INTRODUCTION

The problem is that there is an image Z. But, I just have an image X which is noised by factor of pi(ie. 0.1). I try to reach the original image by using my program and guessing the efficient constants to enter(beta and pi).

PROGRAM INTERFACE

I create my program in Linux system. To run the program user must give some command line arguments before execution. And also you should have been installed openmpi.

argv[1] : the input file name (ie. input.txt, image.txt)

argv[2] : the output file name (ie. output.txt, outimage.txt)

argv[3] : double type of beta constant (ie. between 0 and 1)

argv[4] : double type of pi constant (ie. between 0 and 1)

And also you should enter the processor num which is dividible by size. Finally, you should enter something like these two commands:

```
>> mpicc -g imagedenoise.c -o imagedenoise
>> mpiexec -n 11 ./imagedenoise input.txt output.txt 0.4 0.15
  ( mpiexec -n [NUM_OF_PROCESSOR + 1] ./imagedenoise argv[1] argv[2] argv[3] argv[4] )
```

After you get the output file, you can convert it to the image.

PROGRAM EXECUTION

What I use in terms of achieve our goal:

>> I write this program in C programming language.

>> For process communication, I use openmpi.

>> The input file is formed by Ising Model (just 1's and -1's)

>> I use Metropolis Hastings and acceptance probability to decide the flips

>> The output file is also formed in Ising model.

So, if you want to see the output as an image you must use “text_to_image.py” script.

INPUT AND OUTPUT

For example, we have an input file like this:

1 -1 1 -1 1

-1 1 -1 1 -1

1 1 -1 -1 1

-1 -1 1 1 -1

After program is executed by some constants and adequate iteration we have an output file like this:

1 1 1 -1 -1

-1 -1 1 1 -1

-1 1 -1 -1 1

1 -1 1 -1 -1

This output will be similar to orinal image file. You can convert it to the image.

PROGRAM STRUCTURE

I will explain the program block by block. First, We have constants’ definitions. Then, we are in main part. I take the command line arguments. Then I started MPI. I get the process number and partition them one master and slaves.

>> Master Work

+ I created 2D array and got the input file in it.

+ Then, send tasks to slave processes.

+ For syncronization, I use Mpi_Barrier.

+ And, master waits output from slaves processes.

+ When he get, then write the output to the file.

>> Slave Work (for each slave processes)

+ Here, we have parallel work and message passing.

+ I create 2D array and receive the input from master process.

+ And, before calculation each process sends their probably-shared data to the others.

+ For syncronization, I again use Mpi_Barrier.

+ Then, the calculation to decide whether flip the pixel or not.

+ I calculate the acceptance probability of each pixel.

+ And, I decide if probability higher than limit then I flip else not flip.

+ After adequate iteration (for 200x200 picture total 500.000 iteration) each slave process send their part to the master.

>> Finally all processes are done and terminate(Mpi_Finalize).

EXAMPLES

We are trying to get similar image to the original one. We couldn't get exactly the original one because we are moving with probabilistic approach and randomness. Also, while we don't change the constants, we could have different output in each execution.