#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define TOPOLOGY\_DIM 2

#define ROWS 0

#define COLUMNS 1

int check\_input(int n1, int n2, int n3, int\* dims);

void fill\_matrix(double\* A, double\* B, int n1, int n2, int n3);

void print\_matrix(double\* matrix, int rows, int columns);

void make\_grid(MPI\_Comm\* row\_comm, MPI\_Comm\* col\_comm, MPI\_Comm\* grid\_comm, int\* coords, int\* dims);

void separate\_matrix(double\* A, double\* B, double\* drobA, double\* drobB, int n1, int n2, int n3, MPI\_Comm row\_comm, MPI\_Comm col\_comm, int\* coords, int\* dims);

void mul\_matrix(double\* drobA, double\* drobB, double\* drobC, int drob\_n1, int n2, int drob\_n3);

void union\_matrix(double\* drobC, double\* C, MPI\_Comm grid\_comm, int\* dims, int\* coords, int n1, int n3, int size);

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

MPI\_Init(&argc, &argv);

int dims[TOPOLOGY\_DIM] = { 0,0 };

int coords[TOPOLOGY\_DIM];

int size;

int rank;

double start;

MPI\_Comm row\_comm;

MPI\_Comm col\_comm;

MPI\_Comm grid\_comm;

int exit\_flag = 0;

int n1 = (int)strtol(argv[1], NULL, 10);

int n2 = (int)strtol(argv[2], NULL, 10);

int n3 = (int)strtol(argv[3], NULL, 10);

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

MPI\_Dims\_create(size, TOPOLOGY\_DIM, dims);

make\_grid(&row\_comm, &col\_comm, &grid\_comm, coords, dims);

MPI\_Comm\_rank(grid\_comm, &rank);

if (rank == 0) {

exit\_flag = check\_input(n1, n2, n3, dims);

}

MPI\_Bcast(&exit\_flag, 1, MPI\_INT, 0, grid\_comm);

if (exit\_flag == 1){

MPI\_Finalize();

return 0;

}

double\* A = (double\*)malloc(sizeof(double) \* n1 \* n2);

double\* B = (double\*)malloc(sizeof(double) \* n2 \* n3);

double\* C = (double\*)malloc(sizeof(double) \* n1 \* n3);

double\* drobA = (double\*)malloc(sizeof(double) \* n1 \* n2 / dims[ROWS]);

double\* drobB = (double\*)malloc(sizeof(double) \* n2 \* n3 / dims[COLUMNS]);

double\* drobC = (double\*)calloc(n1 \* n3 / (dims[ROWS] \* dims[COLUMNS]), sizeof(double));

if (rank == 0) {

fill\_matrix(A, B, n1, n2, n3);

start = MPI\_Wtime();

}

separate\_matrix(A, B, drobA, drobB, n1, n2, n3, row\_comm, col\_comm, coords, dims);

mul\_matrix(drobA, drobB, drobC, n1 / dims[ROWS], n2, n3 / dims[COLUMNS]);

union\_matrix(drobC, C, grid\_comm, dims, coords, n1, n3, size);

if (rank == 0) {

//print\_matrix(C, n1, n3);

double end = MPI\_Wtime();

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

}

free(A);

free(B);

free(C);

free(drobA);

free(drobB);

free(drobC);

MPI\_Comm\_free(&grid\_comm);

MPI\_Comm\_free(&col\_comm);

MPI\_Comm\_free(&row\_comm);

MPI\_Finalize();

}

void make\_grid(MPI\_Comm\* row\_comm, MPI\_Comm\* col\_comm, MPI\_Comm\* grid\_comm, int\* coords, int\* dims) {

int periods[TOPOLOGY\_DIM];

int rank;

MPI\_Cart\_create(MPI\_COMM\_WORLD, TOPOLOGY\_DIM, dims, periods, 0, grid\_comm);

MPI\_Comm\_rank(\*grid\_comm, &rank);

MPI\_Cart\_coords(\*grid\_comm, rank, TOPOLOGY\_DIM, coords);

MPI\_Comm\_split(\*grid\_comm, coords[ROWS], coords[COLUMNS], row\_comm);

MPI\_Comm\_split(\*grid\_comm, coords[COLUMNS], coords[ROWS], col\_comm);

}

void fill\_matrix(double\* A, double\* B, int n1, int n2, int n3) {

for (int i = 0; i < n1; i++) {

for (int j = 0; j < n2; j++) {

A[i \* n2 + j] = 1;

}

}

//print\_matrix(A, n1, n2);

for (int i = 0; i < n2; i++) {

for (int j = 0; j < n3; j++) {

B[i \* n3 + j] = i + j;

}

}

//print\_matrix(B, n2, n3);

}

void separate\_matrix(double\* A, double\* B, double\* drobA, double\* drobB, int n1, int n2, int n3, MPI\_Comm row\_comm, MPI\_Comm col\_comm, int\* coords, int\* dims) {

if (coords[COLUMNS] == 0) {

MPI\_Scatter(A, n1 \* n2 / dims[ROWS], MPI\_DOUBLE, drobA, n1 \* n2 / dims[ROWS], MPI\_DOUBLE, 0, col\_comm);

}

if (coords[ROWS] == 0) {

MPI\_Datatype send\_time\_vec;

MPI\_Datatype resized\_send\_time\_vec;

MPI\_Type\_vector(n2, 1, n3, MPI\_DOUBLE, &send\_time\_vec);

MPI\_Type\_commit(&send\_time\_vec);

MPI\_Type\_create\_resized(send\_time\_vec, 0, 1 \* sizeof(double), &resized\_send\_time\_vec);

MPI\_Type\_commit(&resized\_send\_time\_vec);

MPI\_Scatter(B, n3 / dims[COLUMNS], resized\_send\_time\_vec, drobB, n2 \* n3 / dims[COLUMNS], MPI\_DOUBLE, 0, row\_comm);

MPI\_Type\_free(&resized\_send\_time\_vec);

MPI\_Type\_free(&send\_time\_vec);

}

MPI\_Bcast(drobA, n1 \* n2 / dims[ROWS], MPI\_DOUBLE, 0, row\_comm);

MPI\_Bcast(drobB, n2 \* n3 / dims[COLUMNS], MPI\_DOUBLE, 0, col\_comm);

}

void mul\_matrix(double\* drobA, double\* drobB, double\* drobC, int drob\_n1, int n2, int drob\_n3) {

for (int i = 0; i < drob\_n1; i++) {

for (int j = 0; j < drob\_n3; j++) {

for (int k = 0; k < n2; k++) {

drobC[i \* drob\_n3 + j] += drobA[i \* n2 + k] \* drobB[j \* n2 + k];

}

}

}

}

void union\_matrix(double\* drobC, double\* C, MPI\_Comm grid\_comm, int\* dims, int\* coords, int n1, int n3, int size) {

int\* received\_number = (int\*)malloc(sizeof(int) \* size);

int\* displs = (int\*)malloc(sizeof(int) \* size);

MPI\_Datatype received\_vec;

MPI\_Datatype resized\_received\_vec;

MPI\_Type\_vector(n1 / dims[ROWS], n3 / dims[COLUMNS], n3, MPI\_DOUBLE, &received\_vec);

MPI\_Type\_commit(&received\_vec);

MPI\_Type\_create\_resized(received\_vec, 0, n3 / dims[COLUMNS] \* sizeof(double), &resized\_received\_vec);

MPI\_Type\_commit(&resized\_received\_vec);

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

{

received\_number[i] = 1;

MPI\_Cart\_coords(grid\_comm, i, TOPOLOGY\_DIM, coords);

displs[i] = dims[COLUMNS] \* (n1 / dims[ROWS]) \* coords[ROWS] + coords[COLUMNS];

}

MPI\_Gatherv(drobC, n1 \* n3 / (dims[COLUMNS] \* dims[ROWS]), MPI\_DOUBLE, C, received\_number, displs, resized\_received\_vec, 0, grid\_comm);

MPI\_Type\_free(&received\_vec);

MPI\_Type\_free(&resized\_received\_vec);

}

void print\_matrix(double\* matrix, int rows, int columns) {

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

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

printf("%lf ", matrix[i \* columns + j]);

}

printf("\n");

}

printf("\n");

}

int check\_input(int n1, int n2, int n3, int\* dims) {

if ((n1 == 0) || (n2 == 0) || (n3 == 0)){

printf("Wrong format of matrix sizes in arguments:\nArguments: N1 N2 N3, matrix A = N1\*N2 and B = N2\*N3 and C = N1\*N3\n");

return 1;

}

if ((n1 % dims[ROWS] != 0) || (n3 % dims[COLUMNS] != 0)){

printf("Incorrect size of matrices in arguments:\nN1 must be a multiple of dims[ROWS] = %d \nN3 must be a multiple of dims[COLUMNS] = %d \n", dims[ROWS], dims[COLUMNS]);

return 1;

}

return 0;

}

Время выполнения программы в зависимости от размера матриц и количества процессов