```c
#include "SDC-example.h"
#include "stdio.h"
#include "stdlib.h"
#include <mpi.h>
#include <time.h>

float **allocate_float_array(const int n, const int m);

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("\tSDC-example {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]);

    srand((unsigned) seed);

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

    //compute
    clock_t start = clock(), diff;
    compute_sdc(n,m,pc);
    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].x = i;
            pc[i][j].y = j;
            pc[i][j].z = ((float)rand()/(float)(RAND_MAX));
        }
    }
}

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*));
    int i;
    for (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
*));
```

```
    int i;
    for (i=0; i<n; i++)
        array[i] = &(data[m*i]);
    return array;
}

float S(float ** II, const int m, const int n, const int r, const int xmax, const in
t ymax) {
    int r_squared = r*r;
    int x1 = m + r;
    int y1 = n + r;
    int x2 = m - r;
    int y2 = n - r;

    if (x1 >= xmax || y1 >= ymax || x2 >= xmax || y2 >= ymax ||
      x1 < 0|| y1 < 0 || x2 < 0 || y2 < 0) {
      return 0;
    }
    return 0.25/r_squared * (II[x1][y1] - II[x2][y1] - II[x1][y2] + II[x2][y2]);
}

void compute_sdc(const int n, const int m, point_cloud *const *pc) {
    int rank;
    int nprocs;
    int i, j;
    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;
    const int start_row = rank * num_rows;

    point_cloud ** smoothed_pc = allocate_raster(num_rows,m);

    clock_t start = clock(), diff;

    //Generate Integral Image
    float **II = allocate_float_array(n, m);
    II[0][0] = pc[0][0].z;
    for (i = 1; i < m; i++) {
        II[0][i] = II[0][i-1] + pc[0][i].z;
    }

    for (i = 1; i < n; i++) {
        II[i][0] = II[i-1][0] + pc[i][0].z;
        for(j = 1; j < m; j++) {
            II[i][i] = II[i-1][j] - II[i-1][j-1] + II[i][j-1] + pc[0][i].z;
        }
    }

    //Create Final Smoothing Area Map
    float **FSAM = allocate_float_array(num_rows, m);
    for (i = 0; i < num_rows; i++) {
        for (j = 0; j < m; j++) {
            FSAM[i][j] = ((float)rand()/(float)(RAND_MAX)) * 5;
        }
    }

    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);

    //Cross Product
    for (i = 0; i < num_rows; i++) {
        int ii = start_row + i;
        for (j = 0; j < m; j++) {
            float r = FSAM[i][j];
            int r2 = (int) (r - 1);
            float v1 = S(II, ii+1, j, r2, n, m);
            float v2 = S(II, ii-1, j, r2, n, m);
            float h1 = S(II, ii, j+1, r2, n, m);
```

```
            float h2 = S(II, ii, j-1, r2, n, m);
            float z1 = 0.5 * (v1 - v2);
            float z2 = 0.5 * (h1 - h2);
            smoothed_pc[i][j].x = r * (z2 - z1);
            smoothed_pc[i][j].y = r * (z1 - z2);
            smoothed_pc[i][j].z = r*r;
        }
    }

    if(rank != 0) MPI_Send(&(smoothed_pc[0][0]), 3 * num_rows * m, MPI_FLOAT,0,0,MPI
_COMM_WORLD);

    int sender;
    if (rank == 0 ) {
        for(sender = 0; sender < nprocs; sender++) {
            for(i = 0; i <num_rows; i++) {
                for(j = 0; j < m; j++) {
                    pc[sender*num_rows + i][j].z = smoothed_pc[i][j].z;
                }
            }
        }
        point_cloud ** rec_buff = allocate_raster(num_rows,m);
        for(sender = 1; sender < nprocs; sender++) {
            MPI_Status stat;
            MPI_Recv(&(rec_buff[0][0]), 3*num_rows*m,MPI_FLOAT,sender,0,MPI_COMM_WOR
LD,&stat);
            for(i = 0; i <num_rows; i++) {
                for(j = 0; j < m; j++) {
                    pc[sender*num_rows + i][j].z = rec_buff[i][j].z;
                }
            }
        }
        free(rec_buff[0]);
        free(rec_buff);
    }
    free(smoothed_pc[0]);
    free(smoothed_pc);
}
```