# TDDC78: Lab Report

Lab 2: Pthreads


| **Name** | **PIN** | **Email** |
|---|---|---|
| Alexander Yngve | 930320-6651 | aleyn573@student.liu.se |
| Pål Kastman | 851212-7575 | palka285@student.liu.se |

# Contents

# 1 Introduction

This lab consists just as the previous lab 1 of two image filters: blur filter and threshold filter.

The difference from the first lab where we used MPI is that we will instead use Pthreads in this lab. All the threads will have access to the same data so that we won't have to distribute this to the new threads.

## 1.1 Blur filter

The blur filter uses a normal distribution together with a given radius to calculate the mean value for every pixel in an image, this will create a blur the given image.

## 1.2 Threshold filter

The threshold filter first calculates a mean value for every pixel in the image, it will then go through the image one more time and either set every pixel to black or white depending if the pixel value is over or under the calculated mean value.

# 2 Our implementation

This section will describe how we used MPI to parallelize the execution of the programs.

## 2.1 Blur filter

Just as in the first lab, each thread will have an overlapping region of the image that they will apply the filter to. But in this lab the threads will have access to the same data.

To give the threads this data we define a struct called thread_data which contains pointer to all the necessary data, which is calculated by the first thread before it starts the other threads. It also contains pointer to the image and an intermediary image.

When we create the threads we pass in the struct along with the function thread_blur_x, this will run the filter in the x axis, and update the intermediary image. The main thread will wait for all the threads by running pthreads_join. When all the threads are done, the main thread will then create new threads but instead pass in the thread_blur_y function which will run the filter in the y axis.

The reason we do it this way is that the threads are now working on the same memory and if we run the filter as it is, then one thread might read a value that another thread have changed. This is not good, and we want to avoid it.

The main thread will then save the result and exit.

### 2.2 Threshold filter

Here we define a struct just as in the blur filter, we assign areas to the new threads and start them. The threads will run the function threshold_average, this function will calculate the average values of their area and save this to the struct.

When all the tasks are done, the main task will calculate a total average value from the various average values in the structs. Then it will save this value in all the structs and start new threads, passing in the threshold_filter function to the threads, and now they will run the filter.

When done the main thread, which have waited for the other threads, will save the result and exit.

## 3 Execution times

In this section we will present the execution times of the filters.

It is worth mentioning that the times for all graphs below are in seconds.

The tests were performed on four images with sizes as can be seen in 1.

**Table 1** – The sizes for the images used in the tests.

| **Image name** | **x pixels** | **y pixels** |
|---|---|---|
| im1.ppm | 676 | 763 |
| im2.ppm | 1024 | 1024 |
| im3.ppm | 1600 | 1200 |
| im4.ppm | 3000 | 3000 |

We accidentally made the graphs for lab1 from the results of image 1 and image 2, and the graphs for lab 2 from the results of image 3 and image 4 and hence, the reader can't compare the difference in execution times between pthreads and MPI. This wasn't not intentionally done.

### 3.1 Blur filter

Our result show that if we increase the number of threads, we will get a decrease in time. But this relationship is not linear due to deminishing returns, since the problem is not perfectly parallell the serial section sections start to dominate the execution time.

The results for image 3 and image 4 can the seen in figure 1 and figure 2 respectively.

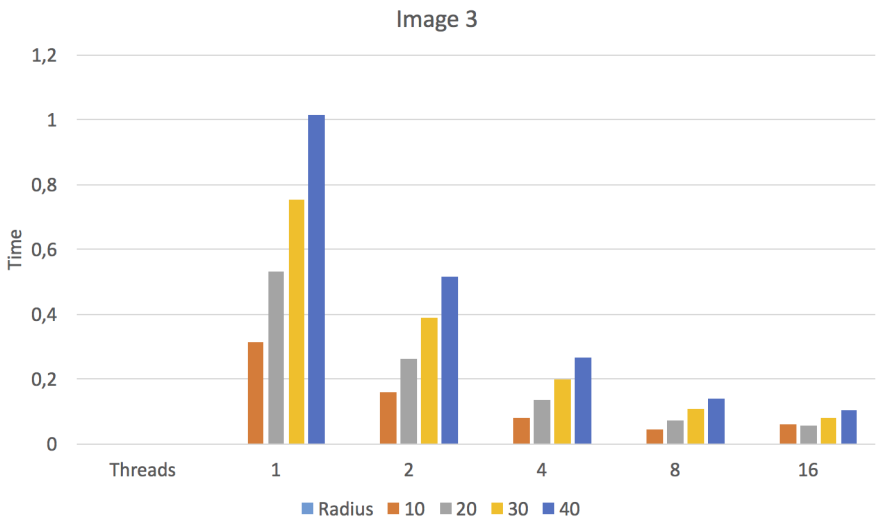


**Figure 1** – Result for the blur filter run on image 3.

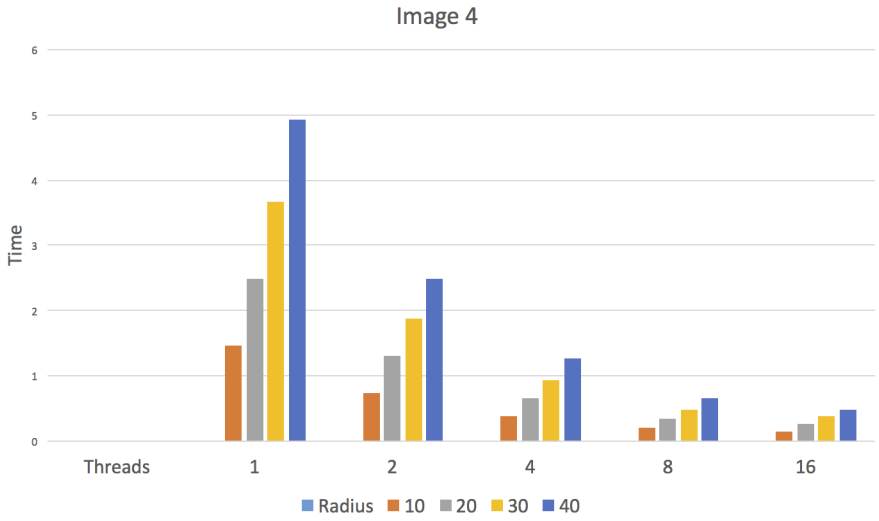


**Figure 2** – Result for the blur filter run on image 4.

## 3.2 Threshold filter

We obtained the same pattern of results for this filter as for the blur filter. Executions times for all images can be seen in figure 3

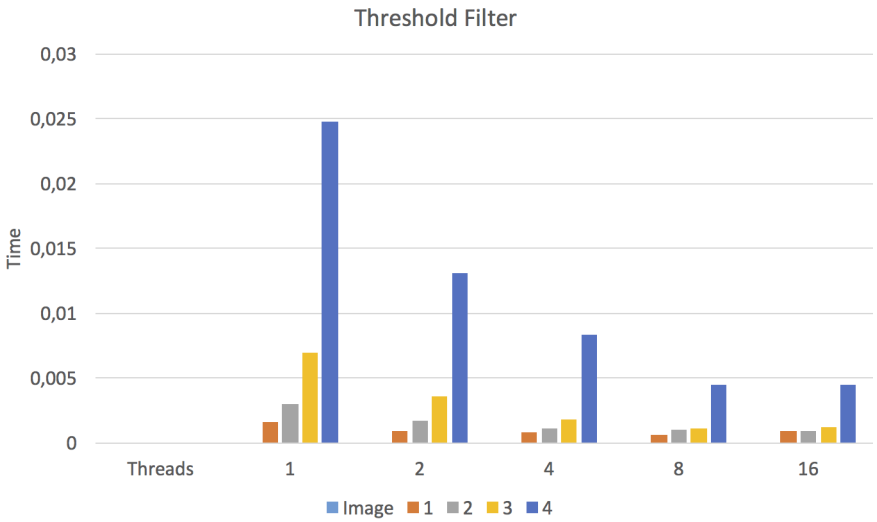


**Figure 3** – Result for the threshold filter.