Lab6 – MPI (grouping data)

ex 2-Dimensional block composition

Complete the following MPI program(compose.c) to compose 2-D arrays from all other processes(2-D grid) to p_0.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mpi.h"

#define N 24

int **malloc_2d(int row, int col)
{
    int **A, *ptr;
    int len, i;

    len = sizeof(float *)*row + sizeof(float)*col*row;
    A = (int **)malloc(len);
    ptr = (int *)(A + row);
    for(i = 0; i < row; i++)
        A[i] = (ptr + col*i);
    return A;
}

main(int argc, char* argv[])
{
    int A[N][N], **local_A;
    int np2, np, pid, local_N, i, j;
    MPI_Status status;
    int tag;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np2);

    np = sqrt(np2);
    local_N = N/np;

    local_A = malloc_2d(local_N, local_N);

    // initializaton of arrays
    if (pid == 0) {
        for (i=0; i<local_N; i++)
            for (j=0; j<local_N; j++)
                A[i][j] = pid;
    }
    else {
        for (i=0; i<local_N; i++)
            for (j=0; j<local_N; j++)
                local_A[i][j] = pid;
    }

// COMPLETE THIS AREA
    // composition
    if (pid != 0)
        for (i=0; i<local_N; i++)
            MPI_Send(...);
    else {
        for (i=0; i<local_N; i++)
            for (j=1; j<np2; j++) {
                x = ...;
                y = ...;
                MPI_Recv(&A[x+i][y], ...);
            }
    }

    // print the result
    if (pid == 0)
        for (i=0; i<N; i++) {
            for (j=0; j<N; j++)
                printf("%3d", A[i][j]);
            printf("₩n");
        }

    MPI_Finalize();
}
```

Tip:

(1) Array A and local_A are 2-D arrays, i.e. A[N][N], local_A[local_N][local_N].

(2) The processors also handle a 2-dimensional layout, so #prococess = np2 = np X np

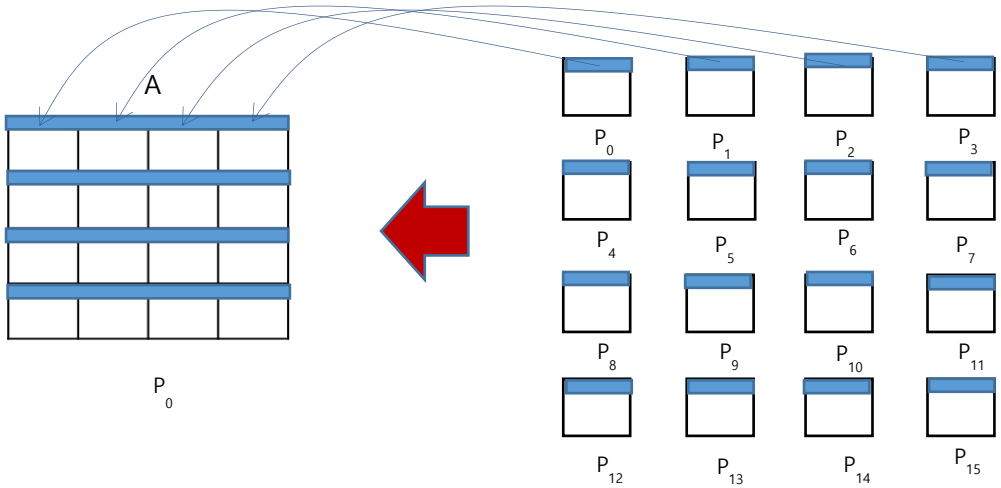
(3) One array(blue segments in the following figure) is send to P_0 in every loop.

(4) Use **MPI_Send()** and **MPI_Recv()**.

(5) Run only 4(=2x2), 9(=3x3) or 16(=4x4) processors for tests.

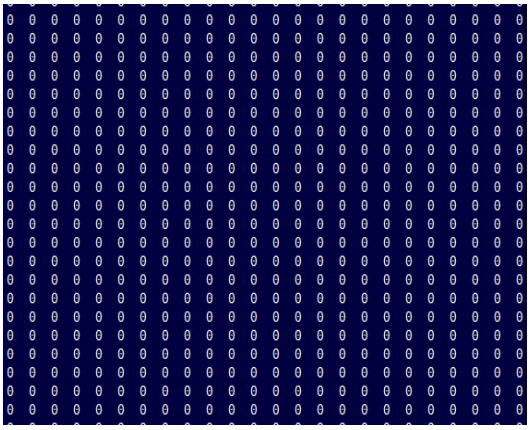
(6) Submit your program(compose.c) when complete.

e.g. N=24, np2 = 16, np = sqrt(np2) = 4, local_N = N/np = 24/4 = 6

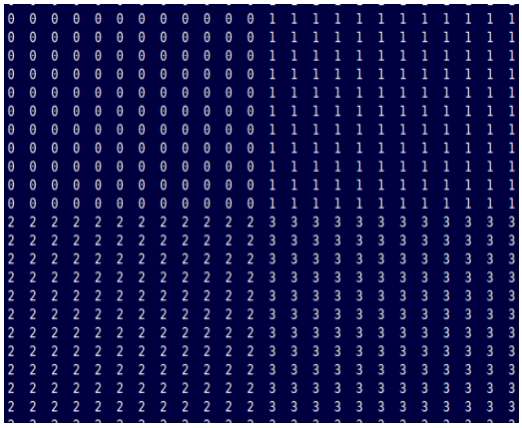


Execution results:

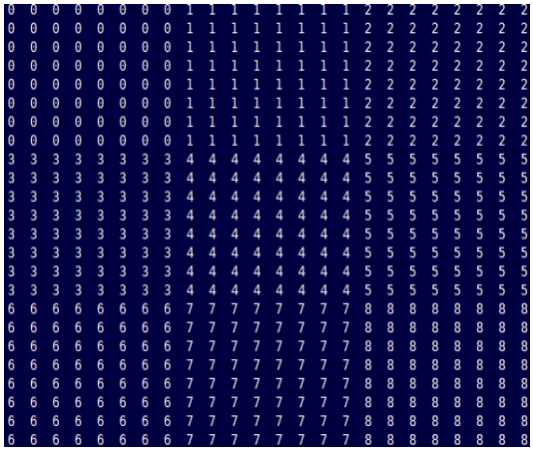
#processes = 1



#processes = 4



#processes = 9



#processes = 16