Author: Eric Simmons, Yijie Sun

Due day: Saturday, December 17, 11:55 pm

The input: number of rows, number of cols, the matrix (which is a N\*N matrix, row is equal to cols) The file name should be matrix.txt

Example:

5 5

1 2 4 8 4

3 4 4 0 32

5 6 5 23 1

3 3 4 5 0

2 9 8 5 2

The output: The transposed matrix

How to run:

mpicc -o project2 ./project2\_scatter.c

mpirun –n <number of process> ./project2

The number of process must be greater than or equal to the number of rows

// author: Eric Simmons, Yijie Sun

// Project: Transpose a m\*m dense matrix parallely using MPI

// input: text file with the number of row and col, the original matrix followed by the matrix body

// output: the transposed matrix

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

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

//declare variables

int myrank, nprocs;

int nrow, ncol;

//open file and read the number of columns and rows

FILE\* fp;

fp = fopen("matrix.txt", "r+");

fscanf(fp,"%d %d", &nrow, &ncol);

//create 2 dynamic arrays that are contigeous in memory

int (\*matrix)[nrow];

int (\*result)[ncol];

if(myrank == 0){

matrix = malloc(ncol \* sizeof(\*matrix));

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

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

fscanf(fp,"%d", &matrix[i][j]);

}

}

result = malloc(nrow \* sizeof(\*result));

}

fclose(fp);

//initalize MPI environment

MPI\_Init(NULL,NULL);

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

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

// If there are not enough processes exit the program

if(nrow > nprocs)

{

MPI\_Finalize();

if(myrank ==0)

printf("Number of processes need to be >= %d\n", nrow);

exit(0);

}

//create new communicator. If number of processes is greater than the matrix size split the communicator into two groups

//if the process = matrix size just set the new communicator = to MPI\_COMM\_WORLD

MPI\_Comm newworld;

int color;

if(nprocs > nrow){

color = myrank<nrow?0:1;

MPI\_Comm\_split(MPI\_COMM\_WORLD, color, myrank, &newworld);

}else{

newworld = MPI\_COMM\_WORLD;

}

//reinitalize MPI environment

int row\_rank, row\_size;

MPI\_Comm\_rank(newworld, &row\_rank);

MPI\_Comm\_size(newworld, &row\_size);

//process 0 prints out the original matrix

if(myrank == 0){

printf("\n Ori: \n");

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

//create matrix buffers

int local\_row[ncol];

int local\_new\_row[nrow];

//MPI\_Barrier(newworld);

//scatter and reorganize the matrix

MPI\_Scatter(&matrix[row\_rank], ncol, MPI\_INT, &local\_row, ncol, MPI\_INT, 0, newworld);

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

/\*printf("Rank: %d\n", myrank);

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

printf("%d ", local\_row[j]);

}

printf("\n");\*/

free(matrix);

MPI\_Alltoall(&local\_row, 1, MPI\_INT, &local\_new\_row, 1, MPI\_INT, newworld);

/\*

printf("Rank: %d\n", myrank);

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

printf("%d ", local\_new\_row[j]);

}\*/

MPI\_Gather(&local\_new\_row, ncol, MPI\_INT, &result[row\_rank], ncol, MPI\_INT, 0, newworld);

//print out transposed matrix

if(myrank == 0){

printf("\n Result: \n");

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

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

printf("%d ", result[i][j]);

}

printf("\n");

}

}

if(nprocs > nrow){

MPI\_Comm\_free(&newworld);

}

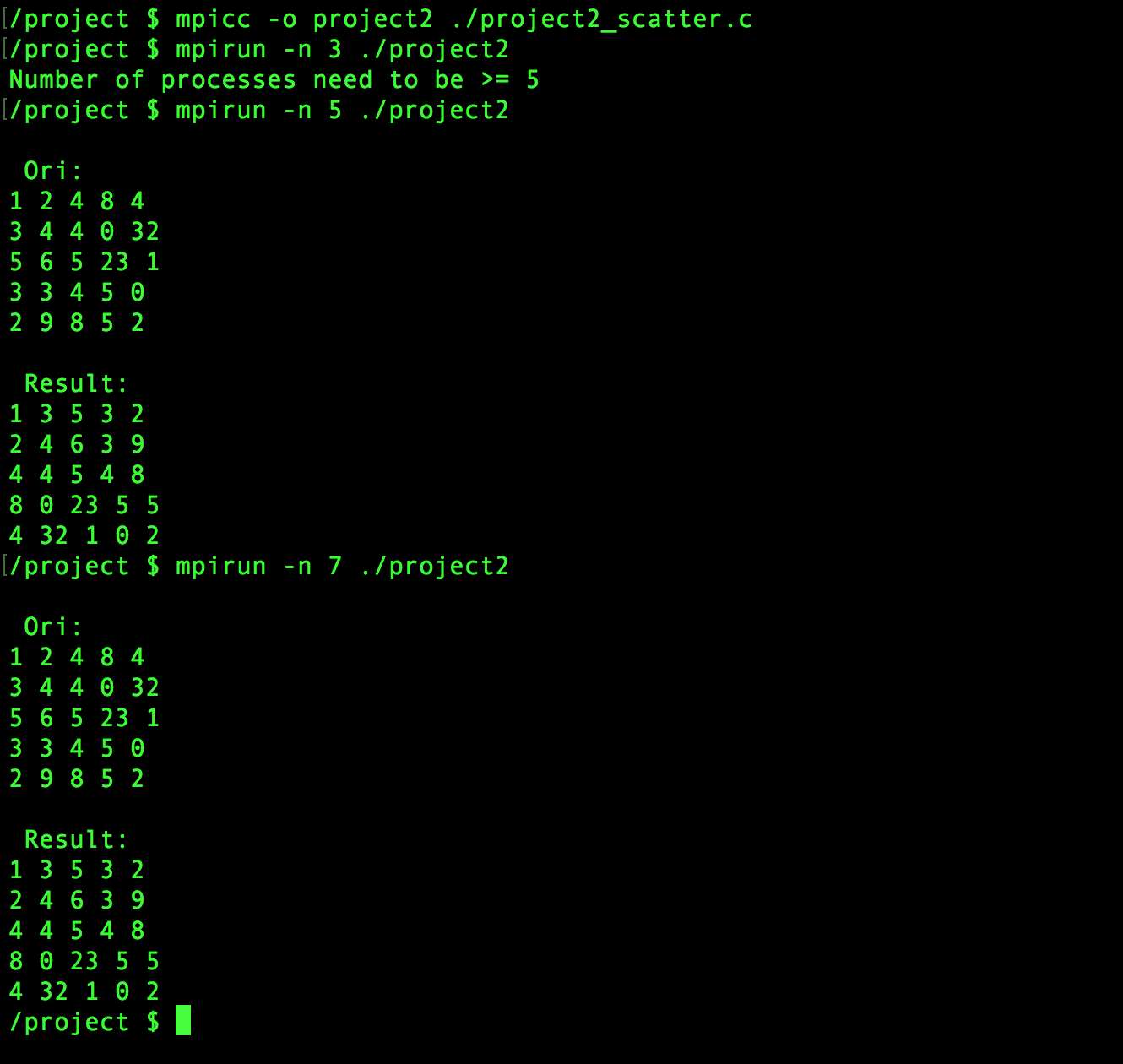
free(result);

MPI\_Finalize();

return 0;

}

The output screenshot:



matrix.txt:

