#include<iostream>

#include<string>

#include<fstream>

#include<sstream>

#include<windows.h>

#include<bitset>

#include <immintrin.h>//AVX512

#include<windows.h>

#include <immintrin.h>//AVX

#include <xmmintrin.h>

#include <intrin.h>

#include <omp.h>

#include"mpi.h"

using namespace std;

//消元行列数（行数）、被消元行行数、消元子个数

const int col=130;

const int beEliRow=8;

const int EliNum=22;

int ci = 0;

int MPI\_size = 1;//记录进程数

const int row = col+1000;

int num=(col-1)/32+1; //需要的int型元素的数量,num=119

int g;

int tmp = 0;

//类dataMatrix,两个成员：首元素firstCol，数组指针ByteMatrix

//采用int型数组存储矩阵，一个int中可以囊括32个布尔数据

class dataMatrix{

public:

int firstCol;

int\* Matrix;

dataMatrix(){firstCol=-1;

Matrix=new int[num];

for(int i=0;i<num;i++){

Matrix[i]=0;

}

};//初始化函数

void Insert(int x);//插入数据

bool isNull();//判断某行是否为NULL

void OPxor(dataMatrix );//消元子与被消元子异或操作

};

static dataMatrix \*Eli=new dataMatrix[row];//消元子矩阵

static dataMatrix \*beEli=new dataMatrix[beEliRow];//被消元子矩阵

bool dataMatrix::isNull(){

if(firstCol==-1){

return 1;

}

return 0;

}

void dataMatrix::Insert(int x){//输入列数x

if(firstCol==-1){

firstCol = x;

}

//Matrix[x/32]|=(1<<(x%32));

Matrix[x/32]|=(1 << (x%32));

}

void dataMatrix::OPxor(dataMatrix x){

//并行化：

for(int i=0;i<num;i++){

this->Matrix[i] ^= x.Matrix[i];

}

/\*int i;

for(i=num-16;i>=0;i=i-16){

//cout<<"i="<<i<<endl;

\_\_m512i t1 = \_mm512\_loadu\_si512((\_\_m512i \*)&Matrix[i]);

\_\_m512i t2 = \_mm512\_loadu\_si512((\_\_m512i \*)&x.Matrix[i]);

\_\_m512i t3 = \_mm512\_xor\_si512(t1, t2);

cout<<"异或前Matrix：";

for(int k=0;k<8;k++){

// cout<<bitset<sizeof(R[i])\*8>(R[i])<<endl;

cout<<bitset<sizeof(Matrix[i+k])\*8>(Matrix[i+k])<<',';

}

cout<<endl;

cout<<"异或前x.Matrix：";

for(int k=0;k<8;k++){

cout<<bitset<sizeof(x.Matrix[i+k])\*8>(x.Matrix[i+k])<<',';

}

cout<<endl;

\_mm512\_storeu\_si512((\_\_m512i \*)&Matrix[i], t3);

cout<<"异或后Matrix：";

for(int k=0;k<8;k++){

cout<<bitset<sizeof(Matrix[i+k])\*8>(Matrix[i+k])<<',';

}

cout<<endl;

cout<<"异或后x.Matrix：";

for(int k=0;k<8;k++){

cout<<bitset<sizeof(x.Matrix[i+k])\*8>(x.Matrix[i+k])<<',';

}

cout<<endl;

}

if((i+16)!=0){

for(int j=0;(j<num)&&(j<(i+16));j++)

Matrix[j]=Matrix[j]^x.Matrix[j];

}\*/

//更改首列数

for(int i=num-1;i>=0;i--){

for(int j=31;j>=0;j--)

if((Matrix[i]&(1 << j))!=0){

this->firstCol=i\*32+j;

//cout<<"计算更新firstCol:"<<firstCol<<' ';

return;

}

}

firstCol=-1;

}

string c[11][2]={{"被消元行1.txt","消元子1.txt"},

{"被消元行2.txt","消元子2.txt"},

{"被消元行3.txt","消元子3.txt"},

{"被消元行4.txt","消元子4.txt"},

{"被消元行5.txt","消元子5.txt"},

{"被消元行6.txt","消元子6.txt"},

{"被消元行7.txt","消元子7.txt"},

{"被消元行8.txt","消元子8.txt"},

{"被消元行9.txt","消元子9.txt"},

{"被消元行10.txt","消元子10.txt"},

{"被消元行11.txt","消元子11.txt"},

};

bool parallel = 1;

//读文件数据

void readtxt(string c1,string c2){

ifstream data;

//被消元行

data.open(c1);

string x;

int r=0;//行数

while(getline(data,x)){

istringstream temp(x);

int a;//列数

while(temp>>a){

beEli[r].Insert(a);

}

r++;

}

data.close();

//消元行

data.open(c2);

string X;

int R;

while(getline(data,X)){

R=-1;

istringstream temp(X);

int a;

while(temp>>a){//读入列数a

if(R==-1){//最初第一个读入的列数也是消元子的行数

R=a;

}

Eli[R].Insert(a);

}

}

data.close();

}

dataMatrix\* bri = new dataMatrix[MPI\_size];

void gauss(){

//获取进程号rank

int rank;

MPI\_Status status;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

//开始计时

long long head, tail , freq ;

QueryPerformanceFrequency((LARGE\_INTEGER \*)&freq );

QueryPerformanceCounter((LARGE\_INTEGER \*)&head);

//划分不同进程对应的任务

int r1 = rank \* (beEliRow /MPI\_size), r2 = (rank == MPI\_size - 1) ? beEliRow - 1:(rank + 1)\*(beEliRow / MPI\_size)-1;

int i, j, k;

//#pragma omp parallel if(parallel),num\_threads(num\_thread),private(i,j)

for (i = col - 1; i >= 0; i--) {

//如果消元子存在，执行异或消去计算

if (!Eli[i].isNull()) {

//#pragma omp for

for (j = r1; j <= r2; j++) {

if (beEli[j].firstCol == i)

beEli[j].OPxor(Eli[i]);

}

}

else {

//消元子不存在

g = -1;

int t = -1;

//如果不是0号进程:执行升格消元子任务

if (rank != 0) {

//#pragma omp single

for (k = r1; k <= r2; k++)

if (beEli[k].firstCol == i) {

bri[rank] = beEli[k];

t = k;

MPI\_Send(&t, 1, MPI\_INT, 0, k + beEliRow \* 4 + 3, MPI\_COMM\_WORLD);

MPI\_Send(&bri[rank].Matrix[0], num, MPI\_INT, 0, k + 3, MPI\_COMM\_WORLD);

MPI\_Send(&bri[rank].firstCol, 1, MPI\_INT, 0, k + beEliRow + 3, MPI\_COMM\_WORLD);

break;

}

if (k > r2) {

//向0号进程传回数据

MPI\_Send(&t, 1, MPI\_INT, 0, k + beEliRow \* 4 + 3, MPI\_COMM\_WORLD);

MPI\_Send(&bri[rank].Matrix[0], num, MPI\_INT, 0, k + 3, MPI\_COMM\_WORLD);

MPI\_Send(&bri[rank].firstCol, 1, MPI\_INT, 0, k + beEliRow + 3, MPI\_COMM\_WORLD);

}

}

else {

//0号进程：接受其他进程发来的数据，同时也进行升格消元子任务

//#pragma omp single

for (k = MPI\_size - 1; k > 0; k--) {

MPI\_Recv(&t, 1, MPI\_INT, k, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&bri[k].Matrix[0], num, MPI\_INT, k, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&bri[k].firstCol, 1, MPI\_INT, k, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

if (t != -1) {

Eli[i] = bri[k];

g = t;

}

}

//#pragma omp single

for (k = r1; k <= r2; k++)

if (beEli[k].firstCol == i) {

Eli[i] = beEli[k];

g = k;

break;

}

//#pragma omp for

for (k = 1; k < MPI\_size; k++) {

int t1 = k \* (beEliRow/MPI\_size), t2=(k==MPI\_size-1) ? beEliRow-1:(k + 1)\*(beEliRow/MPI\_size)-1;

MPI\_Send(&g, 1, MPI\_INT, k, 0, MPI\_COMM\_WORLD);

if (g != -1 && t2 >= g) {

MPI\_Send(&Eli[i].Matrix[0], num, MPI\_INT, k, 1, MPI\_COMM\_WORLD);

MPI\_Send(&Eli[i].firstCol, 1, MPI\_INT, k, 2, MPI\_COMM\_WORLD);

}

}

}

//MPI\_Barrier(MPI\_COMM\_WORLD);

if (rank != 0) {

//接受数据

MPI\_Recv(&g, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

if (g != -1 && r2 >= g) {

MPI\_Recv(&Eli[i].Matrix[0], num, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&Eli[i].firstCol, num, MPI\_INT, 0, 2, MPI\_COMM\_WORLD, &status);

}

}

if (g != -1 && r2 >= g) {

//#pragma omp for

for (j = r1; j <= r2; j++) {

if (beEli[j].firstCol == i && j != g)

beEli[j].OPxor(Eli[i]);

}

}

}

}

if (rank != 0)

//传回0号进程

for (k = r1; k <= r2; k++) {

MPI\_Send(&beEli[k].Matrix[0],num,MPI\_INT, 0, k + 3 + beEliRow \* 2, MPI\_COMM\_WORLD);

MPI\_Send(&beEli[k].firstCol,1,MPI\_INT,0,k+3+beEliRow\*3,MPI\_COMM\_WORLD);

}

else

for (k = 1; k < MPI\_size; k++) {

int t1 = k\*(beEliRow/MPI\_size), t2 = (k == MPI\_size - 1)?beEliRow - 1 : (k + 1)\*(beEliRow/MPI\_size) - 1;

for (int q = t1; q <= t2; q++) {

MPI\_Recv(&beEli[q].Matrix[0],num, MPI\_INT, k, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&beEli[q].firstCol, 1, MPI\_INT, k, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

}

}

QueryPerformanceCounter((LARGE\_INTEGER \*)&tail );

if (rank == 0) {

cout<<"time:"<<(tail - head)\*1000.0/freq<<endl;//单位ms

}

}

int main(){

/\*

MPI\_Init(0, 0);

int rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

long long head, tail , freq ;

QueryPerformanceFrequency((LARGE\_INTEGER \*)&freq );

//读取文件

QueryPerformanceCounter((LARGE\_INTEGER \*)&head);

readtxt(c[ci][0],c[ci][1]);

QueryPerformanceCounter((LARGE\_INTEGER \*)&tail );

if(rank==0)

cout<<"read:"<<(tail - head)\*1000.0/freq<<endl;//单位ms

//高斯消去

QueryPerformanceCounter((LARGE\_INTEGER \*)&head);

gauss();

QueryPerformanceCounter((LARGE\_INTEGER \*)&tail );

if(rank==0)

cout<<"opera:"<<(tail - head)\*1000.0/freq<<endl;//单位ms

\*/

MPI\_Init(0, 0);

readtxt(c[ci][0],c[ci][1]);

gauss();

//MPI\_Comm\_size(MPI\_COMM\_WORLD, &MPI\_size);

MPI\_Finalize();

}