транспонировать матрицу А в формат CRS (Compressed Row Storage), отличие которого от CCS состоит в хранении элементов по строкам. Тогда перемножение матриц АТ и В (АТ – матрица А в формате CRS) не составит особого труда (поскольку все операции будут проходить над столбцами АТ и В) и приведёт к корректному результату. Алгоритм транспонирования описан в книге Писсанецки – Технология разреженных матриц.

Сам алгоритм умножения будет выглядеть следующим образом:

1. Создание векторов **values**, **row** и **row\_index**. Размер вектора **row\_index = N+1**, где N – размер матрицы;
2. Транспонирование матрицы А (получим матрицу АТ);
3. В цикле по *i* от 0 до N-1 перебор всех строк матрицы AТ. Для каждого *i* в цикле по *j* от 0 до N-1 перебор всех строк матрицы B. Вычисление скалярного произведения векторов-строк AТ*i* и B*j*, пусть оно равно V.

Сам алгоритм скалярного произведения можно взять следующим:

для каждого элемента строки матрицы AТ перебирает все элементы строки матрицы B, пока не будет найден элемент с таким же значением в массиве **numberOfRow** или не кончится строка.

Если V отлично от нуля, добавить в вектор **value** элемент V, в вектор number of row – элемент *j*. При окончании цикла по *j* ­­­– корректировка значения **ind[i+1]** путём записи туда текущего значения числа ненулевых элементов V.

1. Инициализация CCS-матрицы С с поледующим заполнением её полей на основе векторов **values**, **row** и **row\_index**;

# Схема распараллеливания

В качестве инструментария для создания параллельной версии программы будем использовать OpenMP и Intel TBB. Начнём с OpenMP.

В данной задаче для ускорения вычислений можно применить распараллеливание цикла, в котором осуществляются вычисления значений. Трудность в данном случае состоит в записи вычисленных значений в массивы **value** и **numberOfRow**. Если в последовательной версии для записи значений можно было использовать **push\_back()** (вектора из библиотеки STL), то в случае с многопоточным приложением потоки могут помешать друг другу корректно выполнить **push\_back()**. В качестве решения проблемы можно объявить рабочие массивы в каждом из потоков. После вычислений в одном потоке организуем объединение массивов. Массив **ind** будет заполняться после выполнения параллельной части, используя уже вычисленные значения **value** и **numberOfRow**.

Также изменениям подвергнется и вычислительная часть. Поскольку OpenMP не предполагает использование **break** в распараллеленной версии цикла **for**, то приходится искать иной способ реализации скалярного произведения. Можно поступить следующим образом:

1. Создадим дополнительный целочисленный массив **temp** длины N. Инициализируем его числами -1. Обнулим вещественную переменную S.
2. Просмотрим в цикле все ненулевые элементы вектора матрицы АТ. Пусть такой элемент с порядковым номером *i* расположен в столбце с номером **j = AT.numberOfRow[i]**. В этом случае запишем *i* в *j*-ю ячейку дополнительного массива.
3. Просмотрим в цикле все ненулевые элементы матрицы B. Пусть элемент с порядковым номером *k* расположен в столбце с номером **z = B. numberOfRow [k]**. Проверим значение **temp[z]**. Если оно равно -1, в первом векторе нет соответствующего элемента, т.е. умножение вы-полнять не нужно. Иначе умножаем **B.value[k]** и **AT.value[temp[z]]** и накапливаем в S.

Массив **temp** будет объявлен в параллельной секции, но до объявления цикла **for**.

Та же идея алгоритма умножения будет применена и для Intel TBB. Основное отличие будет связано с другим принципом распараллеливания цикла for, для которого необходимо написать класс-функтор, содержащий в себе то, что было написано в параллельной секции для OpenMP.

# Описание программной реализации

Структура CCS-матрицы:

struct CCSmatrix

{

int size;//N

int countOfElems;//количество ненулевых элементов

double \*value;//ненулевые значения

int \*numberOfRow;//номер строки для соотв. элемента

int \*ind;

};

Инициализация матрицы:

void init\_CCSmatrix(int sz, int cnt, CCSmatrix &matr)

{

matr.size=sz;

matr.countOfElems=cnt;

matr.value=new double[cnt];

matr.numberOfRow=new int[cnt];

matr.ind=new int[sz+1];

memset(matr.ind,0,sizeof(int)\*(sz+1));

memset(matr.value,0,sizeof(double)\*cnt);

memset(matr.numberOfRow,0,sizeof(int)\*cnt);

}

Транспонирование матрицы:

CCSmatrix transposeMatrix(CCSmatrix &matr)

{

CCSmatrix res;

init\_CCSmatrix(matr.size,matr.countOfElems,res);

for (int i=0;i<matr.countOfElems;i++)

res.ind[matr.numberOfRow[i]+1]++;

int S=0;

int tmp;

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

{

tmp=res.ind[i];

res.ind[i]=S;

S=S+tmp;

}

for (int i=0; i<matr.size;i++)

{

int j1=matr.ind[i];

int j2=matr.ind[i+1];

int row=i;

for(int j=j1;j<j2;j++)

{

double val=matr.value[j];

int ind1=matr.numberOfRow[j];

int ind2=res.ind[min(ind1+1,matr.size-1)];

res.value[ind2]=val;

res.numberOfRow[ind2]=row;

res.ind[ind1+1]++;

}

}

return res;

}

Последовательная версия умножения (возвращает количество ненулевых элементов):

int multiply(CCSmatrix A, CCSmatrix B, CCSmatrix &res)

{

int total\_cnt=0;

if (A.size!=B.size)

throw "size error";

CCSmatrix AT;

AT=transposeMatrix(A);

vector<int>cols;

vector<double> vals;

vector<int>inds;

inds.push\_back(0);

int tmpInd;

for (int i=0;i<AT.size;i++)

{

tmpInd=0;

for (int j=0;j<AT.size;j++)

{

double sum=0;

//---------------

for (int k=AT.ind[i];k<AT.ind[i+1];k++)

{

for (int l=B.ind[j];l<B.ind[j+1];l++)

if(AT.numberOfRow[k]==B.numberOfRow[l])

{

sum+=AT.value[k]\*B.value[l];

break;

}

}

//--------------

if (abs(sum)>0)

{

total\_cnt++;

cols.push\_back(j);

vals.push\_back(sum);

tmpInd++;

}

}

inds.push\_back(tmpInd+inds[i]);

}

init\_CCSmatrix(AT.size,cols.size(),res);

for (int i=0;i<cols.size();i++)

{

res.numberOfRow[i]=cols[i];

res.value[i]=vals[i];

}

for (int i=0;i<inds.size();i++)

res.ind[i]=inds[i];

res=transposeMatrix(res);

return total\_cnt;

}

OpenMP-версия:

int multiply(CCSmatrix A, CCSmatrix B, CCSmatrix &res)

{

int total\_cnt=0;//число элементов

if (A.size!=B.size)

throw "size error";

CCSmatrix AT;

AT=transposeMatrix(A);

B=transposeMatrix(B);

B=transposeMatrix(B);

int N=A.size;

int chunk=20;

int i, j, k;

vector<int> \*rows = new vector<int>[N];

vector<double> \*values = new vector<double>[N];

int\* row\_index = new int[N + 1];

memset(row\_index, 0, sizeof(int) \* N);

#pragma omp parallel num\_threads(thrds)

{

int \*temp = new int[N];

#pragma omp for private(j,k) schedule (dynamic)

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

{

memset(temp, -1, N \* sizeof(int));

int ind1 = AT.ind[i], ind2 = AT.ind[i + 1];

for (j = ind1; j < ind2; j++)

{

int col = AT.numberOfRow[j];

temp[col] = j;

}

for (j = 0; j < N; j++)

{

double sum = 0;

int ind3 = B.ind[j], ind4 = B.ind[j + 1];

for (k = ind3; k < ind4; k++)

{

int bcol = B.numberOfRow[k];

int aind = temp[bcol];

if (aind != -1)

sum += AT.value[aind] \* B.value[k];

}

if (fabs(sum) > 0)

{

rows[i].push\_back(j);

values[i].push\_back(sum);

row\_index[i]++;

}

}

}

delete [] temp;

}

int NZ = 0;

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

{

int tmp = row\_index[i];

row\_index[i] = NZ;

NZ += tmp;

}

row\_index[N] = NZ;

init\_CCSmatrix(N,NZ,res);

int count = 0;

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

{

int size = rows[i].size();

if(size!=0)

{

memcpy(&res.numberOfRow[count], &rows[i][0], size \* sizeof(int));

memcpy(&res.value[count], &values[i][0], size \* sizeof(double));

count += size;

}

}

memcpy(res.ind, &row\_index[0], (N + 1) \* sizeof(int));

for (int t=0;t<res.countOfElems;t++)

for (int t=0;t<res.size+1;t++)

delete [] row\_index;

delete [] rows;

delete [] values;

res=transposeMatrix(res);

return NZ;

}

ТВВ-версия:

Поскольку использование ТВВ подразумевает написание класса-функтора (имеющий в своём распоряжении только метод **operator()**), то стоит отдельно расписать необходимые нововведения в код:

* Инициализация библиотеки с использованием объекта класса **task\_scheduler\_init**;
* Класс-функтор, реализующий основную часть умножения в методе **operator()**.
* Одномерное итерационное пространство **blocked\_range** (воспользуемся встроенными возможностями ТВВ)

Распараллеливание будет реализовано шаблонной функцией ТВВ

**parallel\_for()**.

Класс-функтор:

class Multiplicator

{

CCSmatrix A, B;

vector<int>\* rows;

vector<double>\* values;

int \*row\_index;

public:

Multiplicator(CCSmatrix& \_A, CCSmatrix& \_B, vector<int>\* &\_rows, vector<double>\* &\_values, int \*\_row\_index) : A(\_A), B(\_B), rows(\_rows), values(\_values), row\_index(\_row\_index)

{}

void operator()(const blocked\_range<int>& r) const

{

int begin = r.begin();

int end = r.end();

int N = A.size;

int i, j, k;

int \*temp = new int[N];

for (i = begin; i < end; i++)

{

memset(temp, -1, N \* sizeof(int));

int ind1 = A.ind[i], ind2 = A.ind[i + 1];

for (j = ind1; j < ind2; j++)

{

int col = A.numberOfRow[j];

temp[col] = j;

}

for (j = 0; j < N; j++)

{

double sum = 0;

int ind3 = B.ind[j], ind4 = B.ind[j + 1];

for (k = ind3; k < ind4; k++)

{

int bcol = B.numberOfRow[k];

int aind = temp[bcol];

if (aind != -1)

sum += A.value[aind] \* B.value[k];

}

if (fabs(sum) > 0)

{

rows[i].push\_back(j);

values[i].push\_back(sum);

row\_index[i]++;

}

}

}

delete [] temp;

}

};

Функция умножения:

int multiply(CCSmatrix A, CCSmatrix B, CCSmatrix &res,int thrds)

{

int total\_cnt=0;

if (A.size!=B.size)

throw "size error";

CCSmatrix AT,BT;

AT=transposeMatrix(A);

BT=transposeMatrix(B);

B=transposeMatrix(BT);

int N=A.size;

int i, j, k;

vector<int> \*rows = new vector<int>[N];

vector<double> \*values = new vector<double>[N];

int\* row\_index = new int[N + 1];

memset(row\_index, 0, sizeof(int) \* N);

task\_scheduler\_init init(thrds);

int grainsize = 2;

parallel\_for(blocked\_range<int>(0, A.size, grainsize), Multiplicator(AT, B, rows, values, row\_index));

int NZ = 0;

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

{

int tmp = row\_index[i];

row\_index[i] = NZ;

NZ += tmp;

}

row\_index[N] = NZ;

init\_CCSmatrix(N, NZ, res);

int count = 0;

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

{

int size = rows[i].size();

if(size!=0)

{

memcpy(&res.numberOfRow[count], &rows[i][0], size \* sizeof(int));

memcpy(&res.value[count], &values[i][0], size \* sizeof(double));

count += size;

}

}

memcpy(res.ind, &row\_index[0], (N + 1) \* sizeof(int));

delete [] row\_index;

delete [] rows;

delete [] values;

res=transposeMatrix(res);

return NZ;

}

# Подтверждение корректности

Поскольку программа подразумевает чтение матриц с бинарного файла, то становится очевидной необходимость создания генератора разреженных матриц.

Было решено выбрать следующий алгоритм для генерации матриц:

1. При запуске указываются размер матрицы (**size**) и количество ненулевых элементов (**not\_null\_elements\_in\_one\_col**) в каждом столбце;
2. Каждый столбец разбивается на **not\_null\_elements\_in\_one\_col** фрагментов, в каждом из которых случайным образом генерируются место и значение элемента матрицы;
3. Сгенерированные данные записываются в бинарный файл;

Код программы-генератора:

#include <iostream>

#include <cstdlib>

#include <random>

#include <algorithm>

#include <chrono>

using namespace std;

struct CCSmatrix

{

int size;

int countOfElems;

double \*value;

int \*numberOfRow;

int \*ind;

};

void init\_CCSmatrix(int sz, int cnt, CCSmatrix &matr)

{

matr.size=sz;

matr.countOfElems=cnt;

matr.value=new double[cnt];

matr.numberOfRow=new int[cnt];

matr.ind=new int[sz+1];

for (int i=0;i<sz+1;i++)

matr.ind[i]=0;

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

{

matr.value[i]=0;

matr.numberOfRow[i]=-1;

}

}

int main(int argc, char \* argv[])

{

char \*file\_name;

int m\_size,not\_null\_elements\_in\_one\_col,k,Index;

if (argc > 1)

{

m\_size =atoi(argv[1]);

not\_null\_elements\_in\_one\_col=atoi(argv[2]);

if(m\_size<=0)

{

cout<<"wrong size\n";

return -1;

}

if(not\_null\_elements\_in\_one\_col<=0)

{

cout<<"wrong count of elements\n";

return -1;

}

if(m\_size<not\_null\_elements\_in\_one\_col)

{

cout<<"wrong count of elements\n";

return -1;

}

file\_name=argv[3];

}

default\_random\_engine generator(chrono::system\_clock::now().time\_since\_epoch().count());

uniform\_real\_distribution <double> distribution(-1e4, 1e4);

int minNum,maxNum;

Index=0;

if (argc > 1)

{

cout<<"test size "<<m\_size<<endl;

cout<<"test count "<<not\_null\_elements\_in\_one\_col<<endl;

cout<<"test name "<<file\_name<<endl;

}

else

{

cout<<"Enter size of matrix: ";

cin>>m\_size;

cout<<"Enter count of not null elements in one col: ";

cin>>not\_null\_elements\_in\_one\_col;

cout<<"Enter file name: ";

cin>>file\_name;

}

FILE \*fl=fopen(file\_name,"wb");

fwrite(&m\_size,sizeof(m\_size),1,fl);//запись размерности матрицы в файл

fwrite(&not\_null\_elements\_in\_one\_col,sizeof(not\_null\_elements\_in\_one\_col),1,fl);

k=m\_size/not\_null\_elements\_in\_one\_col;

if(m\_size%not\_null\_elements\_in\_one\_col!=0)

k++;

CCSmatrix A,B;

init\_CCSmatrix(m\_size,not\_null\_elements\_in\_one\_col\*m\_size,A);

init\_CCSmatrix(m\_size,not\_null\_elements\_in\_one\_col\*m\_size,B);

//A

//values

for (int u=0;u<m\_size\*not\_null\_elements\_in\_one\_col;u++)

{

A.value[u]=distribution(generator);

}

//rows

for (int j=0;j<m\_size;j++)//по каждой колонке

{

for(int i=0;i<not\_null\_elements\_in\_one\_col;i++)

{

minNum=k\*i;

maxNum=min(k\*(i+1),m\_size);

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<unsigned long long> dist(minNum,maxNum+1);

A.numberOfRow[Index]=dist(gen);

Index++;

}

}

//ind

A.ind[0]=0;

for (int y=1;y<A.size+1;y++)

{

A.ind[y]=A.ind[y-1]+not\_null\_elements\_in\_one\_col;

}

fwrite(A.value,sizeof(\*A.value),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fwrite(A.numberOfRow,sizeof(\*A.numberOfRow),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fwrite(A.ind,sizeof(\*A.ind),m\_size+1,fl);

//==================================================================

//B

Index=0;

//values

for (int u=0;u<m\_size\*not\_null\_elements\_in\_one\_col;u++)

{

B.value[u]=distribution(generator);

}

//rows

for (int j=0;j<m\_size;j++)//по каждой колонке

{

for(int i=0;i<not\_null\_elements\_in\_one\_col;i++)

{

minNum=k\*i;

maxNum=min(k\*(i+1),m\_size);

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<unsigned long long> dist(minNum,maxNum+1);

B.numberOfRow[Index]=dist(gen);

Index++;

}

}

//ind

B.ind[0]=0;

for (int y=1;y<B.size+1;y++)

{

B.ind[y]=B.ind[y-1]+not\_null\_elements\_in\_one\_col;

}

fwrite(B.value,sizeof(\*B.value),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fwrite(B.numberOfRow,sizeof(\*B.numberOfRow),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fwrite(B.ind,sizeof(\*B.ind),m\_size+1,fl);

fclose(fl);

cout<<"======================================================\n";

return 0;

}

Для проверки полученных ответов будем проводить сравнение с результатами выполнения умножения матриц с использованием средств Intel MKL, а также напишем специальную программу (checker) для сравнения полученных ответов. Для хранения и чтения матриц воспользуемся функциями чтения и записи в бинарный файл.

Вычисление с использованием Intel MKL:

#include "mkl.h"

#include "mkl\_spblas.h"

#include <iostream>

#include <cstdio>

#include <algorithm>

#include "mkl\_blas.h"

using namespace std;

struct CCSmatrix

{

int size;

int countOfElems;

double \*value;

int \*numberOfRow;

int \*ind;

};

void init\_CCSmatrix(int sz, int cnt, CCSmatrix &matr)

{

if(sz\*sz<cnt)

throw "wrong count of elements";

if(sz<=0)

throw "wrong size";

matr.size=sz;

matr.countOfElems=cnt;

matr.value=new double[cnt];

matr.numberOfRow=new int[cnt];

matr.ind=new int[sz+1];

for (int i=0;i<sz+1;i++)

matr.ind[i]=0;

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

{

matr.value[i]=0;

matr.numberOfRow[i]=-1;

}

}

CCSmatrix transposeMatrix(CCSmatrix &matr)

{

CCSmatrix res;

init\_CCSmatrix(matr.size,matr.countOfElems,res);

for (int i=0;i<matr.countOfElems;i++)

res.ind[matr.numberOfRow[i]+1]++;

int S=0;

int tmp;

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

{

tmp=res.ind[i];

res.ind[i]=S;

S=S+tmp;

}

for (int i=0; i<matr.size;i++)

{

int j1=matr.ind[i];

int j2=matr.ind[i+1];

int row=i;

for(int j=j1;j<j2;j++)

{

double val=matr.value[j];

int ind1=matr.numberOfRow[j];

int ind2=res.ind[min(ind1+1,matr.size-1)];

res.value[ind2]=val;

res.numberOfRow[ind2]=row;

res.ind[ind1+1]++;

}

}

return res;

}

void to\_MKL(CCSmatrix &matr)

{

for (int i=0;i<matr.countOfElems;i++)

matr.numberOfRow[i]++;

for (int j=0;j<=matr.size;j++)

matr.ind[j]++;

}

void from\_MKL(CCSmatrix &matr)

{

for (int i=0;i<matr.countOfElems;i++)

matr.numberOfRow[i]--;

for (int j=0;j<=matr.size;j++)

matr.ind[j]--;

}

int main(int argc, char\* argv[] )

{

double \*C;

int m\_size,not\_null\_elements\_in\_one\_col,inf,nzmax;

const double EPSILON = 0.000001;

char \*file\_name;

char \*output\_name;

char trans='t';

int request=2;

int is\_sort=3;

nzmax=-1;

if(argc>1)

{

file\_name=argv[1];

output\_name=argv[2];

}

else

{

file\_name="nn.dat";

}

FILE \*fl=fopen(file\_name,"rb");

CCSmatrix A\_test,B\_test,res;

fread(&m\_size,sizeof(m\_size),1,fl);

cout<<"we have "<<m\_size<<" size"<<endl;

C=new double[m\_size\*m\_size];

int \*rC;

int \*iC;

iC=new int[m\_size+1];

fread(&not\_null\_elements\_in\_one\_col,sizeof(not\_null\_elements\_in\_one\_col),1,fl);

cout<<"we have "<<not\_null\_elements\_in\_one\_col<<" count in one column"<<endl;

init\_CCSmatrix(m\_size,not\_null\_elements\_in\_one\_col\*m\_size,A\_test);

init\_CCSmatrix(m\_size,not\_null\_elements\_in\_one\_col\*m\_size,B\_test);

init\_CCSmatrix(m\_size,m\_size\*m\_size,res);

fread(A\_test.value,sizeof(\*A\_test.value),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fread(A\_test.numberOfRow,sizeof(\*A\_test.numberOfRow),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fread(A\_test.ind,sizeof(\*A\_test.ind),m\_size+1,fl);

fread(B\_test.value,sizeof(\*B\_test.value),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fread(B\_test.numberOfRow,sizeof(\*B\_test.numberOfRow),m\_size\*not\_null\_elements\_in\_one\_col,fl);

fread(B\_test.ind,sizeof(\*B\_test.ind),m\_size+1,fl);

fclose(fl);

B\_test=transposeMatrix(B\_test);

to\_MKL(A\_test);

to\_MKL(B\_test);

//=============================================================

//=============================================================

mkl\_dcsrmultcsr(

&trans,&request,&is\_sort,

&A\_test.size,&A\_test.size,&B\_test.size,

A\_test.value,A\_test.numberOfRow,A\_test.ind,

B\_test.value,B\_test.numberOfRow,B\_test.ind,

res.value,res.numberOfRow,res.ind,&nzmax,&inf

);

from\_MKL(res);

res=transposeMatrix(res);

cout<<"------------------------------\n";

cout<<"res\_test \n";

fl=fopen(output\_name,"wb");

for (int i=0;i<res.ind[m\_size-1];i++)

{

//cout<<res.value[i]<<" | ";

fwrite(&res.value[i],sizeof(res.value[i]),1,fl);

}

cout<<"rows\n\n";

for (int i=0;i<res.ind[m\_size-1];i++)

{

//cout<<res.numberOfRow[i]<<" | ";

fwrite(&res.numberOfRow[i],sizeof(res.numberOfRow[i]),1,fl);

}

cout<<"inds\n";

cout<<"\n";

for (int i=0;i<m\_size+1;i++)

{

//cout<<res.ind[i]<<" | ";

fwrite(&res.ind[i],sizeof(res.ind[i]),1,fl);

}

cout<<"calculateed";

cout<<endl;

cout<<endl;

fclose(fl);

return 0;

}

Стоит отметить, что в MKL счёт начинается с единицы, а не с нуля, как в С++.

Код программы checker:

#include <cstdio>

#include <cmath>

#include <string>

using namespace std;

enum verdict { NO = 1, AC, WA, CE, ML, TL, RE, IL, PE, DE };

class result

{

private:

FILE \* bur;

public:

enum ext\_cls { NO = 1, VERDICT, MESSAGE, TIME, MEMORY };

result (bool read = false)

{

if (read)

bur = fopen("result.txt", "r");

else

bur = fopen("result.txt", "w");

}

~result()

{

fclose (bur);

}

void write\_type(ext\_cls t) //записать что-то

{

fwrite(&t, sizeof (t), 1, bur);

}

void write\_verdict(verdict v) //записать вердикт

{

write\_type(ext\_cls::VERDICT);

fwrite(&v, sizeof (v), 1, bur);

}

void write\_message(string str)

{

write\_type(ext\_cls::MESSAGE);

int l = str.size ();

fwrite(&l, sizeof (l), 1, bur);

fwrite (&str[0], sizeof (str[0]), l, bur);

}

}checker\_result;

/////////////////////////////////////////////////////////////////////////////////////////

int main (int argc, char \* argv[])

{

char \*user\_answer;

char \*mkl\_answer;

if(argc>1)

{

user\_answer=argv[1];

mkl\_answer=argv[2];

}

FILE \* user\_file = fopen(user\_answer, "rb");//ответ алгоритма

FILE \* mkl\_file = fopen(mkl\_answer, "rb");//ответ mkl

int noNul\_cnt,m\_size;

// Выделяем память для матрицы ответа жюри и ответа участника

fread(&noNul\_cnt,sizeof(noNul\_cnt),1,user\_file);

fread(&m\_size,sizeof(m\_size),1,user\_file);

double \*user\_vals=new double[noNul\_cnt];

int \*user\_rows=new int[noNul\_cnt];

int \*user\_index=new int[m\_size+1];

double \*mkl\_vals=new double[noNul\_cnt];

int \*mkl\_rows=new int[noNul\_cnt];

int \*mkl\_index=new int[m\_size+1];

fread(user\_vals,sizeof(user\_vals),1,user\_file);

fread(user\_rows,sizeof(user\_rows),1,user\_file);

fread(user\_index,sizeof(user\_index),1,user\_file);

fread(mkl\_vals,sizeof(mkl\_vals),1,mkl\_file);

fread(mkl\_rows,sizeof(mkl\_rows),1,mkl\_file);

fread(mkl\_index,sizeof(mkl\_index),1,mkl\_file);

// вычисление расхождения и установление несоответствия в индексах

double diff\_val = 0.0;

bool rows\_eqal=true;

bool inds\_eqal=true;

for (int i = 0; i <noNul\_cnt; i++)

diff\_val += (user\_vals[i] - mkl\_vals[i]) \* (user\_vals[i] - mkl\_vals[i]);

for (int i = 0; i <noNul\_cnt; i++)

if(user\_rows[i]!=mkl\_rows[i])

rows\_eqal=false;

for (int i = 0; i <m\_size+1; i++)

if(user\_index[i]!=mkl\_index[i])

inds\_eqal=false;

if ((diff\_val < 1e-4) &&(rows\_eqal)&&(inds\_eqal))

{

checker\_result.write\_message ("AC. All are equal.");

checker\_result.write\_verdict (verdict::AC);

}

else

{

if(!rows\_eqal)

checker\_result.write\_message ("WA. Wrong row");

if(!inds\_eqal)

checker\_result.write\_message ("WA. Wrong index");

if(diff\_val>=1e-4)

checker\_result.write\_message ("WA. Wrong values");

checker\_result.write\_verdict (verdict::WA);

}

fclose(mkl\_file);

fclose(user\_file);

return 0;

}

# Оценка ускорения при распараллеливании алгоритма

Ниже приведены таблицы с замерами времени выполнения программ на разном числе потоков:

OpenMP

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| матрица | 1 поток | 2 потока | 3 потока | 4 потока |
| 1567х1567 по 88 элементов в каждом столбце | 2,9912 | 2,1943 | 1,58192 | 1,14028 |
| 2173х2173 по 46 элементов в каждом столбце | 3,60562 | 2,64106 | 1,86502 | 1,43511 |
| 5337х5337 по 16 элементов | 6,07382 | 3,43533 | 2,33679 | 1,80004 |
| 1111х1111 по 33 элемента | 0,921062 | 0,680873 | 0,495649 | 0,367893 |
| 2203х2203 по 7 элементов | 0,841065 | 0,475846 | 0,326794 | 0,250365 |
| 987х987 по 16 элементов | 0,377847 | 0,262356 | 0,184923 | 0,118146 |

Intel TBB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| размер | кол-во элементов в каждой строке | 1 поток | 2 потока | 4 потока |
| 987 | 16 | 0.032273 | 0.0229587 | 0.0173418 |
| 1111 | 33 | 0.075144 | 0.0554704 | 0.0409254 |
| 1567 | 88 | 0.321903 | 0.210182 | 0.136159 |
| 2203 | 7 | 0.0778093 | 0.0294243 | 0.0192736 |
| 5337 | 16 | 0.470842 | 0.290261 | 0.23491 |
| 2173 | 46 | 0.221703 | 0.158201 | 0.0911708 |

В среднем ускорение при использовании четырёх потоков 2.5 или 3-кратное.

Путём изменения количества порций вычислений (**grainsize**) в Intel TBB или параметрами **schedule** в OpenMP можно добиться повышения производительности на заданном числе потоков.