# Lab1 TDDC78 – Filtering with MPI

*Jonathan Bosson, jonbo665*

## 1 Program Description

### 1.1 Threshold filter

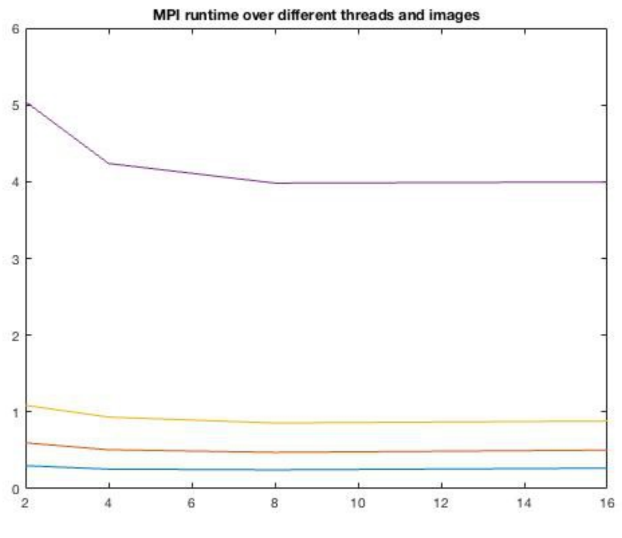
The threshold filter program calculates the average of each pixel in the image. At a later stage each pixel is compared with the average value and if the intensity is larger than the average its value is set to white and smaller set to black.

To utilise multiple processes, the work is split horizontally in equal parts.The images dimensions is then broadcasted to all threads through **MPI_Bcast()**. Each node calculates its own local average value with its assigned pixels. Once done the function **MPI_Allreduce()** is called to combine the values from all processes and distribute the result back to each node. Once done a total average value is found it can be used as the weighting value in the **thresfilter()** function. The last thing to do is to gather all results from the different processes and finalise an image which is done through **MPI_Gatherv**.

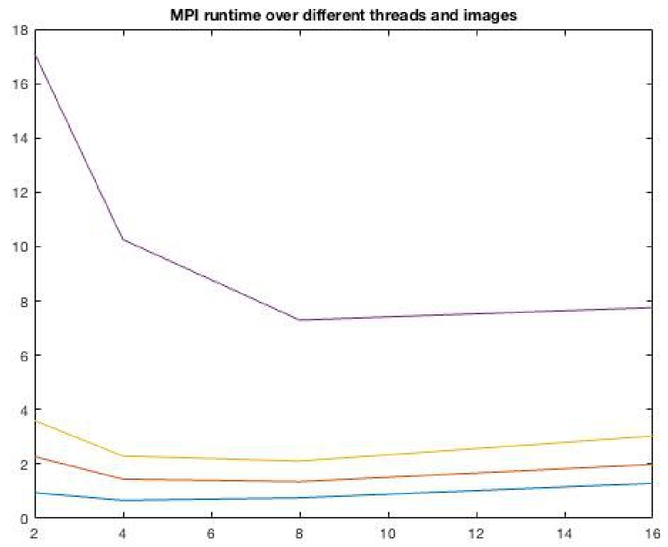
### 1.2 Blur filter

Instead of weighting the average value, the blur filter calculates the average value in a certain radius around a pixel to assign that color to the pixel. The result is a blurred picture. The bigger the radius is the more will it blur the image. The image is read at the root node and the broadcasted out to the other processes along with the chosen radius. Just like with the threshold filter is the work divided equally amongst the available threads through a horizontal split. The algorithm takes a radius which means it may need pixel data that belongs to another process. This is distrivuted by using **MPI_Scatterv**. **MP_Gatherv** is used to collect all results from the different processes. Here does the root node need to 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 Result



The image above shows the threshold filter program with different images and number of processes. The purple line is the biggest problem (im4.ppm) and the blue is the smallest (im1.ppm). We can see that the biggest gain is parallelizing with a low amount of processes in total. As more threads are introduced the gain will be smaller since more communications must be done by the MPI to distribute the shared variables.



Second image shows different images and number of processes but with the blur filter program. Similar conclusions can be drawn from the image, but it seems the benefit of 8 threads on large problems (im4.ppm) is more clear on the blurfilter than the threshold filter.

Looking at the MFLOP/s it can be seen that the value is close to the same at all problem sizes at a fixed number of cores as well as proportional to the number of cores..

| Threads: | 2 | 4 | 8 | 16 |
|---|---|---|---|---|
| img1 | 27.5 | 64.62 | 134.2 | 246.68 |
| img2 | 28 | 66.08 | 141.5 | 266.9 |
| img3 | 28.3 | 65.86 | 143.55 | 278.78 |
| img4 | 28.55 | 67.97 | 144.68 | 288.36 |

This table describes the MFLOP/s from the threshold filter program.