#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define X 250

#define Y 250

void init\_field(int\* field);

void separate\_field(int\* field, int\* scounts, int\* displs, int\*\* part\_field, int size, int rank);

void find\_neighbours(int\* part\_field, int length, int\* up\_neighbour, int\* down\_neighbour, int size, int rank);

void count\_stop\_vec(int\*\* iter\_field, int\* stop\_vec, int iter, int len);

int\* change\_stop\_vec(int\* stop\_vec, int size, int rank, int len, MPI\_Request\* change\_request);

void count\_alive(int\* current\_field, int\* new\_field, int len);

void count\_up\_rows(int\* current\_field, int\* new\_field, int\* up\_neighbour);

void count\_down\_rows(int\* current\_field, int\* new\_field, int\* down\_neighbour, int len);

int check\_stop\_vec(int\* new\_stop\_vec, int size, int len, int rank);

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

MPI\_Init(&argc, &argv);

int rank, size;

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

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

int field[X \* Y];

init\_field(field);

double start;

if (rank == 0) {

start = MPI\_Wtime();

}

int\* scounts = new int[size];

int\* displs = new int[size];

int\*\* iter\_field = new int\* [10000];

separate\_field(field, scounts, displs, iter\_field, size, rank);

int\* up\_neighbour = new int[X];

int\* down\_neighbour = new int[X];

int\* up\_row = new int[X];

int\* down\_row = new int[X];

int it = 0;

int stop\_flag = 0;

int stop\_sum;

while (1) {

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

up\_row[i] = iter\_field[it][i];

down\_row[i] = iter\_field[it][scounts[rank] - X + i];

}

MPI\_Request send\_up\_request;

MPI\_Isend(up\_row, X, MPI\_INT, (rank + size - 1) % size, 123, MPI\_COMM\_WORLD, &send\_up\_request);

MPI\_Request send\_down\_request;

MPI\_Isend(down\_row, X, MPI\_INT, (rank + 1) % size, 124, MPI\_COMM\_WORLD, &send\_down\_request);

MPI\_Request recv\_up\_request;

MPI\_Irecv(up\_neighbour, X, MPI\_INT, (rank + size - 1) % size, 124, MPI\_COMM\_WORLD, &recv\_up\_request);

MPI\_Request recv\_down\_request;

MPI\_Irecv(down\_neighbour, X, MPI\_INT, (rank + 1) % size, 123, MPI\_COMM\_WORLD, &recv\_down\_request);

int\* stop\_vec = new int[it];

int\* new\_stop\_vec;

MPI\_Request change\_request;

count\_stop\_vec(iter\_field, stop\_vec, it, scounts[rank]);

new\_stop\_vec = change\_stop\_vec(stop\_vec, size, rank, it, &change\_request);

iter\_field[it + 1] = new int[scounts[rank]];

count\_alive(iter\_field[it], iter\_field[it + 1], scounts[rank]);

MPI\_Wait(&send\_up\_request, MPI\_STATUS\_IGNORE);

MPI\_Wait(&recv\_up\_request, MPI\_STATUS\_IGNORE);

count\_up\_rows(iter\_field[it], iter\_field[it + 1], up\_neighbour);

MPI\_Wait(&send\_down\_request, MPI\_STATUS\_IGNORE);

MPI\_Wait(&recv\_down\_request, MPI\_STATUS\_IGNORE);

count\_down\_rows(iter\_field[it], iter\_field[it + 1], down\_neighbour, scounts[rank]);

MPI\_Wait(&change\_request, MPI\_STATUS\_IGNORE);

if (check\_stop\_vec(new\_stop\_vec, size, it, rank)) {

stop\_flag = 1;

}

MPI\_Allreduce(&stop\_flag, &stop\_sum, 1, MPI\_INT, MPI\_SUM, MPI\_COMM\_WORLD);

delete[] stop\_vec;

delete[] new\_stop\_vec;

if (stop\_sum != 0) {

if (rank == 0) {

printf("%d\n", it);

}

break;

}

it++;

}

if (rank == 0) {

double end = MPI\_Wtime();

printf("Time - %lf\n", end - start);

}

for (int i = 0; i <= it + 1; i++) {

delete[] iter\_field[i];

}

delete[] iter\_field;

delete[] scounts;

delete[] displs;

delete[] up\_neighbour;

delete[] down\_neighbour;

delete[] up\_row;

delete[] down\_row;

MPI\_Finalize();

return 0;

}

void init\_field(int\* field) {

for (int i = 0; i < X \* Y; i++) {

field[i] = 0;

}

field[1] = 1;

field[X + 2] = 1;

field[2 \* X + 0] = 1;

field[2 \* X + 1] = 1;

field[2 \* X + 2] = 1;

}

void separate\_field(int\* field, int\* scounts, int\* displs, int\*\* part\_field, int size, int rank) {

int cnt = Y / size;

int rmd = Y % size;

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

scounts[i] = cnt \* X;

if (rmd > 0) {

scounts[i] += X;

rmd--;

}

}

displs[0] = 0;

for (int i = 1; i < size; i++) {

displs[i] = displs[i - 1] + scounts[i - 1];

}

\*part\_field = new int[scounts[rank]];

MPI\_Scatterv(field, scounts, displs, MPI\_INT, \*part\_field, scounts[rank], MPI\_INT, 0, MPI\_COMM\_WORLD);

}

void count\_stop\_vec(int\*\* iter\_field, int\* stop\_vec, int iter, int len) {

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

stop\_vec[i] = 1;

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

if (iter\_field[iter][j] != iter\_field[i][j]) {

stop\_vec[i] = 0;

break;

}

}

}

}

int\* change\_stop\_vec(int\* stop\_vec, int size, int rank, int len, MPI\_Request\* change\_request) {

int cnt = len / size;

int rmd = len % size;

int\* send\_scounts = new int[size];

int\* send\_displs = new int[size];

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

send\_scounts[i] = cnt;

if (rmd > 0) {

send\_scounts[i] += 1;

rmd--;

}

}

send\_displs[0] = 0;

for (int i = 1; i < size; i++) {

send\_displs[i] = send\_displs[i - 1] + send\_scounts[i - 1];

}

int\* recv\_scounts = new int[size];

int\* recv\_displs = new int[size];

if (rank < len % size) {

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

recv\_scounts[i] = send\_scounts[0];

recv\_displs[i] = i \* send\_scounts[0];

}

}

else {

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

recv\_scounts[i] = send\_scounts[size - 1];

recv\_displs[i] = i \* send\_scounts[size - 1];

}

}

int\* new\_stop\_vec = new int[send\_scounts[rank] \* size];

MPI\_Ialltoallv(stop\_vec, send\_scounts, send\_displs, MPI\_INT,

new\_stop\_vec, recv\_scounts, recv\_displs, MPI\_INT,

MPI\_COMM\_WORLD, change\_request);

delete[] send\_scounts;

delete[] send\_displs;

delete[] recv\_scounts;

delete[] recv\_displs;

return new\_stop\_vec;

}

void count\_alive(int\* current\_field, int\* new\_field, int len) {

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

new\_field[i] = current\_field[i];

}

for (int i = X; i < len - X; i++) {

int alive = 0;

int offset;

offset = 1;

if (i % X + 1 == X) {

offset = 1 - X;

}

alive += current\_field[i + offset];

alive += current\_field[i + X + offset];

alive += current\_field[i - X + offset];

offset = -1;

if (i % X - 1 < 0) {

offset = X - 1;

}

alive += current\_field[i + offset];

alive += current\_field[i + X + offset];

alive += current\_field[i - X + offset];

alive += current\_field[i + X];

alive += current\_field[i - X];

if (alive == 3) {

new\_field[i] = 1;

}

if (alive < 2 || alive > 3) {

new\_field[i] = 0;

}

}

}

void count\_up\_rows(int\* current\_field, int\* new\_field, int\* up\_neighbour) {

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

int alive = 0;

int offset;

offset = 1;

if (i % X + 1 == X) {

offset = 1 - X;

}

alive += current\_field[i + offset];

alive += current\_field[i + X + offset];

alive += up\_neighbour[i + offset];

offset = -1;

if (i % X - 1 < 0) {

offset = X - 1;

}

alive += current\_field[i + offset];

alive += current\_field[i + X + offset];

alive += up\_neighbour[i + offset];

alive += current\_field[i + X];

alive += up\_neighbour[i];

if (alive == 3) {

new\_field[i] = 1;

}

if (alive < 2 || alive > 3) {

new\_field[i] = 0;

}

}

}

void count\_down\_rows(int\* current\_field, int\* new\_field, int\* down\_neighbour, int len) {

for (int i = len - X; i < len; i++) {

int alive = 0;

int offset;

offset = 1;

if (i % X + 1 == X) {

offset = 1 - X;

}

alive += current\_field[i + offset];

alive += down\_neighbour[i % X + offset];

alive += current\_field[i - X + offset];

offset = -1;

if (i % X - 1 < 0) {

offset = X - 1;

}

alive += current\_field[i + offset];

alive += down\_neighbour[i % X + offset];

alive += current\_field[i - X + offset];

alive += down\_neighbour[i % X];

alive += current\_field[i - X];

if (alive == 3) {

new\_field[i] = 1;

}

if (alive < 2 || alive > 3) {

new\_field[i] = 0;

}

}

}

int check\_stop\_vec(int\* new\_stop\_vec, int size, int len, int rank) {

int offset;

if (rank < len % size) {

offset = len / size + 1;

}

else {

offset = len / size;

}

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

int check = 1;

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

if (new\_stop\_vec[i + j \* offset] == 0) {

check = 0;

break;

}

}

if (check == 1) {

return 1;

}

}

return 0;

}