Laboration 2 - TDDC78

Jonathan Karlsson - jonka293 - 890201-1991
Niclas Olofsson - nicol271 - 900904-5338

May 22, 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 main thread. It also calculates the average intensity of the image. A data structure is created for each new thread that is to be created. This structure contains the intensity level, a pointer to the image itself as well as a start and an end value that defines the area of the image on which the thread is going to apply the filter. The threads is created and each of them start to work in their part, since memory is shared the whole image is done when all threads is done. Each thread is operating on different parts of the image so there is no risk of memory being overwritten. It is also not dependent on any other parts of the image, thus making this filter easy to parallelize.

When each thread is done, threads are joined again and the main thread writes the resulting image to disk.

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 large radius gives a very blurry image.

Just as in the threshold filter, the image is read by the root node. Gaussian weights needed for the filter is also calculated here. The structure used for passing parameters to the threds contains some additional data compared to the thresh filter, such as the generated Gaussian weights and the blurring radius.

Since the resulting value of each pixel depends on neighbour pixel data, we use different arrays for reading than for writing the resulting (blurred) image. This way, we do not need any more synchronization since all write operations is done on different parts of the image. We have also changed the implementation of the blur filter, so that it only filters a certain part of the image,

depending on the start and end values calculated by the main thread.

After running the filter, threads are joined and the result is written to disk by the main thread.

2 Execution times

In figure 1 and figure 4 the same number of nodes is used on different images and we see that the execution time is increased in a linear manor, for the thresh and blur filter. Figure 2 and figure 5 shows how the execution time decreases with increasing number of nodes for thresh and blur. Figure 3 and figure 6 shows how the same increase in image size and number of nodes give the same execution time, for 6 this is not exactly the case, possibly due to communication overhead when using too many nodes.

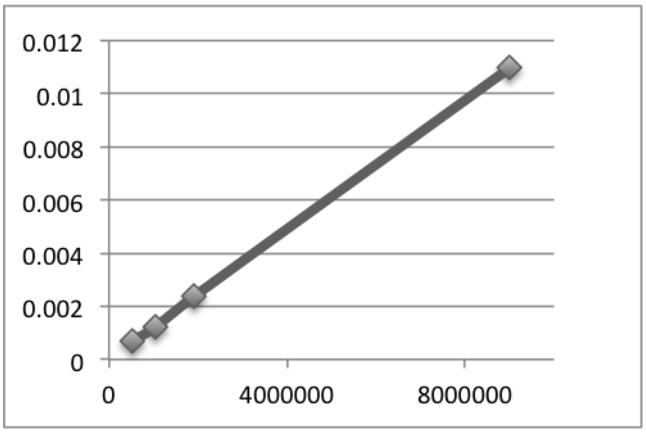

Figure 1: Using the same amount of nodes on different images. (Thresh)

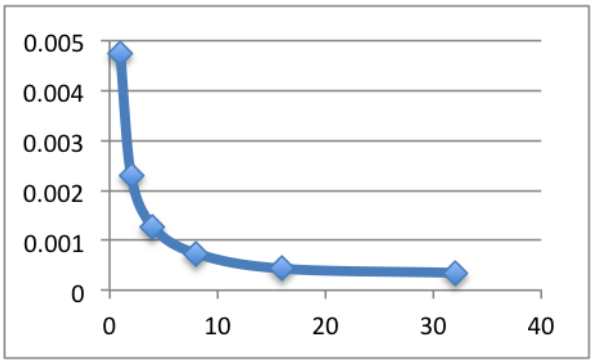

Figure 2: Using the same image with different amount of nodes. (Thresh)

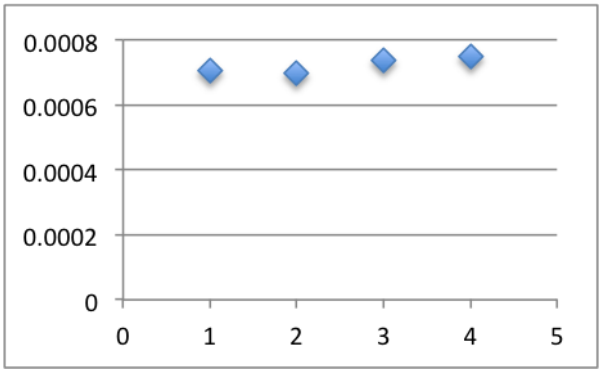

Figure 3: Scaling the image at the same rate as the number of nodes. Using 2, 4, 8 and 38 nodes. (Thresh)

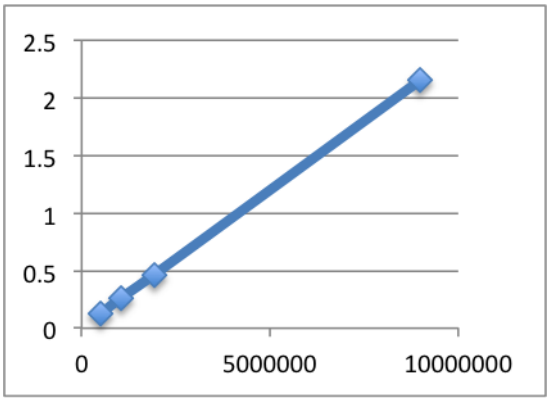

Figure 4: Using the same amount of nodes on different images. (Blur)

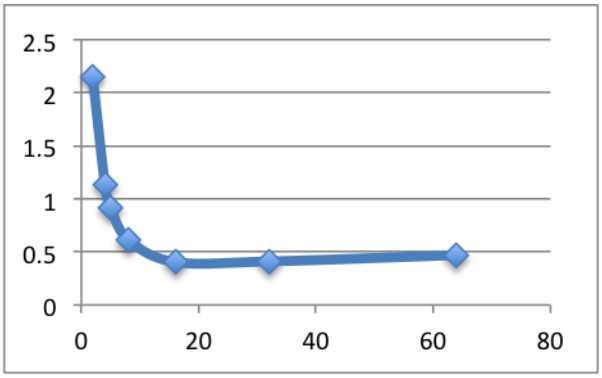

Figure 5: Using the same image with different amount of nodes. (Blur)

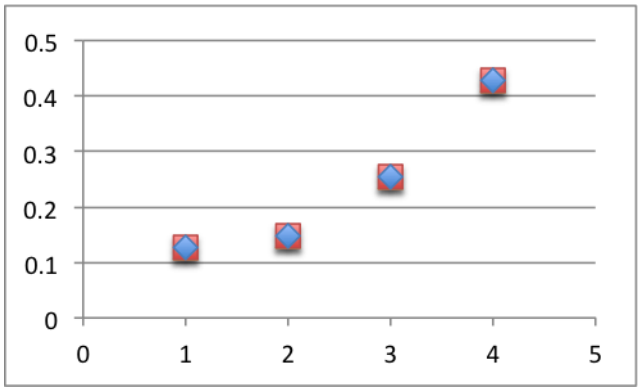

Figure 6: Scaling the image at the same rate as the number of nodes. Using 2, 4, 8 and 38 nodes. (Blur)