Parallel Algorithm for Smoothing and Line Detection

Mahmut Uğur TAŞ

2011104090

CmpE 300

Analysis of Algorithms

27.12.2016

INTRODUCTION

In this project, the concept of multiprocessor environment mpi is exemplified by an image processing operation. It is assumed that we have 200x200 matrix file of an image and by smooting this matrix and applying convolution operation we can detect lines that are appear in this image. Basically, the program executes matrix multiplication operation with some 3x3 matrices and create 198x198 matrix after smooting operation and 196x196 matrix after convolution operation. This 196x196 matrix is line detected version of our image.

PROGRAM INTERFACE

Program is written in C language and can be executed using command line. Program takes input file name and output file name and threshold value. Program terminates itself without prompting to user anything and puts result matrix to output file. MPI environment should be installed to compile and execute.

PROGRAM EXECUTION

A matrix of an image with 200x200 dimensions is given in input file is executed by this program. To compile it with command line “mpicc -g ‘programName.c’ -o ‘executableName’ ” is used. And to run this created executable, “ mpiexec -n ‘processorNumber’ ‘executableName’ ‘inputFile’ ‘outputFile’ threshold” command is used in command line.

In my case programName is proje. Processor number should be divider of 200 plus one such as 3, 5, 6, 41, 51, 101. Threshold can be 10, 25 or 40.

INPUT AND OUTPUT

Input file contains 200x200 integers. That are come from image. Only black and white pixels are represented at output file which will keep 196x196 matrix which is line detected version of given matrix of image. That means we have only 0 and 255 in output file.

PROGRAM STRUCTURE

In this section technical details of the program will be discussed.

First of all, *initialize* method initializes mpi environment and gets rang and total number of the processors. I used 0 processor as master processor and rest are slave processors. Then I create necessary arrays and variables. My first operation is reading file with master processor. For this operation, I have a function that reads file and add them to given matrix. This function takes four parameters that are size of array that will keep inputs, array, command line arguments, and number of processors. This function fills first processor part of array with 0's to prevent giving some part of input to master. Then reads file and fill array with inputs. After reading operation, I distribute data with MPI_Scatter function to slave processors. When slave processors get data, they add this input to their local matrices. After distribution operation, I passed data between slaves and add these received data to local matrices too. By doing this, I first send even number processors data to previous and next processors. While adding received data to local matrices, I check processor number because first and last processors are take only one line from their previous or next processors.

Second operation is smooting. Slave processors execute smooting operation on their local matrix. To do this, I create a function named as getArray which is take 3x3 matrix from local matrix of processors according to given positions. Such as when 1,1 is given position, this function return

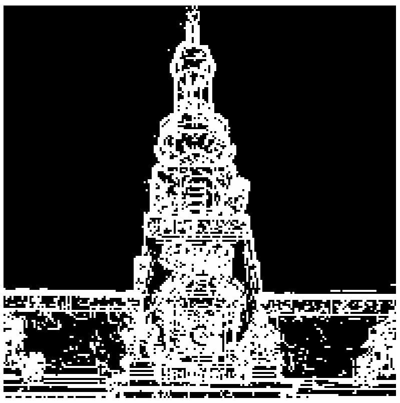
(00,01,02),(10,11,12),(20,21,22) as matrix. getArray function take six parameters which are row size and column size of local matrix, local matrix, row position, column position and an 3x3 array which will be used for smooting. After 3x3 array is accuired, I send this array to smoothing operation. I have a function for smooting operation named as mySmooth. This function calculates sum of given array entries and divide them to 9, and finally returns the result. After execution of mySmooth function, I add received result to a new matrix named as localSmoothMatrix. By doing this, I checked processors numbers. Because first and last processors have a special conditions. First line of local matrix of first processor should not be send to smoothing operation. After completion of smooting operation, I exchange data in bourder lines between slave processors. This process was the same as I done before. When this exchange operation complete, I add incoming data to localCompleteSmoothMatrix with own smoothed matrix of slave processors. Finally all processors get their smoothed data which are required to start next process.

Third operation is convolution. At this operation, I used getArray function again. After i get 3x3 matrix from getArray function, I send this matrix to a function which will execute convolution operation. This function is named as convol. This function takes two parameters which are 3x3 array and threshold. When it takes array, sends the taken array to mulArray function which handles multiplication operation of this function with given filter matrices. Then returns the results to convolution function, and if one of these result is greater then threshold which is taken as parameter then convol returns 255 otherwise returns 0. With this operation we get a matrix which is executed convolution operation on it.

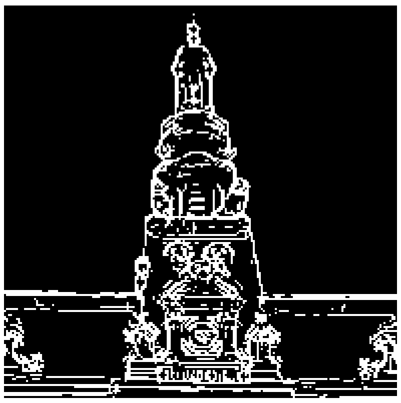
Final part of project is gathering data from slave processors. All slave processors send their localConvolArrays to master and master receives them. Then master prints them to output file.

SAMPLES

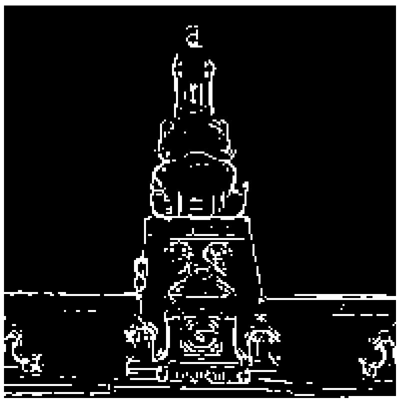
Samples are added to project file too.



Threshold 10



Threshold 25



Threshold 40

IMPROVEMENTS AND EXTENSIONS

I used to many space in this project to be sure there is no conflict between data. This can be improved by using same array and matrices carefully.

DIFFICULTIES ENCOUNTERED

Since this is the first time I used multiprogramming environment, getting used to mpi is essential part of this project for me. Sending and receiving messages without causing deadlock and handling lots of errors enhances my perspective of multiprogramming concept. I ended up with the idea of ensuring the equity of sending and receiving messages; and doing communication between adjacent processors at each iteration. This was the issue concerning slave processors.

Error handling is so hard in mpi. There is no messages why program doesn’t compiled or run. A simple mistake can cause to waste of hours. MPI has its own read function and i write a function to read file with name read. When i run it doesn’t give a proper error messages. This was the harded one in difficulties which i encountered.

CONCLUSION

Briefly, multiprocessor environment is different from sequential one and should be thought differently. Processors works like asynchronous system on web.