Cmpe300
Analysis of Algorithms
Tunga Güngör

Student:
Canberk Yıldırım
2013400210

Submitted Person:
Burak Suyunu

MPI Programming Project
Image Denoising

26.12.2018

Introduction

In this project, i use parallel programming with C/C++ using MPI library. I implemented a parallel algorithm for image denoising with the Ising model using Metropolis-Hastings algorithm. The Ising model, named after the physicist Ernst Ising, is a mathematical model of ferromagnetism in statistical mechanics. The model consists of discrete variables that represent magnetic dipole moments of atomic spins that can be in one of two states (+1 or -1).

Program Interface

The project is written in the IDE Clion with the programming language C++. You can communicate with the program with the terminal.

To compile program you have to enter this command:

```
mpic++ -g main.cpp -o canberk
```

To execute it:

```
mpiexec -n 6 ./canberk lena200_noisy.txt output.txt 0.6 0.1
```

There are 4 parameters you need to enter:

1. lena200_noisy.txt :: the input noisy image file with ising model
2. output.txt :: the output file that your program will write the result in
3. 0.6 :: the β number
4. 0.1 :: the π number

Program Execution

To execute the program you should do the steps in the Programming Interface section. After those steps your program will generate an output.txt file which is the noisy-free version of your noisy image with the Ising Model. To change the txt file to a jpg file and see the noisy-free image you have to use this python

script with the command “python text_to_image.py output.txt aa.jpg”:

```
import sys
import numpy as np
import warnings
warnings.filterwarnings("ignore")

from matplotlib import pyplot as plt
import imageio

filepath = sys.argv[1]

img = np.loadtxt(filepath)
imageio.imwrite(sys.argv[2], img)

plt.imshow(img,cmap='gray',vmin=-1,vmax=1)
plt.show()
```

You have to save it a file with name “ text_to_image.py”. Then your image will be open.

Input and Output

As i described in the section Program Interface there are 4 parameters you need to enter:

1. lena200_noisy.txt :: the input noisy image file with ising model
2. output.txt :: the output file that your program will write the result in
3. 0.6 :: the β number
4. 0.1 :: the π number

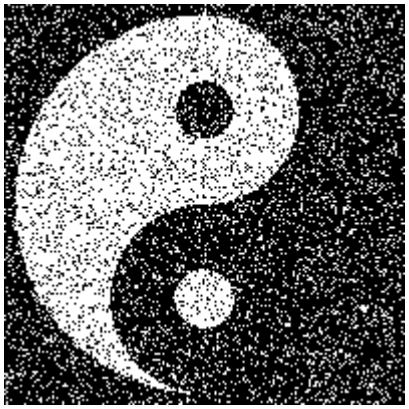
There are 2 files you need to give as parameters.
The input file is a txt file contains the 200x200 pixels noisy image text file with the Ising Model.
The output file should be a txt file.

Program Structure

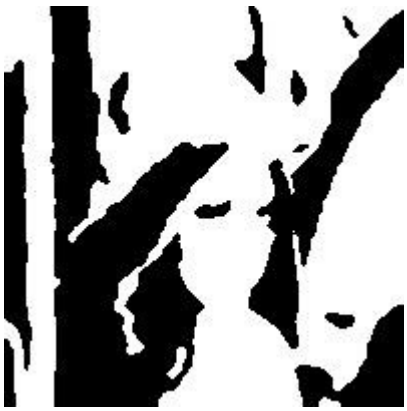
I will explain the program structure with comment in the source code which i will put in the Appendices section.

Examples

The input image of
lena200_noisy
and
yinyang



The output noisy-free
image of
lena
and
yinyang



Improvements and Extensions

The program needs some improvements about the running speed. It is slow some reasons i could not solve. And it needs some improvements about the denoising algorithm. It may delete some nonnoisy pixel by mistake and harms the originality of the image.

Difficulties Encountered

I had difficulties with the MPI environment installation and some other issues about the python modules that are not loaded in my computer.

Conclusion

With this project i learnt a lot of issues about the parallel programming and mpi environment. Also, it was helpful working with the image processing stuffs, because i did not have such an experience with such image stuffs.
I think i wrote the program well, it works fine. But it need some improvements like i mentioned in the improvement section.

Appendices

The Source Code

```
/*
Student Name: Canberk Yıldırım
Student Number: 2013400210
Compile Status: Compiling
Program Status: Working
Notes:
*/
#include <fstream>
#include <stdio.h>
#include <math.h>
#include <random>
#include "mpi.h"
using namespace std;
int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    string input = argv[1];
    string output = argv[2];
    double beta = atof(argv[3]);
    double pi = atof(argv[4]);
    double gamma = 0.5*log10((1-pi)/pi);
    int i,j;
    /*
     * The master processor's rank is 0
     */
    if(rank==0){
        ifstream file1(input);
        /*
         * The input array is declared
         */
        int** arr = NULL;
        arr = (int **)malloc(sizeof(int*)*200);
        for(i = 0 ; i < 200 ; i++)
            arr[i] = (int *)malloc(sizeof(int) * 200);
        /*
         * The output array is declared
         */
        int** arr2 = NULL;
        arr2 = (int **)malloc(sizeof(int*)*200);
```

```
        for(i = 0 ; i < 200 ; i++)
            arr2[i] = (int *)malloc(sizeof(int) * 200);
        /*
         * The input array is read from the input file
         */
        if (file1.good()) {
            int num;
            for(i=0;i<200;i++){
                for (j = 0; j < 200;j++) {
                    if(file1>>num) {
                        arr[i][j] = num;
                    }
                }
            }
        }
        file1.close();
        /*
         * The input array is send to the processors
         */
        for(i = 1 ; i <= size-1 ; i++) {
            for (j = 0; j < 200 / (size - 1); j++)
                MPI_Send(arr[200 / (size - 1) * (i - 1) + j], 200, MPI_INT, i,
j, MPI_COMM_WORLD);
        }
        /*
         * The output array is received from the processors
         */
        for(i = 1 ; i <= size-1 ; i++) {
            for (j = 0; j < 200 / (size - 1); j++)
                MPI_Recv(arr2[200 / (size - 1) * (i - 1) + j], 200, MPI_INT, i,
j, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        /*
         * The output array is written to the output file
         */
        ofstream file2("output.txt");
        string str;
        for(i=0;i<200;i++){
            for (j = 0; j < 200;j++) {
                str+=to_string(arr2[i][j]);
                str+=" ";
            }
            str+="\n";
        }
        if (file2.good()) {
            file2<<str;
        }
        file2.close();
    }
    /*
     * The other processors
     */
    else {
        /*
         * the original array X
         */
        int** X = NULL;
        X = (int **)malloc(sizeof(int*) * 200/(size-1));
        for(i = 0 ; i < 200/(size-1) ; i++)
            X[i] = (int *)malloc(sizeof(int) * 200);
        /*
         * X is filled with the data received from the master processor
```

```
         */
        for(i = 0 ; i < 200/(size-1) ; i++) {
            MPI_Recv(X[i], 200, MPI_INT, 0, i, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }
        /*
         * Z is declared and copied from X
         */
        int** Z = NULL;
        Z = (int **)malloc(sizeof(int*) * 200/(size-1));
        for(i = 0 ; i < 200/(size-1) ; i++) {
            Z[i] = (int *) malloc(sizeof(int) * 200);
            Z[i] = X[i];
        }
        /*
         * Array A is the last line of upper neighbor
         */
        int** A = NULL;
        A = (int **)malloc(sizeof(int*) * 1);
        A[0] = (int *)malloc(sizeof(int) * 200);
        /*
         * Array B is the first line of lower neighbor
         */
        int** B = NULL;
        B = (int **)malloc(sizeof(int*) * 1);
        B[0] = (int *)malloc(sizeof(int) * 200);
        /*
         * The Image Denoising part with iterations
         */
        for(int k=0;k<200*12*200/(size-1);k++) {
            /*
             * random i coordinates
             */
            i = rand() % (200/(size-1));
            /*
             * random j coordinates
             */
            j = rand() % 200;
            /*
             * The searching for the neighbor pixels
             */
            int sum=0;
            for(int k = max((i-1),0) ; k<=min((i+1),200/(size-1)-1) ;k++){
                for (int l = max((j-1),0) ; l<=min((j+1),199) ;l++){
                    sum = sum + Z[k][l];
                }
            }
            /*
             * This part of the program let the processors communicate with each
other so that they
             * can get the information about the neighbor cells
             */
            if(rank==1){
                /*
                 * The top processor sends its last row to its lower neighbor
                 * and it receives the first row of its lower neighbor
                 */
                MPI_Send(Z[200/(size-1)-1], 200, MPI_INT, 2, 0, MPI_COMM_WORLD);
                MPI_Recv(B[0], 200, MPI_INT, 2, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                /*
```

```
                 * If the random row (i) is the last of the array, lower
neighbor cells are added to sum
                 */
                if(i==200/(size-1)-1){
                    sum+=B[0][j-1]+B[0][j]+B[0][j+1];
                }
            }else if(rank==size-1){
                /*
                 * The bottom processor sends its first row to its upper
neighbor
                 * and it receives the last row of its upper neighbor
                 */
                MPI_Send(Z[0], 200, MPI_INT, rank-1, 0, MPI_COMM_WORLD);
                MPI_Recv(A[0], 200, MPI_INT, rank-1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                /*
                 * If the random row (i) is the first of the array, upper
neighbor cells are added to sum
                 */
                if(i==0){
                    sum+=A[0][j-1]+A[0][j]+A[0][j+1];
                }
            }else{
                /*
                 * The other processors send their first row to their upper
neighbor
                 * and they receive the last row of their upper neighbor
                 */
                MPI_Send(Z[0], 200, MPI_INT, rank-1, 0, MPI_COMM_WORLD);
                MPI_Recv(A[0], 200, MPI_INT, rank-1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                /*
                 * The other processors send their last row to their lower
neighbor
                 * and they receive the first row of their lower neighbor
                 */
                MPI_Send(Z[200/(size-1)-1], 200, MPI_INT, rank+1, 0,
MPI_COMM_WORLD);
                MPI_Recv(B[0], 200, MPI_INT, rank+1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
                /*
                 * If the random row (i) is the last of the array lower neighbor
cells are added to sum
                 */
                if(i==0){
                    sum+=A[0][j-1]+A[0][j]+A[0][j+1];
                }
                /*
                 * If the random row (i) is the last of the array, lower
neighbor cells are added to sum
                 */
                else if(i==200/(size-1)-1){
                    sum+=B[0][j-1]+B[0][j]+B[0][j+1];
                }
            }
            /*
             * the acceptance probability
             */
            double delta_E = -2 * gamma * (X[i][j] * Z[i][j]) - 2 * beta * Z[i]
[j] * (sum - Z[i][j]);
            /*
             * changing pixels with the probability
```

```
             */
            double rand_number = (double) rand() / RAND_MAX;
            if (log10(rand_number) < delta_E){
                Z[i][j] = - Z[i][j];
            }
        }
        /*
         * Sending the last version of the arrays
         */
        for(i = 0 ; i < 200/(size-1) ; i++) {
            MPI_Send(Z[i], 200, MPI_INT, 0, i, MPI_COMM_WORLD);
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}
```