



# PARALLEL IMAGE DENOISING

CMPE300 – ANALYSIS OF ALGORITHMS

Programming Project

Due: 26.12.2018

Fatih iver

Fatih.iver@boun.edu.tr

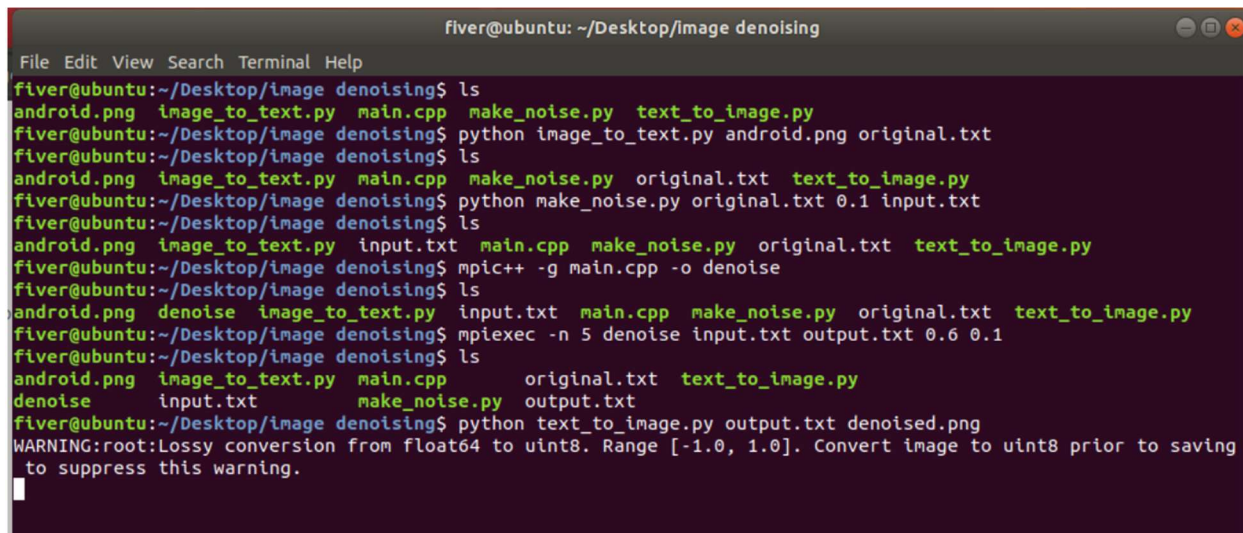
## INTRODUCTION

This project aims to use parallel programming to denoise a given noisy picture. It takes advantage of randomizations. The idea comes from the Ising model. We assume that given noisy picture can be denoised by using the fundamental idea in Ising model. Since Ising model requires two states our algorithms, only work for black & white pictures. Another leverage we take is to use Metropolis-Hastings algorithms. Since we use randomization and want to converge to a result, we need a decision making mechanism while guessing whether to flip the bit.

## PROGRAM INTERFACE & EXECUTION

The explanation and screenshot below show how to compile and run the program:

1. Initially, you have main.cpp and three python scripts and an image (original).
2. User converts original image to text a file representation by using image\_to\_text.py.
3. User add noise to original text representation of the image by using make\_noise.py.
4. User compile the main.cpp by using parallel programming environment.
5. Then, runs the compiled program with inputs explained below to get denoised image.
6. User converts obtained text file to an image format by using text\_to\_image.py



```
fiver@ubuntu: ~/Desktop/image denoising
File Edit View Search Terminal Help
fiver@ubuntu:~/Desktop/image denoising$ ls
android.png image_to_text.py main.cpp make_noise.py text_to_image.py
fiver@ubuntu:~/Desktop/image denoising$ python image_to_text.py android.png original.txt
fiver@ubuntu:~/Desktop/image denoising$ ls
android.png image_to_text.py main.cpp make_noise.py original.txt text_to_image.py
fiver@ubuntu:~/Desktop/image denoising$ python make_noise.py original.txt 0.1 input.txt
fiver@ubuntu:~/Desktop/image denoising$ ls
android.png image_to_text.py input.txt main.cpp make_noise.py original.txt text_to_image.py
fiver@ubuntu:~/Desktop/image denoising$ mpic++ -g main.cpp -o denoise
fiver@ubuntu:~/Desktop/image denoising$ ls
android.png denoise image_to_text.py input.txt main.cpp make_noise.py original.txt text_to_image.py
fiver@ubuntu:~/Desktop/image denoising$ mpiexec -n 5 denoise input.txt output.txt 0.6 0.1
fiver@ubuntu:~/Desktop/image denoising$ ls
android.png image_to_text.py main.cpp original.txt text_to_image.py
denoise input.txt make_noise.py output.txt
fiver@ubuntu:~/Desktop/image denoising$ python text_to_image.py output.txt denoised.png
WARNING:root:Lossy conversion from float64 to uint8. Range [-1.0, 1.0]. Convert image to uint8 prior to saving
to suppress this warning.
```

\*If users have the noisy black & white image already, and if they have the compiled program they only need to do step 5 and 6.

## INPUT & OUTPUT

### Input File

The input file is a two-dimensional array representation of a black and white noisy image. For a 200px\*200px image, input file includes 200 rows and columns. Black is represented by 1 and white is represented by 0. At one location only one color can be found which are 1 and 0.

### Output File

The output file is a two-dimensional array representation of a black and white denoised image. It has same characteristics with the input file.

### Beta Value

This value represents the consistency between neighboring cells. If beta is higher, we expect more consistency between neighboring cells.

### Pi Value

This value represents the noise probability. It is the measure of how noisy given picture is. So higher pi value implies higher noise for the given noisy picture.

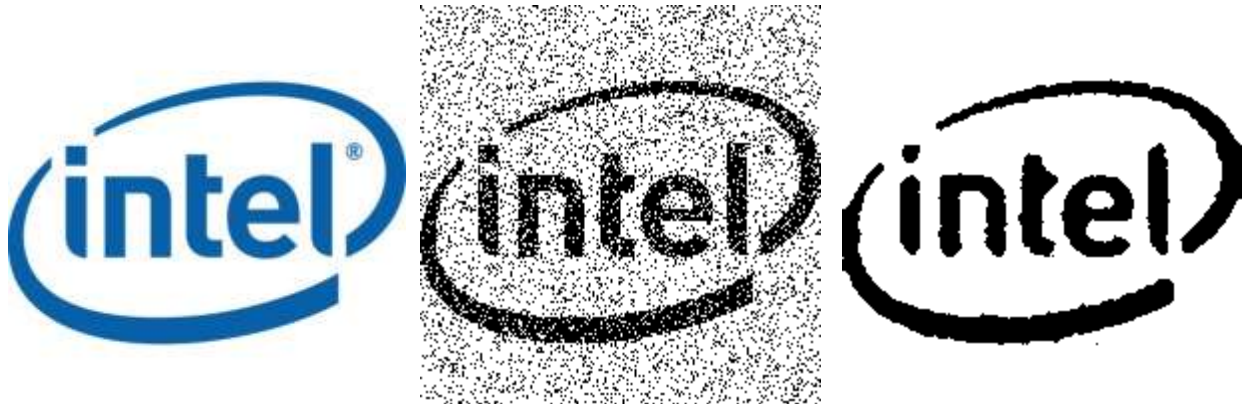
## EXAMPLES

I share two examples to show what exactly this program does. (original, noisy, denoised)

### Example: Android



## Example: Intel



### IMPROVEMENT & EXTENSIONS

The part of the code that is related with the exchange of the rows could be improved a lot. Since I have very limited time those days, I coded in a way that comes to mind first. It is like a “brute force” algorithm. You were not able to figure out a smarter way, so you code it in a less smart way just to see it really works. Another improvement can be done on distributing data. Since I find it easier to send a 2D array piece by piece rather than sending it as a one block, it causes lots of overhead. Improving those two, makes it a lot faster.

### DIFFICULTIES ENCOUNTERED

This was my first time to code for parallel programming. Even I have never used threads before. No multi programming. Since I used to code in a sequential manner it was hard to think in a parallel manner. Especially I had problem with the part that you just write one source code. However, this source codes supposed to run on all processors. You need divide your code into pieces so that you can assign divided parts to your processors. Not all of them need to execute a part. Another problem was the synchronization issues and deadlocks. However, thanks to this project, it is much clearer for me and I have now at least a sense of feeling for how to program in a parallel manner. I have realized that parallel programming is fun!

## CONCLUSIONS

I have reached several conclusions with the help of this project, I want to share them.

- Randomization may help you to solve some serious problems like image denoising. I guess, it would be much hard to solve this problem without using randomization.
- Parallel programming may shorten running time for a program written for non-parallel manner. However, this is not the case all the time. When a program written in a parallel manner, the program runs on several processors simultaneously. This brings communication overhead which may make your program to slow down, a lot. Need to think wise, before converting a non-parallel program to a parallel program. It may not worth to your efforts.
- You need to change the way of your thinking to be able code in parallel manner. If you have never code for parallel manner, don't panic. It may take time your brain to adapt this new concept. I promise, it is fun after you grasp the basics.

## REFERENCES

**Parallel Programming:** Check this detailed article: <https://bit.ly/1JqN9yD>

This article includes lots of example for parallel programming with message passing interface.

**Monte Carlo Method:** This project is only one small application of Monte Carlo method.

Check this website to see its usage on finance: <https://bit.ly/2V8slk6>

**Message Passing Interface:** This website for tutorials: <https://bit.ly/2diBhHg>

If you want to have a quick start on parallel programming, I strongly suggest you to check it.

**Markov Chain Monte Carlo:** Check this GitHub repository: <https://bit.ly/2AghwJn>

This repository includes several examples of application of this method on image denoising.

## APPENDICES

### Source Code

```
1. /*
2. Student Name: Fatih İver
3. Student Number: 2016400264
4. Compile Status: Compiling
```

```
5. Program Status: Working
6. Notes: I was very confused in the beginning, but now it is so clear. Thanks for this project.
7. */
8.
9. #include <iostream>
10. #include <fstream>
11. #include <string>
12. #include <sstream>
13. #include <iostream>
14. #include <cstring>
15. #include <cstdlib>
16. #include <math.h>
17. #include <stdlib.h>
18. #include <time.h>
19. #include <mpi.h>
20.
21. using namespace std;
22.
23. int main(int argc, char** argv)
24. {
25.
26.     MPI_Init(NULL, NULL);
27.
28.     // Get the number of processes
29.     int world_size;
30.     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
31.
32.     // Get the rank of the process
33.     int world_rank;
34.     MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
35.
36.     int N = 200;
37.     int S = world_size-1;
38.     int R = N/S;
39.     int C = N;
40.
41.     if(world_rank == 0){
42.
43.         // ----- PREPARE AN ARRAY TO BE FILLED -----
44.
45.         int** X = new int*[N];
46.         for(int i = 0; i < N; i++) { X[i] = new int[N]; }
47.
48.
49.         // ----- READ INPUT FILE AND FILL THE PREPARED ARRAY -----
50.
51.         ifstream input_file(argv[1]);
52.
53.         if(input_file.is_open()){
54.
55.             int row = 0;
56.
57.             string line;
58.
59.             while(getline(input_file, line)){
60.
61.                 int column = 0;
62.
63.                 string element;
64.
```

```

65.         istream iss(line);
66.
67.         while(iss >> element) { X[row][column++] = atoi(element.c_str()); }
68.
69.         row++;
70.     }
71.
72.     input_file.close();
73.
74.     } else {
75.         throw "Unable to Open Input File!";
76.     }
77.
78.     // ----- DISTRIBUTE DATA - TO SLAVES -----
79.
80.     for(int s = 0; s < S; s++)
81.         for(int r = 0; r < R; r++)
82.             MPI_Send(X[s*R + r], N, MPI_INT, s+1, r, MPI_COMM_WORLD);
83.
84.     // -----
85.
86.     } else {
87.
88.     // ----- COLLECT DATA - FROM MASTER -----
89.
90.         int X[R][N];
91.
92.         for(int r = 0; r < R; r++)
93.             MPI_Recv(X[r], N, MPI_INT, 0, r, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
94.
95.         // ----- DENOISE - INITIALIZE VALUES -----
96.
97.         int Z[R][N];
98.
99.         for(int r = 0; r < R; r++)
100.             for(int c = 0; c < N; c++)
101.                 Z[r][c] = X[r][c];
102.
103.         int EXTRA_ROW_ABOVE[N];
104.         int EXTRA_ROW_BELOW[N];
105.
106.         double beta = atof(argv[3]);
107.
108.         double pi = atof(argv[4]);
109.         double gamma = 0.5*log((1-pi)/pi);
110.
111.         long iteration_limit = 250000 / S;
112.
113.         srand(time(NULL));
114.
115.         for(long iteration_number = 1; iteration_number < iteration_limit; iteration_number++) {
116.
117.             // ----- DISTRIBUTE LAST ROWS -----
118.
119.             if (world_rank != 1)
120.                 MPI_Recv(EXTRA_ROW_ABOVE, N, MPI_INT, world_rank - 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
121.
122.             MPI_Send(Z[R-
1], N, MPI_INT, world_rank % S + 1, 1, MPI_COMM_WORLD);

```

```

123.
124.         if (world_rank == 1)
125.             MPI_Recv(EXTRA_ROW_ABOVE, N, MPI_INT, S, 1, MPI_COMM_WORLD, MPI_
STATUS_IGNORE);
126.
127.         // ----- DISTRIBUTE FIRST ROWS -----
128.
129.         if (world_rank != S)
130.             MPI_Recv(EXTRA_ROW_BELOW, N, MPI_INT, world_rank + 1, 2, MPI_COM
M_WORLD, MPI_STATUS_IGNORE);
131.
132.             MPI_Send(Z[0], N, MPI_INT, (world_rank - 2 + S) % S + 1, 2, MPI_COMM
_WORLD);
133.
134.         if (world_rank == S)
135.             MPI_Recv(EXTRA_ROW_BELOW, N, MPI_INT, 1, 2, MPI_COMM_WORLD, MPI_
STATUS_IGNORE);
136.
137.         // ----- DENOISE - PROPOSE BIT FLIP -----
138.
139.         int i = rand() % R;
140.         int j = rand() % N;
141.
142.         int neighbour_sum = 0;
143.
144.         for(int k = -1 ; k <= 1; k++) {
145.
146.             for(int l = -1; l <= 1; l++) {
147.
148.                 int n_i = i + k;
149.                 int n_j = j + l;
150.
151.                 if(n_j > -1 && n_j < N) {
152.
153.                     if(n_i == -1) {
154.                         neighbour_sum += EXTRA_ROW_ABOVE[n_j];
155.                     } else if(n_i == R){
156.                         neighbour_sum += EXTRA_ROW_BELOW[n_j];
157.                     } else {
158.                         neighbour_sum += Z[n_i][n_j];
159.                     }
160.
161.                 }
162.
163.             }
164.
165.         }
166.
167.         // ----- DENOISE - CALCULATE PROBABILITY -----
168.
169.         double delta_E = -
2 * gamma * Z[i][j] * X[i][j] - 2 * beta * Z[i][j] * neighbour_sum;
170.
171.         if ( ((double) rand() / (RAND_MAX)) < exp(delta_E) ) { Z[i][j] = -
Z[i][j]; }
172.
173.         // -----
174.     }
175.
176.     // ----- DISTRIBUTE DATA - TO MASTER -----
177.

```



```

178.         for(int r = 0; r < R; r++)
179.             MPI_Send(Z[r], N, MPI_INT, 0, (world_rank-
180. 1)*R + r, MPI_COMM_WORLD);
181.         // -----
182.     }
183.
184.     MPI_Barrier(MPI_COMM_WORLD);
185.
186.     if(world_rank == 0) {
187.         // ----- COLLECT DATA - FROM SLAVES -----
188.
189.         int Z[N][N];
190.
191.         for(int s = 0; s < S; s++)
192.             for(int r = 0; r < R; r++)
193.                 MPI_Recv(Z[s*R + r], N, MPI_INT, s+1, s*R + r, MPI_COMM_WORLD, M
194. PI_STATUS_IGNORE);
195.
196.         // ----- WRITE TO OUTPUT FILE -----
197.
198.         ofstream output_file(argv[2]);
199.
200.         if (output_file.is_open()){
201.             for(int row = 0; row < N; row++) {
202.
203.                 for(int column = 0; column < N; column++) {
204.                     output_file << Z[row][column] << " ";
205.                 }
206.
207.                 output_file << "\n";
208.             }
209.
210.             output_file.close();
211.
212.             } else {
213.                 throw "Unable to Open Output File!";
214.             }
215.
216.         // -----
217.     }
218.
219.     // Finalize the MPI environment. No more MPI calls can be made after this
220.
221.     MPI_Barrier(MPI_COMM_WORLD);
222.
223.     MPI_Finalize();
224.
225.     return 0;
226. }

```

## Image to Text – Python Script

```

1. import sys
2. from PIL import Image

```

```

3. import numpy as np
4.
5. filepath = sys.argv[1]
6.
7. imgX = Image.open(filepath)
8. imgX = imgX.convert('L')
9. img = np.asarray(imgX)
10. img = 2*(img > 128).astype(int)-1
11.
12. file = open(sys.argv[2], "w")
13. txt = list(img)
14. for i in txt:
15.     for j in i:
16.         file.write(str(j) + " ")
17.     file.write('\n')
18. file.close()

```

## Make Noise – Python Script

```

1. import sys
2. import numpy as np
3.
4. img = np.loadtxt(sys.argv[1])
5.
6. pi = float(sys.argv[2])
7. I,J = img.shape
8. flip = np.random.rand(I,J) < pi
9. X = img * (-1)**flip
10.
11. file = open(sys.argv[3], "w")
12. txt = list(X)
13. for i in txt:
14.     for j in i:
15.         file.write(str(int(j)) + " ")
16.     file.write('\n')
17. file.close()

```

## Text to Image – Python Script

```

1. import sys
2. import numpy as np
3. import warnings
4. warnings.filterwarnings("ignore")
5.
6. from matplotlib import pyplot as plt
7. import imageio
8.
9. filepath = sys.argv[1]
10.
11. img = np.loadtxt(filepath)
12. imageio.imwrite(sys.argv[2], img)
13.
14. plt.imshow(img, cmap='gray', vmin=-1, vmax=1)
15. plt.show()

```