CMPE 300

Analysis of Algorithms

Ekrem Öztürk

Programming Project

27.12.2016

Introduction

In this project, I have implemented a simple smoothing and line detecting algorithm for 200x200 grayscale images. For a given text file with 200 rows and 200 inputs on each row, the program first does smoothing and demeans into 198 rows with 198 inputs each, and then does line detecting and demeans into 196 rows with 196 inputs on each row. The input file contains rgb value of each pixel of the image.

Smoothing: Consider an (n x n) matrix. Smoothing is done by taking mean of each 3x3 matrix by designating a center element from row 2 to n-1 and column 2 to n-1.

Line Detecting: Consider an (n x n) matrix. Line detecting is done by multiplying each 3x3 matrix by designating a center element from row 2 to n-1 and column 2 to n-1 by 4 different 3x3 matrices to detect horizontal, vertical and oblique lines.

After these steps, the input is subjected to a threshold to set an output with pure black and pure white pixels.

The project was implemented in C language with use of MPI (Message Passing Interface) libraries.

Program Interface

The executable file's name is 'linedetector'. It can be executed running the command below in its directory:

mpiexec -n <#ofprocessors> ./linedetector <inputfile> <outputfile> <threshold>

where <#ofprocessors> can be any number which divides 200 without remainder plus 1 for master processor and threshold can be any number between 0 and 255 but for reasonable results 10, 25 and 40 are recommended. One can give any name (using same name more than once will overwrite the existing file) to output file.

The program terminates itself after a while (approximately in 2-3 seconds depending on the machine). If the user wants to terminate earlier than that, (s)he can press ctrl+c (or kntrl+c in mac).

Program Execution

The input should have rgb values of a grayscale image. Each pixel of a 200x200 image should be converted into rgb value and be in form of 200 rows and 200 columns. The program takes the input, divides it into all servant processors, runs smoothing and line detection algorithm using given threshold.

After execution, the program creates an output or overwrites existing output file. This file have 196 rows and columns consists of 0's and 255's which is black or white.

The program for conversion is given: visulize.py. It also converts its input into an image, which we will use this functionality to convert out output file to an image.

Input and Output

I have explained input and output files in above section. In addition to that, there should be one character space between values in input. Output is also produced in this format.

Program Structure

int rank: Rank of the processor (from 0 to n-1 for n processors)

int size: Number of processors (n)

int transNumber: Variable used to transfer values in mpi_send and mpi_receive functions.

int transSize: Transferred array size: 200 / (# of servant processors)

int threshold: Threshold variable.

After above values (excluding transNumber) is determined, the master processor starts reading the input file.

int input[200][200]: 2D input array

int output[196][196]: 2D output array

In a nested for loop input file is read, then divided with number of servant processors and sent to them. After servants are finished running the algorithms, processed values are received from them. In this case, first and last servant send 48x196 array, the middle ones send 50x196 array. Then all concatenated output is written into output file.

int transInput[ ][ ] : 2D input array for processor

All servant processors receives transferred input from master and inserts it into transInput. Its size is transSize+1 for first and last, and transSize+2 for processors in the middle. It is because to open a line for border values coming from neighbour rows. Then border values are sent and received by each processor. After that smoothing starts.

int smoothedInput[ ][ ]: 2D array for inserting smoothed values

int smoothedNumber: Variable for produced number after taking mean.

The 3x3 matrices are summed and divided by 9 to take mean. Then the number is inserted into array and set to zero afterwards to be ready for next calculation.

Again border lines are sent and received, this time from smoothed input.

int thresholdedMatrix[ ][ ]: 2D array for inserting thresholded values

int horizontalLine, verticalLine, obliquePlus45, obliqueMinus45: Variables for produced numbers after multiplications.

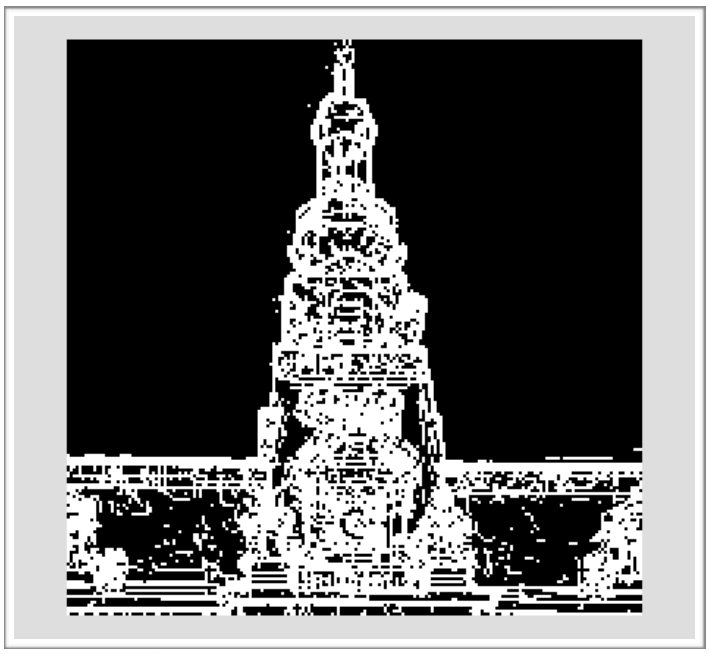
int isLine: Variable for checking if one of the above variables has passed the threshold.

All servant processors multiplies its 3x3 elements with designated 4 matrices to detect lines. Then these four results compared with threshold. If there is an above value the pixel is set to 255, otherwise 0 and inserted to thresholdedMatrix. The results set to zero afterwards for next calculation.

Last but not least, every servant sends values inside their thresholdedMatrix to master processor. Then master processor concatenates these values and writes into the output file.

Examples

This is the grayscale of original image. The image consists 200x200 = 40000 pixels. Its pixel's color values turned into text with visulize.py program. The values are in input.txt.



This is output image of output10.txt. This is program output with threshold 10.

This is output image of output25.txt. This is program output with threshold 25.

This is output image of output40.txt. This is program output with threshold 40.

Improvements and Extensions

Some of the values between borders comes out wrong. I could not find what causes this bug. I have also tried my own pictures. The program did pretty good. However a picture with fewer lines (for example a self-portraiture, face has few lines) cannot be converted good. I think its because of image's being 200x200 pixels. If I had time, I would improve the program to take every size of input. This would gave flexibility to user to try the program on better quality images.

Difficulties Encountered

Parallel programming was a new concept to me. I had to learn MPI's features and functions first. Besides, C is a language that I am not very familiar with (I know C++, but this is my first project in C).

While passing border values between processors, processors entered deadlock. I was using MPI_Send and MPI_Recv functions. I have tried to change their order, give priority to send etc. Then I have found MPI_sendrecv function to solve deadlock problem and hopefully it did.

Conclusion

In conlusion, this project helped me to learn more about parallel programming and MPI library. I have learned line detecting in a simple way and I had enjoyed while implementing the algorithm.

Appendices

```
/* Student Name: Ekrem Ozturk
 * Student Number: 2012400006
 * Compile Status: Compiling
 * Program Status: Working
 */

#include <stdio.h>
#include "mpi.h"
```

```
int main(int argc, char* argv[])
{

        int rank, size;


  MPI_Init(&argc, &argv);  //initialize MPI
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);   //Rank of each processor initialized from 0 to n-1
  MPI_Comm_size(MPI_COMM_WORLD, &size);  //Total number of processors initialized

  int thresholdArgument[1];  //

  sscanf(argv[3], "%d", &thresholdArgument[0]) ;

    int transNumber = 0;   //Variable used to pass values in mpi_send and mpi_receive functions

  int transSize = 200/(size-1);  //Transferred array size: 200 / (# of servant processors)

  int threshold = thresholdArgument[0];  //Threshold variable

  if(rank == 0){ //Master processor

        FILE *File= fopen(argv[1], "r");
                int input[200][200];  //input 2D array
    int output[196][196];  //output 2D array

    //reads the input file and inserts into array
                for(int row=0; row<200; row++){
                for( int col=0; col<200; col++){
                        int s[1];
                        fscanf(File, "%d" ,&s[0]);
                        input[row][col]= s[0];
```

```
                }
        }

    //Divides the input into equal parts and sends them to servants
        for(int proc = 1; proc < size; proc++){
                for(int row = 0; row <transSize; row++){
                        for(int column = 0; column < 200; column++){
                                int transNumber = input[row+(transSize*(proc-1))]
[column];
                                MPI_Send(&transNumber, 1, MPI_INT, proc, 0,
MPI_COMM_WORLD);
                                //printf("Process %d received input %d from process
0\n", proc, transNumber);
                        }
                }
            }

    //Receives output and inserts into output array
    for(int proc = 1; proc<size; proc++){
      if(proc == 1){
        for(int row = 0; row <transSize-2; row++){
          for(int column = 0; column < 196; column++){
                MPI_Recv(&transNumber, 1, MPI_INT, proc, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            output[row][column] = transNumber;
            //printf("Process 0 received input %d from process %d\n", transNumber, proc);
          }
        }
      } else if(proc == size-1){
        for(int row = 0; row <transSize-2; row++){
          for(int column = 0; column < 196; column++){
                MPI_Recv(&transNumber, 1, MPI_INT, proc, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            output[row+(transSize*(proc-1)-2)][column] = transNumber;
          }
```

```
        }
      } else {
        for(int row = 0; row <transSize; row++){
          for(int column = 0; column < 196; column++){
                    MPI_Recv(&transNumber, 1, MPI_INT, proc, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            output[row+(transSize-2)+(transSize*(proc-2))][column] = transNumber;
          }
        }
      }
    }


    FILE *outputFile = fopen(argv[2], "w");

    //Writes output into output file
    for(int row=0; row<196; row++){
      for( int col=0; col<196; col++){
        int outputNumber = output[row][col];
        fprintf(outputFile, "%d ", outputNumber);
      }
      fprintf(outputFile, "\n");
    }


  }

  for(int proc = 1; proc < size; proc++){  //Servant processors
       if(rank == proc){

            if(rank == 1){ //processor 2

                 int transInput[transSize+1][200];  //input array for this processor

       //receive input from master
```

```
                    for(int row = 0; row <transSize; row++){
                            for(int column = 0; column < 200; column++){
                                    MPI_Recv(&transNumber, 1, MPI_INT, 0, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                                    transInput[row][column] = transNumber;
                                    //printf("Process %d received input %d from
process 0\n", proc, transInput[row][column]);
                            }
                    }

        //Send and receive border line
                            for(int column = 0; column < 200; column++){

          int transNumber = transInput[transSize-1][column];
                    MPI_Sendrecv(&transNumber, 1, MPI_INT, proc+1, 0, &transNumber, 1,
MPI_INT, proc+1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                            transInput[transSize][column] = transNumber;

                    }

        //SMOOTHING

        int smoothedInput[transSize][198];
        int smoothedNumber = 0;  //Variable for produced number after taking mean

        //Takes mean of 3x3 matrix and inserts into smoothedInput array
        for(int row = 1; row < transSize; row ++){
          for(int column = 1; column < 199; column++){
            for(int i = -1; i < 2; i++){
              for(int j = -1; j < 2; j++){
                smoothedNumber = smoothedNumber + transInput[row+i][column+j];
              }
            }
            smoothedNumber = smoothedNumber/9;
            smoothedInput[row-1][column-1] = smoothedNumber;
```

```
            smoothedNumber = 0;
          }
        }

        //Send and receive border line
        for(int column = 0; column < 198; column++){

          int smoothedNumber = smoothedInput[transSize-2][column];
          //printf("1. sayı : %d\n", smoothedNumber);
            MPI_Sendrecv(&smoothedNumber, 1, MPI_INT, proc+1, 0, &smoothedNumber,
1, MPI_INT, proc+1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
          //printf("2. sayı : %d\n", smoothedNumber);
          smoothedInput[transSize-1][column] = smoothedNumber;

        }



        int thresholdedMatrix[transSize-2][196];

        //Variables for produced numbers after multiplications
        int horizontalLine = 0;
        int verticalLine = 0;
        int obliquePlus45 = 0;
        int obliqueMinus45 = 0;

            int isLine = 0;   // if it is greater than zero, one of above values has passed the
threshold



        //Multiplies 3x3 matrix and inserts 0 or 255 into thresholdedMatrix array
        for(int row = 1; row < transSize-1; row++){
          for(int column = 1; column<197; column++){
            horizontalLine =
            -1*smoothedInput[row-1][column-1]
            +2*smoothedInput[row-1][column]
```

```
-1*smoothedInput[row-1][column+1]
-1*smoothedInput[row][column-1]
+2*smoothedInput[row][column]
-1*smoothedInput[row][column+1]
-1*smoothedInput[row+1][column-1]
+2*smoothedInput[row+1][column]
-1*smoothedInput[row+1][column+1];

if(horizontalLine>threshold) isLine++;

verticalLine =
-1*smoothedInput[row-1][column-1]
-1*smoothedInput[row-1][column]
-1*smoothedInput[row-1][column+1]
+2*smoothedInput[row][column-1]
+2*smoothedInput[row][column]
+2*smoothedInput[row][column+1]
-1*smoothedInput[row+1][column-1]
-1*smoothedInput[row+1][column]
-1*smoothedInput[row+1][column+1];

if(verticalLine>threshold) isLine++;

obliquePlus45 =
-1*smoothedInput[row-1][column-1]
-1*smoothedInput[row-1][column]
+2*smoothedInput[row-1][column+1]
-1*smoothedInput[row][column-1]
+2*smoothedInput[row][column]
-1*smoothedInput[row][column+1]
+2*smoothedInput[row+1][column-1]
-1*smoothedInput[row+1][column]
-1*smoothedInput[row+1][column+1];

if(obliquePlus45>threshold) isLine++;
```

```
      obliqueMinus45 =
      +2*smoothedInput[row-1][column-1]
      -1*smoothedInput[row-1][column]
      -1*smoothedInput[row-1][column+1]
      -1*smoothedInput[row][column-1]
      +2*smoothedInput[row][column]
      -1*smoothedInput[row][column+1]
      -1*smoothedInput[row+1][column-1]
      -1*smoothedInput[row+1][column]
      +2*smoothedInput[row+1][column+1];

      if(obliqueMinus45>threshold) isLine++;

      if(isLine>0){
        thresholdedMatrix[row-1][column-1] = 255;
      } else {
        thresholdedMatrix[row-1][column-1] = 0;
      }

      horizontalLine = 0;
      verticalLine = 0;
      obliquePlus45 = 0;
      obliqueMinus45 = 0;
      isLine = 0;

    }
  }

  //Sends processed input back to master
  for(int row = 0; row <transSize-2; row++){
    for(int column = 0; column<196; column++){
      int transNumber = thresholdedMatrix[row][column];
      MPI_Send(&transNumber, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
```

```
	}

		} else if(rank == size-1){  //processor n
			int transInput[transSize+1][200];//input array for this processor

	//receive input from master
			for(int row = 1; row <transSize+1; row++){
				for(int column = 0; column < 200; column++){
					MPI_Recv(&transNumber, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
					transInput[row][column] = transNumber;
					//printf("Process %d received input %d from process 0\n", proc, transInput[row][column]);
				}
			}

	//Send and receive border line
				for(int column = 0; column < 200; column++){

	  int transNumber = transInput[1][column];
			MPI_Sendrecv(&transNumber, 1, MPI_INT, proc-1, 0, &transNumber, 1, MPI_INT, proc-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
				transInput[0][column] = transNumber;

			}

	//SMOOTHING

	int smoothedInput[transSize][198];
	int smoothedNumber = 0; //Variable for produced number after taking mean

	//Takes mean of 3x3 matrix and inserts into smoothedInput array
	for(int row = 1; row < transSize; row ++){
	  for(int column = 1; column < 199; column++){
	    for(int i = -1; i < 2; i++){
```

```
            for(int j = -1; j < 2; j++){
              smoothedNumber = smoothedNumber + transInput[row+i][column+j];
            }
          }
          smoothedNumber = smoothedNumber/9;
          smoothedInput[row][column-1] = smoothedNumber;
          smoothedNumber = 0;

        }
      }

      //Send and receive border line
      for(int column = 0; column < 198; column++){

        int smoothedNumber = smoothedInput[1][column];
        MPI_Sendrecv(&smoothedNumber, 1, MPI_INT, proc-1, 0, &smoothedNumber, 1,
MPI_INT, proc-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        smoothedInput[0][column] = smoothedNumber;

      }




      int thresholdedMatrix[transSize-2][196];

      //Variables for produced numbers after multiplications
      int horizontalLine = 0;
      int verticalLine = 0;
      int obliquePlus45 = 0;
      int obliqueMinus45 = 0;

          int isLine = 0; // if it is greater than zero, one of above values has passed the
threshold
```

```
//Multiplies 3x3 matrix and inserts 0 or 255 into thresholdedMatrix array
for(int row = 1; row < transSize-1; row++){
  for(int column = 1; column<197; column++){
    horizontalLine =
    -1*smoothedInput[row-1][column-1]
    +2*smoothedInput[row-1][column]
    -1*smoothedInput[row-1][column+1]
    -1*smoothedInput[row][column-1]
    +2*smoothedInput[row][column]
    -1*smoothedInput[row][column+1]
    -1*smoothedInput[row+1][column-1]
    +2*smoothedInput[row+1][column]
    -1*smoothedInput[row+1][column+1];

    if(horizontalLine>threshold) isLine++;

    verticalLine =
    -1*smoothedInput[row-1][column-1]
    -1*smoothedInput[row-1][column]
    -1*smoothedInput[row-1][column+1]
    +2*smoothedInput[row][column-1]
    +2*smoothedInput[row][column]
    +2*smoothedInput[row][column+1]
    -1*smoothedInput[row+1][column-1]
    -1*smoothedInput[row+1][column]
    -1*smoothedInput[row+1][column+1];

    if(verticalLine>threshold) isLine++;

    obliquePlus45 =
    -1*smoothedInput[row-1][column-1]
    -1*smoothedInput[row-1][column]
    +2*smoothedInput[row-1][column+1]
    -1*smoothedInput[row][column-1]
    +2*smoothedInput[row][column]
```

```
      -1*smoothedInput[row][column+1]
      +2*smoothedInput[row+1][column-1]
      -1*smoothedInput[row+1][column]
      -1*smoothedInput[row+1][column+1];

      if(obliquePlus45>threshold) isLine++;

      obliqueMinus45 =
      +2*smoothedInput[row-1][column-1]
      -1*smoothedInput[row-1][column]
      -1*smoothedInput[row-1][column+1]
      -1*smoothedInput[row][column-1]
      +2*smoothedInput[row][column]
      -1*smoothedInput[row][column+1]
      -1*smoothedInput[row+1][column-1]
      -1*smoothedInput[row+1][column]
      +2*smoothedInput[row+1][column+1];

      if(obliqueMinus45>threshold) isLine++;

      if(isLine>0){
        thresholdedMatrix[row-1][column-1] = 255;
      } else {
        thresholdedMatrix[row-1][column-1] = 0;
      }

      horizontalLine = 0;
      verticalLine = 0;
      obliquePlus45 = 0;
      obliqueMinus45 = 0;
      isLine = 0;

    }
  }
```

```
        //sending thresholded values to master

        for(int row = 0; row <transSize-2; row++){
          for(int column = 0; column<196; column++){
            int transNumber = thresholdedMatrix[row][column];
            MPI_Send(&transNumber, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
          }
        }

              } else { //between processor 2 and n-1

                    int transInput[transSize+2][200];//input array for this processor

        //receive input from master
                    for(int row = 1; row <transSize+1; row++){
                                for(int column = 0; column < 200; column++){
                                        MPI_Recv(&transNumber, 1, MPI_INT, 0, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                                        transInput[row][column] = transNumber;
                                }
                          }

        //Send and receive border line
                                for(int column = 0; column < 200; column++){

          transNumber = transInput[1][column];
                    MPI_Sendrecv(&transNumber, 1, MPI_INT, proc-1, 0, &transNumber, 1,
MPI_INT, proc-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                                transInput[0][column] = transNumber;

          int transNumber = transInput[transSize][column];
                    MPI_Sendrecv(&transNumber, 1, MPI_INT, proc+1, 0, &transNumber, 1,
MPI_INT, proc+1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
          transInput[transSize+1][column] = transNumber;
```

```
                }


        //SMOOTHING

        int smoothedInput[transSize+2][198];
        int smoothedNumber = 0; //Variable for produced number after taking mean

        //Takes mean of 3x3 matrix and inserts into smoothedInput array
        for(int row = 1; row < transSize+1; row ++){
          for(int column = 1; column < 199; column++){
            for(int i = -1; i < 2; i++){
              for(int j = -1; j < 2; j++){
                smoothedNumber = smoothedNumber + transInput[row+i][column+j];
              }
            }
            smoothedNumber = smoothedNumber/9;
            smoothedInput[row][column-1] = smoothedNumber;
            smoothedNumber = 0;
          }
        }

        //Send and receive border line
        for(int column = 0; column < 198; column++){

          int smoothedNumber = smoothedInput[transSize][column];
            MPI_Sendrecv(&smoothedNumber, 1, MPI_INT, proc+1, 0, &smoothedNumber,
1, MPI_INT, proc+1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
          smoothedInput[transSize+1][column] = smoothedNumber;

          smoothedNumber = smoothedInput[1][column];
          MPI_Sendrecv(&smoothedNumber, 1, MPI_INT, proc-1, 0, &smoothedNumber, 1,
MPI_INT, proc-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
          smoothedInput[0][column] = smoothedNumber;
```

```
}


int thresholdedMatrix[transSize][196];

//Variables for produced numbers after multiplications
int horizontalLine = 0;
int verticalLine = 0;
int obliquePlus45 = 0;
int obliqueMinus45 = 0;

    int isLine = 0; // if it is greater than zero, one of above values has passed the threshold

//Multiplies 3x3 matrix and inserts 0 or 255 into thresholdedMatrix array
for(int row = 1; row < transSize+1; row++){
  for(int column = 1; column<197; column++){
    horizontalLine =
    -1*smoothedInput[row-1][column-1]
    +2*smoothedInput[row-1][column]
    -1*smoothedInput[row-1][column+1]
    -1*smoothedInput[row][column-1]
    +2*smoothedInput[row][column]
    -1*smoothedInput[row][column+1]
    -1*smoothedInput[row+1][column-1]
    +2*smoothedInput[row+1][column]
    -1*smoothedInput[row+1][column+1];

    if(horizontalLine>threshold) isLine++;

    verticalLine =
    -1*smoothedInput[row-1][column-1]
    -1*smoothedInput[row-1][column]
    -1*smoothedInput[row-1][column+1]
    +2*smoothedInput[row][column-1]
```

```
+2*smoothedInput[row][column]
+2*smoothedInput[row][column+1]
-1*smoothedInput[row+1][column-1]
-1*smoothedInput[row+1][column]
-1*smoothedInput[row+1][column+1];

if(verticalLine>threshold) isLine++;

obliquePlus45 =
-1*smoothedInput[row-1][column-1]
-1*smoothedInput[row-1][column]
+2*smoothedInput[row-1][column+1]
-1*smoothedInput[row][column-1]
+2*smoothedInput[row][column]
-1*smoothedInput[row][column+1]
+2*smoothedInput[row+1][column-1]
-1*smoothedInput[row+1][column]
-1*smoothedInput[row+1][column+1];

if(obliquePlus45>threshold) isLine++;

obliqueMinus45 =
+2*smoothedInput[row-1][column-1]
-1*smoothedInput[row-1][column]
-1*smoothedInput[row-1][column+1]
-1*smoothedInput[row][column-1]
+2*smoothedInput[row][column]
-1*smoothedInput[row][column+1]
-1*smoothedInput[row+1][column-1]
-1*smoothedInput[row+1][column]
+2*smoothedInput[row+1][column+1];

if(obliqueMinus45>threshold) isLine++;

if(isLine>0){
```

```
                thresholdedMatrix[row-1][column-1] = 255;
            } else {
                thresholdedMatrix[row-1][column-1] = 0;
            }

            horizontalLine = 0;
            verticalLine = 0;
            obliquePlus45 = 0;
            obliqueMinus45 = 0;
            isLine = 0;

          }
        }

        //send processed input to master

        for(int row = 0; row <transSize; row++){
          for(int column = 0; column<196; column++){
            int transNumber = thresholdedMatrix[row][column];
            MPI_Send(&transNumber, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
          }
        }

              }

      }
  }

  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize(); //MPI endsr

  return 0;
}
```