```
//
// Created by Marcus Rosti on 5/10/16.
//

#include "kNN-example.h"
#include "stdio.h"
#include "stdlib.h"
#include <mpi.h>
#include <time.h>

int main(int argc, char ** argv) {
    MPI_Init(&argc,&argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    if (argc < 4) {
        printf("Takes three arguments\n");
        printf("\tkNN-exmaple {n-dim} {m-dim} {seed}\n");
        exit(0);
    }
    const int n = atoi(argv[1]);
    const int m = atoi(argv[2]);
    const int seed = atoi(argv[3]);
    int k = 0;
    if(argc>4) k = atoi(argv[4]);


    srand((unsigned) seed);

    point_cloud **pc =  allocate_raster(n, m);
    initialize_raster(n, m, pc);

    //compute
    clock_t start = clock(), diff;
    compute_knn(n,m,pc,k);
    diff = clock() - start;
    int msec = (int) diff * 1000 / CLOCKS_PER_SEC;
    if(rank==0) printf("Time taken %d seconds %d milliseconds\n", msec/1000, msec%10
00);
    //compare
//    for(int i = 0; i < n; i++) {
//        printf("%5i,0 %2.5f\n",i,(double) pc[i][0].z);
//    }

    free(pc[0]);
    free(pc);

    MPI_Finalize();
    return 0;
}

void initialize_raster(const int n, const int m, point_cloud *const *pc) {
    int i, j;
    for(i = 0; i < n; i++) {
        for( j = 0; j < m; j ++) {
            pc[i][j].z = rand();
            /*pc[i][j].r = rand();
            pc[i][j].g = rand();
            pc[i][j].b = rand();
            pc[i][j].a = rand();*/

        }
    }
}

float **allocate_float_array(const int n, const int m) {
    float *data = (float *)malloc((unsigned long) n*m*sizeof(float));
    float **array= (float **)malloc((unsigned long) n*sizeof(float*));
    for (int i=0; i<n; i++)
        array[i] = &(data[m*i]);

    return array;
```

```
}

point_cloud **allocate_raster(const int n, const int m) {
    point_cloud *data = (point_cloud *)malloc((unsigned long) n*m*sizeof(point_cloud
));
    point_cloud **array= (point_cloud **)malloc((unsigned long) n*sizeof(point_cloud
*));
    for (int i=0; i<n; i++)
        array[i] = &(data[m*i]);

    return array;
}

/**
 *      m
 *   ******
 *   *    *
 *   *    *
 * n *    *
 *   *    *
 *   *    *
 *   ******
 */
void compute_knn(const int n, const int m, point_cloud *const *pc, const int k) {
    int rank;
    int nprocs;
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&nprocs);

    if(n % nprocs != 0) { printf("Invalid config"); exit(0); }

    const int num_rows = n/nprocs;
    //printf("Num Rows: %i\n",num_rows);

    float ** smoothed_pc = allocate_float_array(num_rows,m);

    for(int i = 0; i < num_rows; i++) {
        for(int j = 0; j < m; j++) {

            float val = 0;
            int num_cells = 0;
            for(int a = -k; a <= k; a++) {
                for(int b = -k; b <= k; b++) {
                    if(rank*num_rows+i+a >= 0 && rank*num_rows+i+a < n && j+b >= 0 &
& j+b < m) {
                        val += pc[rank * num_rows + i + a][j + b].z;
                        num_cells++;
                    }
                }
            }
            //printf("%i,%i:   %10.10f %10.10f %i\n",i,j,(val/num_cells),pc[rank*num
_rows+i][j].z,num_cells);
            smoothed_pc[i][j] = val/num_cells;
            //if(rank == 1) printf("%i,%i:   %10.10f %10.10f %i\n",i,j,smoothed_pc[i
][j],pc[rank*num_rows+i][j].z,num_cells);
        }
    }

    if(rank != 0) MPI_Send(&(smoothed_pc[0][0]),num_rows * m,MPI_FLOAT,0,0,MPI_COMM_
WORLD);

    if (rank == 0 ) {

        for(int sender = 0; sender < nprocs; sender++) {
            for(int i = 0; i <num_rows; i++) {
                for(int j = 0; j < m; j++) {
                    pc[sender*num_rows + i][j].z = smoothed_pc[i][j];
                }
            }
        }
        float ** rec_buff = allocate_float_array(num_rows,m);
        for(int sender = 1; sender < nprocs; sender++) {
            MPI_Status stat;
            //printf("Trying to rec\n");
            MPI_Recv(&(rec_buff[0][0]),num_rows*m,MPI_FLOAT,sender,0,MPI_COMM_WORLD,
&stat);
            //printf("Received\n");
            for(int i = 0; i <num_rows; i++) {
                for(int j = 0; j < m; j++) {
                    pc[sender*num_rows + i][j].z = rec_buff[i][j];
                }
            }
        }
        free(rec_buff[0]);
        free(rec_buff);
    }
    free(smoothed_pc[0]);
    free(smoothed_pc);

}
```