#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<pthread.h>

#include<semaphore.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 NUM\_THREADS = 6;

const int row = col+1000;

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

//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]|=(0b10000000000000000000000000000000 >>(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]&(0b10000000000000000000000000000000>>j))!=0){

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

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

return;

}

}

firstCol=-1;

}

//pthread初始化

typedef struct{//线程ID

int t\_id;

}threadparam\_t;

pthread\_barrier\_t b1;

pthread\_barrier\_t b2;

pthread\_barrier\_t b3;

int x=0;

//特殊高斯消去-核心计算

//对于每一个t\_id线程

void \*threadfunc(void \*param){

threadparam\_t \*p=(threadparam\_t \*)param;

int t\_id=p->t\_id;

//对于每一个线程，遍历col列

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

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

//如果对应行消元子存在

//任务划分（循环划分）

for(int j=t\_id;j<beEliRow;j+=NUM\_THREADS){

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

//消去计算

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

beEli[j].Matrix[k]^=Eli[i].Matrix[k];

}

bool f=1;

for(int p=num-1;p>=0&&f;p--){

for(int k=31;k>=0&&f;k--)

if((beEli[j].Matrix[p]&(0b10000000000000000000000000000000>>k))!=0){

beEli[j].firstCol=p\*32+k;

f=0;

}

}

if(f){

beEli[j].firstCol=-1;

}

}

}

}

else{

//线程同步

pthread\_barrier\_wait(&b1);

if(t\_id==0){

//0号线程：消元子升格

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

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

{

Eli[i]=beEli[j];

x=j+1;

break;

}

x=j+2;

}

}

//线程同步

pthread\_barrier\_wait(&b2);

int temp=t\_id;

while(temp<x){

temp+=NUM\_THREADS;

}

for(int j=temp;j<beEliRow;j+=NUM\_THREADS){

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

//消去计算

for(int k=0;k<num;k++)

beEli[j].Matrix[k]^=Eli[i].Matrix[k];

bool f=1;

for(int p=num-1;p>=0&&f;p--){

for(int k=31;k>=0&&f;k--){

if((beEli[j].Matrix[p]&(0b10000000000000000000000000000000>>k))!=0){

beEli[j].firstCol=p\*32+k;

f=0;

}

}

}

if(f){

beEli[j].firstCol=-1;

}

}

}

}

pthread\_exit(NULL);

}

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"},

};

//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();

}

void gauss(){

//初始化阻塞量

pthread\_barrier\_init(&b1,NULL,NUM\_THREADS);

pthread\_barrier\_init(&b2,NULL,NUM\_THREADS);

pthread\_barrier\_init(&b3,NULL,NUM\_THREADS);

//初始化线程（静态线程）并调用threadfunc函数

pthread\_t handles[NUM\_THREADS];

threadparam\_t param[NUM\_THREADS];

for(int t\_id=0;t\_id<NUM\_THREADS;t\_id++){

param[t\_id].t\_id=t\_id;

pthread\_create(&handles[t\_id],NULL,threadfunc,(void\*)&param[t\_id]);

}

//同步

for(int i=0;i<NUM\_THREADS;i++)

pthread\_join(handles[i],NULL);

pthread\_barrier\_destroy(&b1);

pthread\_barrier\_destroy(&b2);

pthread\_barrier\_destroy(&b3);

/\*

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

//cout<<i<<endl;

while(!beEli[i].isNull()){

int FCol=beEli[i].firstCol;

if(!Eli[FCol].isNull()){//若消元子存在

beEli[i].OPxor(Eli[FCol]);//异或计算

}

else{

//被消元子升格为消元子，不再参与后续计算

Eli[FCol]=beEli[i];

beEli[i].firstCol = -1;

break;

}

}

//cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

}\*/

}

int main(){

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 );

cout<<(tail - head)\*1000.0/freq<<' ';//单位ms

//高斯消去

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

gauss();

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

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

}