

Laboration 1 - TDDC78

Jonathan Karlsson - jonka293 - 890201-1991

Niclas Olofsson - nicol271 - 900904-5338

21 maj 2013

1 Program description

1.1 Threshold filter

This program calculates the average intensity of the image and makes all pixels with a higher-than-average value white, and all other pixels black.

The whole image is read on the root node. The root node also calculates the average intensity of the image. The calculated intensity is sent to all other nodes via `MPI_Broadcast()`.

The image data array is split in as many parts as there are nodes, and each node gets its own part via `MPI_Scatter()`. Since each node needs to know the data length to receive, this is done in two steps where the total data length is first sent via MPI broadcast, and then the data is sent with MPI scatter. Each node then runs the threshold filter on its own part of the image. `MPI_Gather()` then reassembles the resulting image, which is written to disk by the root node.

1.2 Blur filter

For each pixel this program calculates the average color of the neighbors with a given radius, of the pixel and sets the pixels color to the average. This gives a blurred effect over the whole image with a small radius giving almost the same image back, and a big radius gives a very blurry image.

The whole image is read at the root node. The intervals that each processor is going to work in is calculated. Since the algorithm is dependent on the value of the neighbours of each pixels, we have to send some more image information than we are actually blurring on each node, otherwise there will be some broken lines in the image. The data is distributed to the nodes using `MPI_Scatterv()`. Now each chunk is iterated by the blur filter and then transmitted to the root node with `MPI_Gatherv()`, the root node must take special care to get rid of some overhead created by the sending of extra data. This is done by pointing a bit forward in the array storing the image.

2 Execution times

Below are graphs showing the execution times of our blur- and thresh filter implementations. We notice that the thresh filter scales poorly since it's such a small problem and the overhead

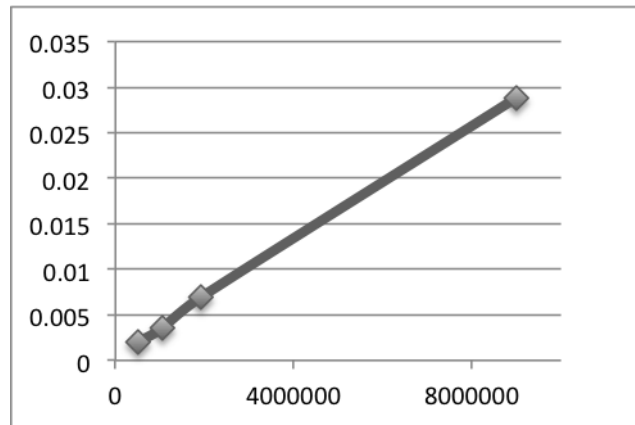


Figure 1: Using the same amount of nodes on different images. (Thresh)

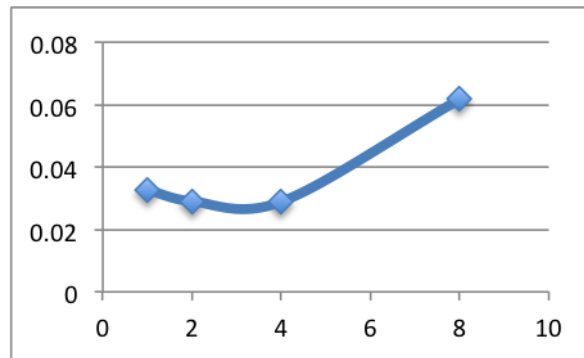


Figure 2: Using the same image with different number of nodes. (Thresh)

of communication is too big. The blur filter scales better, but we notice some overhead when we need to send to different processors to access our nodes.

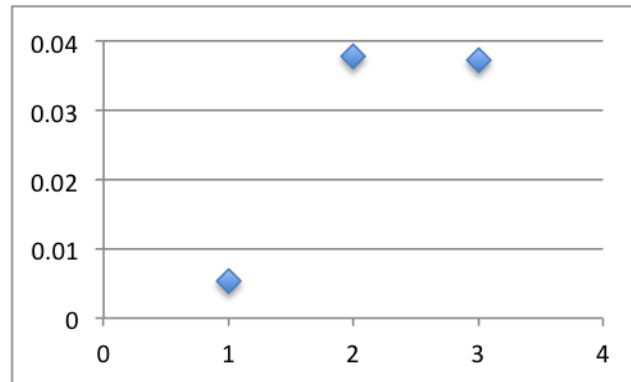


Figure 3: Scaling the image at the same rate as the number of nodes. (Thresh)

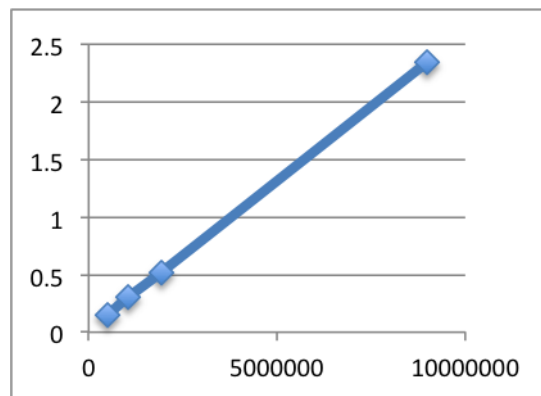


Figure 4: Using the same amount of nodes on different images. (Blur)

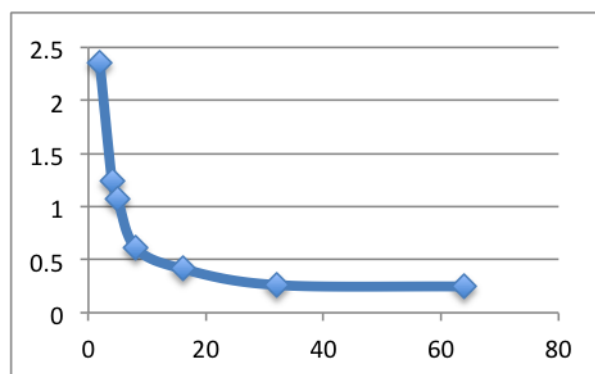
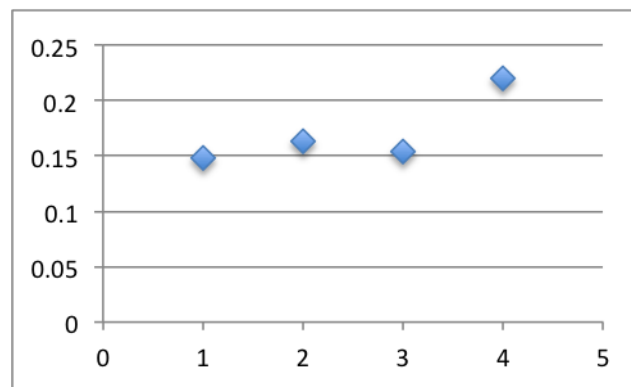


Figure 5: Using the same image with different number of nodes. (Blur)



Figur 6: Scaling the image at the same rate as the number of nodes. (Blur)