Bright Spots Detection Using MPI

Avinash Reddy Yenumula
Department of Computer Science
The University of Texas at San Antonio
San Antonio ,USA
wrh932@my.utsa.edu

Disney Davis Koola
Department of Computer Science
The University of Texas at San Antonio
San Antonio ,USA
oqx822@my.utsa.edu

Lokesh Reddy Gundla
Department of Computer Science
The University of Texas at San Antonio
San Antonio ,USA
vmh076@my.utsa.edu

***Abstract*—In this paper we have implemented the process of detecting bright spots from any large image. This is achieved using opencv, parallel communication and adaptive threshold value.**

***Keywords*—mpi, opencv, adaptive threshold, image processing, dynamic slicing, python,**

I. Introduction

To identify any sort of white pixel in an image , we have determined a methodology using opencv and MPI. We used opencv to obtain a greyscale image which was originally a color image. Opencv is integrated with a numpy array which helps us to convert image values into numpy array values. To make computation faster, image is sliced and sent to worker nodes. Once the computation is completed each node retune their individual computation value to the master node. Thereby, total number of bright spots in an image is determined.

II. Background

A. OpenCV

Opencv is an image processing tool which reads images at a faster rate when compared to the other libraries. The purpose of using opencv in our project is to read the image from a working directory or a full path of image using cv2.imread(). It can read images in three different ways and they are color, greyscale and unchanged respectively. But for our project we need the image to be read in greyscale mode for which we have to pass integer 0. Its 1 and -1 respectively for color and unchanged.

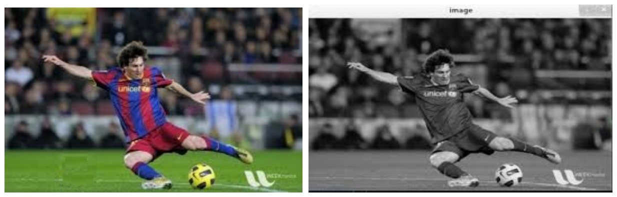


Fig. 1. A screenshot of the image converted to greyscale would look like this.

From fig.1 we can see that image of Messi which a color image is originally was converted into a greyscale image. As a conclusive proof we were able to read larger images at faster rate within less duration.

B. Adaptive Threshold

In the case of simple thresholding the threshold values are assigned globally which means that they would be same for all the pixels in the image, since we need different pixel values at different portions of the image, we use a method called adaptive thresholding. In this method threshold values are calculated even for smaller regions which ensures different threshold values for different regions. It is performed using OpenCV on an image using method cv.adaptivethreshold() which is of Imgproc class. This method accepts various parameters and let's see them one by one.

- src (jpg, tif) – This is the name of the source image which will be provided after converting to greyscale.
- dst- This will be the name of the image with which it will be saved after applying adaptive threshold.
- maxValue (double)- It is a value to be given if the pixel value is more than the threshold value
- adaptiveMethod (int)- It determines the adaptive methods to be used, in general there are of two types
    - ADAPTIVE_THESH_MEAN_C- This gives the mean value of the neighboring values obtained.
    - ADAPTIVE_THESH_GAUSSIAN_C-This gives the weighted sum of the neighboring values obtained.
- threshholdType (int)- It represents the type of the threshold to be used.

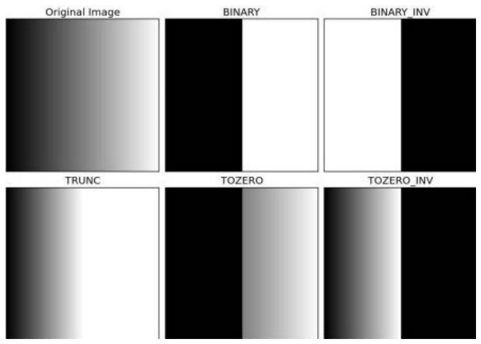


Fig. 2. From left to right these are various images obtained after using different types of threshold

These are the names of various types of thresholds of the above image from left to right cv.THRESH_BINARY, cv.THRESH_BINARY_INV, cv.THRESH_TRUNC, cv.THRESH_TOZERO, cv.THRESH_TOZERO_INV

- blockSize (int)- It is used to calculate the size of pixel neighborhood, which is used to determine the threshold value of the image.
- C (double)- It is a constant which is used to subtract values from mean or weighted mean.

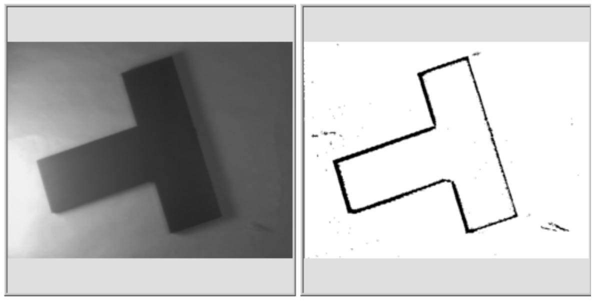


Fig. 3. A screenshot of the image after applying adaptive threshold

C. MPI

Also known as Message passing interface is widely known as a message passing specification of library interface. It mainly addresses parallel programming model where data is moved from space of one process to other through operations on each process. An extension to this model is provided in collective operation and that is what we are considered about in our project. Collective communication is a kind of communication which involves communication of group or groups of processes. In this we are interested in two methods scatter and gather.

- Gather - If the comm is an intracommunicator then each of its process sends the contents of send buffer to its root buffer. We will get an outcome where each of the n process had executed.
-
- Scatter – If comm is an intracommunicator then we will get an outcome where roots executes n operations.

III. METHODOLOGY

Implementation of this idea needed several in built libraries from python such as numpy, Image, opencv(cv2) and MPI. Once all these libraries are imported comm is used as a communicator for the process, in order to determine the rank and size of the number of process. Image is converted into numpy array using opencv integration with numpy. Leading to slicing of images based on the number of process. Collective communication is used so that the computation time is faster and optimized. Though collective communication is faster we have implemented the same using point to point communication to distinguish computation time results. In conclusion we were able to find the number of bright spots in an image.

A. Image Read using Open CV

Open CV read functionality was implemented in the program to convert the colourful image of large size to grayscale image. CV2.imread was used to convert an colourful image to grayscale image using value 0(indicated greyscale conversion). To optimize the program computation time and keep track of overall time needed to process the request, we calculate the image read processing time, node computational time and the computation time for bright spots.

- *OpenCV reads colorful image to grayscale image. Grey scale image in then converted into numpy array value.*
- *The image is converted to a matrix of values based on the color. Matrix is consisted of numerical values varying from 0 to 255.*
- OpenCV is used for faster processing and has inbuilt integration with numpy to convert into matrix values which makes easier to compute.

B. Dynamic Slicing of images

Computation of image in a single process is either time consuming or getting timed out error. Since the time consumption is more in single process we thought of scaling this computation using parallel computing(MPI). Scalability comes into effect once we were able to slice the image based on the number of process. Once the image is sliced it is sent to respective nodes so that all the worker nodes are equally provided with work and no other processis idle.

1) Image Slicing: Initially slice the image horizontally based on the number of process. Once the image is sliced for the first part, check the remaining slice based on the rank order. Each row size is considered as a slice and further slicing is done based on the remaining image slice. All the images are sliced and sent to respective node using scatterv so that all worker nodes perform equal computation.

2) Broadcast and Scatterv: Array dimensions converted from image gray scalevalues is sent to all worker nodes using broadcast. Then scatter the image array slices equally to all the worker nodes for computation.

C. Adaptive Threshold

Implementation of adaptive threshold using mean values with surrounding pixel. Window size of 59 is used to scan all the pixels in the sliced images to determine the mean value greater than 200. Total count of mean values which is greater than 200 is summed up to determine the total bright spots count.

Sliced images bright spots count is calculated in all the worker nodes and returned to the master node. All the communication happens through MPI(MPI.SUM). All the bright spots count from each worker node is summed up in the master node. Hence the total bright spots count and time was determined.

D. *Point to Point Communication*

Implementation of bright spots count in an image is calculated using MPI send and MPI receive. Similar to collective communication we slice the images based on the number of processes. Once the image is sliced, it is sent to multiple destination nodes through point to point communication.

Sliced image parts are sent to multiple worker nodes with unequal work division. As a result computation time for each worker nodes is relatively more than collective communication. Dictionary is used to compute all the bright spots count in worker nodes for point to point communication.

Image is reconstructed to verify the dimension of the original image with the total sum of individual worker node sliced image.

Reconstruction is done using numpy horizontal stack from the sliced images of the worker nodes. Computation time and bright spots count was also determined from point to point communication.

E. *Figures and Tables*

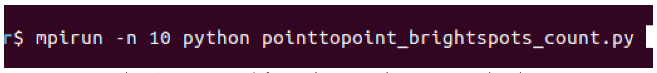
Time/count	Point to Point Communication	Collective Communication
Time taken to open and read	1.143 sec	1.139 sec
Time taken to count the bright spots	4.071 sec	0.298 sec
Total bright spots counted	1583027 spots	1591787 spots

Fig. 4. Side by side comparison of values obtained from both the programs

IV. Results

A. *Point to Point Communication outputs*

Run command:

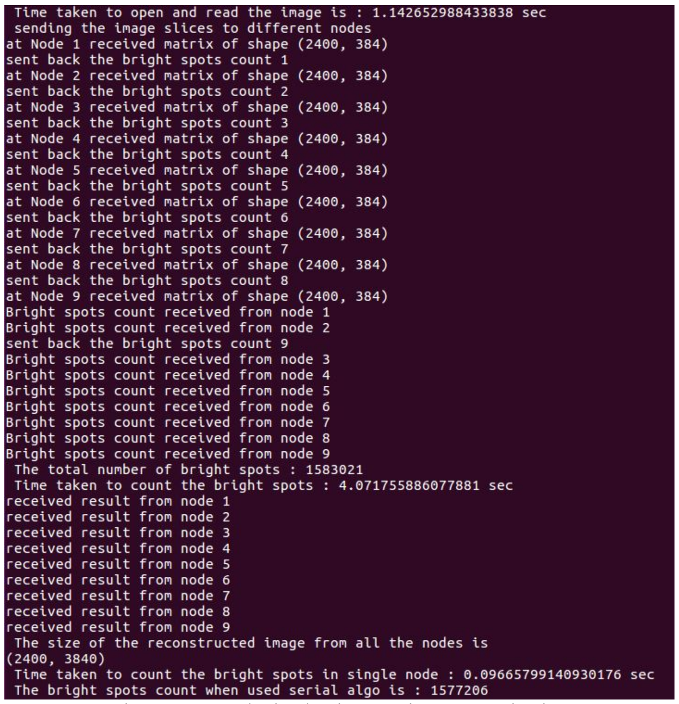


```
r$ mpirun -n 10 python pointtopoint_brightspots_count.py
```

Fig. 4. command for point to point communication

OUTPUT

We have captured the transmission of data from one node to other node in point to point communication and also the time taken to open and read the image. Finally, we have obtained the total time taken to count the total number of bright spots from all the node which is 4.071 sec.



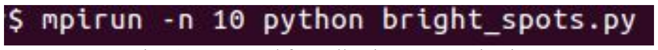
```
 Time taken to open and read the image is : 1.142652988433838 sec
 sending the image slices to different nodes
at Node 1 received matrix of shape (2400, 384)
sent back the bright spots count 1
at Node 2 received matrix of shape (2400, 384)
sent back the bright spots count 2
at Node 3 received matrix of shape (2400, 384)
sent back the bright spots count 3
at Node 4 received matrix of shape (2400, 384)
sent back the bright spots count 4
at Node 5 received matrix of shape (2400, 384)
sent back the bright spots count 5
at Node 6 received matrix of shape (2400, 384)
sent back the bright spots count 6
at Node 7 received matrix of shape (2400, 384)
sent back the bright spots count 7
at Node 8 received matrix of shape (2400, 384)
sent back the bright spots count 8
at Node 9 received matrix of shape (2400, 384)
Bright spots count received from node 1
Bright spots count received from node 2
sent back the bright spots count 9
Bright spots count received from node 3
Bright spots count received from node 4
Bright spots count received from node 5
Bright spots count received from node 6
Bright spots count received from node 7
Bright spots count received from node 8
Bright spots count received from node 9
 The total number of bright spots : 1583021
 Time taken to count the bright spots : 4.071755886077881 sec
received result from node 1
received result from node 2
received result from node 3
received result from node 4
received result from node 5
received result from node 6
received result from node 7
received result from node 8
received result from node 9
 The size of the reconstructed image from all the nodes is
(2400, 3840)
 Time taken to count the bright spots in single node : 0.09665799140930176 sec
 The bright spots count when used serial algo is : 1577206
```

Fig. 5. Output obtained point to point communication

B. *Collective Communication Outputs*

Output is calculated using mpirun command.

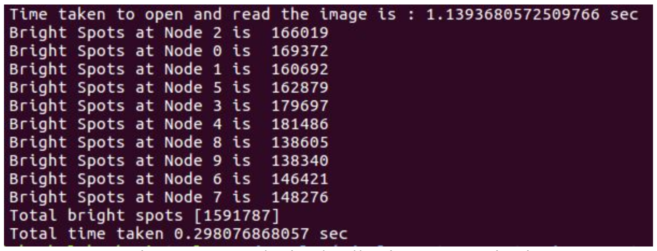
Run command:



```
$ mpirun -n 10 python bright_spots.py
```

Fig. 6. Command for collective communication

OUTPUT



```
Time taken to open and read the image is : 1.1393680572509766 sec
Bright Spots at Node 2 is  166019
Bright Spots at Node 0 is  169372
Bright Spots at Node 1 is  160692
Bright Spots at Node 5 is  162879
Bright Spots at Node 3 is  179697
Bright Spots at Node 4 is  181486
Bright Spots at Node 8 is  138605
Bright Spots at Node 9 is  138340
Bright Spots at Node 6 is  146421
Bright Spots at Node 7 is  148276
Total bright spots [1591787]
Total time taken 0.298076868057 sec
```

Fig. 7. Output obtained collective communication

We have also obtained the count of number of bright spots in each node which can be seen in the above figure when 10 nodes are used to process the image. It may vary depending on the number of nodes and size of the image taken for computation. Finally, the total time to count all the bright spots is 0.29807sec which is very less when compared to time taken in point to point communication.

V. Conclusion

Total bright spots count, in an image is calculated using collective communication as well as point to point communication. Point to point communication is implemented in order to verify the time consumption, bright spots count and reconstruction of image from all the nodes. As a result we were able to verify the bright spots count and computation time. Computation time from all worker nodes were calculated individually for point to point and Collective communication.

Overall we were able to conclude that the total computation time for collective communication is less than point to point communication. Total bright spots detection and image reconstruction is more precise for collective communication. Thereby performance is improved in Collective communication.

References

[1]https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html

[2]https://www.tutorialspoint.com/opencv/opencv_adaptive_threshold.html

[3]https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.

[4]https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pd

[5] https://pythonprogramming.net/thresholding-image-analysis-python-opencv-tutorial/

[6] https://computing.llnl.gov/tutorials/mpi/