#include <iostream>

#include <fstream>

#include <sstream>

#include <cmath>

#include <mpi.h>

using namespace std;

extern "C" {

 void Cblacs\_get(int, int, int \*);

 void Cblacs\_gridinit(int \*, const char \*, int, int);

 void Cblacs\_pcoord(int, int, int \*, int \*);

 void Cblacs\_gridexit(int);

 void Cdgerv2d(int, int, int, double \*, int, int, int);

 void Cdgesd2d(int, int, int, double \*, int, int, int);

 int numroc\_(int \*, int \*, int \*, int \*, int \*);

 void descinit\_(int \*, int \*, int \*, int \*, int \*, int \*, int \*, int \*, int \*, int \*);

 void pdgehrd\_(int \*, int \*, int \*, double \*, int \*, int \*, int \*, double \*, double \*, int \*, int \*);

 void pdhseqr\_(char \*, char \*, int \*, int \*, int \*, double \*, int \*, double \*, double \*, double \*,

 int \*, double \*, int \*, int \*, int \*, int \*);

 double dsecnd\_();

}

void read\_text\_matrix(ifstream &file, double \*A, int N, int M)

{

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

 {

 for (int c = 0; c < M; c++)

 {

 file >> A[N \* c + r];

 }

 }

}

void matrix\_transpose(double \*arr, double \*A, int N, int M)

{

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

 {

 for (int j = 0; j < M; j++)

 {

 A[j \* N + i] = arr[i \* M + j];

 }

 }

}

void read\_binary\_matrix(ifstream &file, double \*A, int N, int M)

{

 double \*arr = new double[N \* M];

 file.read((char\*)arr, N \* M \* sizeof(double));

 matrix\_transpose(arr, A, N, M);

 delete[] arr;

}

void read\_matrix(char \*fname, double \*A, int N, int M, string mode)

{

 ifstream file(fname);

 if (mode == "binary")

 {

 read\_binary\_matrix(file, A, N, M);

 }

 else

 {

 read\_text\_matrix(file, A, N, M);

 }

 file.close();

}

void print\_eigenvalues(char \*fname, double \*wr, double \*wi, int N)

{

 ofstream file(fname);

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

 {

 file << wr[i];

 if (wi[i] != 0)

 {

 if (wi[i] > 0) file << "+";

 file << wi[i] << "i";

 }

 file << endl;

 }

 file.close();

}

void make\_hessenberg(double \*A, int N, int \*desc)

{

 int info;

 int iOne = 1;

 int ilo = 1, ihi = N;

 int lwork = -1;

 double work\_size;

 double \*work;

 double \*tau = new double[N - 1];

 pdgehrd\_(&N, &ilo, &ihi, A, &iOne, &iOne, desc, tau, &work\_size, &lwork, &info);

 lwork = (int)work\_size;

 work = new double[lwork];

 pdgehrd\_(&N, &ilo, &ihi, A, &iOne, &iOne, desc, tau, work, &lwork, &info);

 delete[] work;

 delete[] tau;

}

void compute\_eigenvalues(double \*A, int N, int \*desc, double \*z, int \*descz, double \*wr, double \*wi)

{

 int info;

 int ilo = 1, ihi = N;

 char job = 'E', compz = 'N';

 int lwork = -1, liwork = -1;

 double \*work = new double[N];

 int \*iwork = new int[N];

 pdhseqr\_(&job, &compz, &N, &ilo, &ihi, A, desc, wr, wi, z, descz, work, &lwork, iwork, &liwork, &info);

 lwork = work[0] \* 2;

 liwork = lwork;

 delete[] work;

 delete[] iwork;

 work = new double[lwork];

 iwork = new int[liwork];

 pdhseqr\_(&job, &compz, &N, &ilo, &ihi, A, desc, wr, wi, z, descz, work, &lwork, iwork, &liwork, &info);

 delete[] work;

 delete[] iwork;

}

void get\_eigenvalues(double \*A, int N, int M, int Nb, int Mb, int nrows, int ncols, int ctxt, double \*wr, double \*wi)

{

 int info;

 int iZERO = 0;

 int desc[9], descz[9];

 double \*z = new double[nrows \* ncols];

 descinit\_(desc, &N, &M, &Nb, &Mb, &iZERO, &iZERO, &ctxt, &nrows, &info);

 descinit\_(descz, &N, &M, &Nb, &Mb, &iZERO, &iZERO, &ctxt, &nrows, &info);

 make\_hessenberg(A, N, desc);

 compute\_eigenvalues(A, N, desc, z, descz, wr, wi);

 delete[] z;

}

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

{

 int mpirank, numproc;

 MPI\_Init(&argc, &argv);

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

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

 bool mpiroot = (mpirank == 0);

 if (argc < 5)

 {

 if (mpiroot)

 {

 cout << "Usage: eigenvalues matrix\_file matrix\_size result\_file matrix\_mode[binary or text]" << endl;

 }

 MPI\_Finalize();

 return 0;

 }

 int N, M, Nb, Mb;

 double \*A\_glob = NULL, \*A\_loc = NULL;

 if (mpiroot)

 {

 istringstream iss(argv[2]);

 iss >> N;

 M = N;

 A\_glob = new double[N \* M];

 read\_matrix(argv[1], A\_glob, N, M, argv[4]);

 }

 int ctxt, myrow, mycol;

 int procrows, proccols;

 procrows = sqrt(numproc);

 proccols = procrows;

 Cblacs\_get(0, 0, &ctxt);

 Cblacs\_gridinit(&ctxt, "Row-major", procrows, proccols);

 Cblacs\_pcoord(ctxt, mpirank, &myrow, &mycol);

 MPI\_Bcast(&N, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

 M = N;

 Nb = N / procrows;

 Mb = Nb;

 int iZERO = 0;

 int nrows = numroc\_(&N, &Nb, &myrow, &iZERO, &procrows);

 int ncols = numroc\_(&M, &Mb, &mycol, &iZERO, &proccols);

 A\_loc = new double[nrows \* ncols];

 for (int i = 0; i < nrows \* ncols; i++) A\_loc[i] = 0.0;

 int sendr = 0, sendc = 0, recvr = 0, recvc = 0;

 for (int r = 0; r < N; r += Nb, sendr = (sendr + 1) % procrows)

 {

 sendc = 0;

 int nr = Nb;

 if (N - r < Nb) nr = N - r;

 for (int c = 0; c < M; c += Mb, sendc = (sendc + 1) % proccols)

 {

 int nc = Mb;

 if (M - c < Mb) nc = M - c;

 if (mpiroot)

 {

 Cdgesd2d(ctxt, nr, nc, A\_glob + N \* c + r, N, sendr, sendc);

 }

 if (myrow == sendr && mycol == sendc)

 {

 Cdgerv2d(ctxt, nr, nc, A\_loc + nrows \* recvc + recvr, nrows, 0, 0);

 recvc = (recvc + nc) % ncols;

 }

 }

 if (myrow == sendr) recvr = (recvr + nr) % nrows;

 }

 double \*wr = new double[N];

 double \*wi = new double[N];

 double start\_time = dsecnd\_();

 get\_eigenvalues(A\_loc, N, M, Nb, Mb, nrows, ncols, ctxt, wr, wi);

 double end\_time = dsecnd\_();

 if (mpiroot)

 {

 cout << "Matrix size: " << N << endl;

 cout << "Processes number: " << numproc << endl;

 cout << "Time: " << end\_time - start\_time << endl;

 }

 if (mpiroot)

 {

 print\_eigenvalues(argv[3], wr, wi, N);

 }

 delete[] A\_glob;

 delete[] A\_loc;

 delete[] wr;

 delete[] wi;

 Cblacs\_gridexit(ctxt);

 MPI\_Finalize();

 return 0;

}