CMPE300 - ANALYSIS OF ALGORITHMS

Giray Eryılmaz

Instructor:
Tunga Güngör

Assistants:
Yiğit Kültür
Metehan Doyran

Smoothing and Line Detecting (PROGRAMMING PROJECT)

DATE : 27.12.2016

INTRODUCTION

Given an arbitrary visual, we are asked to apply smoothing and line detection. To increase time efficiency we are also asked to utilize multiple proccessors. In order to accomplish this we will implement the project in C programming language with the help of MPI (Message Passing Interface) libraries

Program Interface
The program is compiled and executed from terminal.

To compile the program use the command:

```
>>mpicc -g main.c -o  main
```

which will produce one executable file called main

Overall execution has the following format:

```
>>mpiexec -n <Processors> <executable> <input> <output> <threshold>
```

Example:

```
>> mpiexec -n 3 project.exe input.txt output10.txt 10
```

Here 3 processors the master and 2 slaves will be participated, name of the executable is project.exe, input visual is input.txt, output file is output10.txt and finally the threshold is 10.

To abort the process type ctrl + c

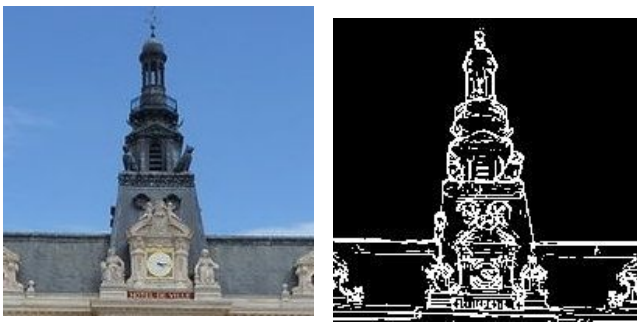
Program Execution:

-INPUT and OUTPUT

The program takes 3 parameters: input file (file path), output file and the threshold value(strictly an integer) to be applied while filtering.
Number of processors (n = master + slaves) will be participated has to be passed as well

Here for this project, input file is assumed to be a 200 to 200 visual in txt format.
Also number of slaves wanted to participate has to be a proper divisor of 200.
I.e. 200%(n-1) = 0 should hold.

Output will is a 196 to 196 txt representation of line detected visual.



Input on the left output on the right

– Program Structure

The program is implemented in C language with help of massage passing interface (MPI). There is one master processor and n-1 slave processors. Master process reads input from input file (txt formatted) to an array named picture and scatters it to slaves each having 200/p lines where p = n-1 ie number of slaves.

To scatter
MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
is used, since this function scatters to all processors including the master before scattering the visual 200/p lines is added in front of it so that the added part is send to the master and each slave got 200/p lines.

After scattering is done, processors get upper and lower lines needed to perform smoothing from their upper and lower neighbors respectively. In order to avoid deadlock first odd numbered processes get their needed lines (by that time even numbered ones send) then the opposite is done. These actions is accomplished with

MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag,MPI_Comm comm); //Performs a blocking send

MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,MPI_Comm comm,MPI_Status_IGNORE) //Blocking receive for a message

After it is made sure that every process has completed its part up to that point by using

MPI_Barrier(MPI_COMM_WORLD);

smoothing is performed.

Then again every process gets its needed upper and lower lines smoothed in its corresponding neighbors with the same send and receive functions.
After this information passing operations, final filtering and thresholding is performed and results are stored in local arrays.
Then all these local smoothed, filtered and thresholded values gathered at master processor via

MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Note that Scatter and gather operations practically provide synchronization as Barrier would. So that extra barriers are not needed.

Eventually gathered results are printed as final line detected visual by the master processor.

Notes About Parallel Programming

Image processing is a well parallelizable task since the same operation is applied to all parts of easily partitioned data. When the data (visual) is large enough we can ignore sequential parts and we can have T/p execution time. Though in this project we have only 200 to 200 visuals, that is why we feel effects of processor allocation, massage passing and sequential operations. Thus, infect when working on small visuals sequential program is more reasonable but otherwise of course execution time reduces to about T/p.

Notes About Chosen Thresholds

Clearly when the threshold is too low, output gets too complicated and "imaginary" lines appear. On the other hand when the threshold is too high, image disappears ie lines can not be detected. In my opinion around 20 is good enough.

Improvements and Extensions

Program can be generalized such that it can process other types of visuals (jpg etc) and can handle any size (not just 200 to 200 visuals).

Difficulties Encountered

- Usual floating point issues cost me a lot of time since expected output was not exactly the same as my output. This was resolved after relevant code snipped was provided.

-Being not C-prof. Was also a problem.

– Conclusion

Code is working properly. Deadlocks are avoided, program produces expected output with any number of slaves provided that it is a proper divider of 200 (1, 2, …, 200).

– Appendices

--Source code is included in a separate file, main.c