Lab2 TDDC78 – Filtering with OpenMP

Jonathan Bosson, jonbo665

1 Program Description

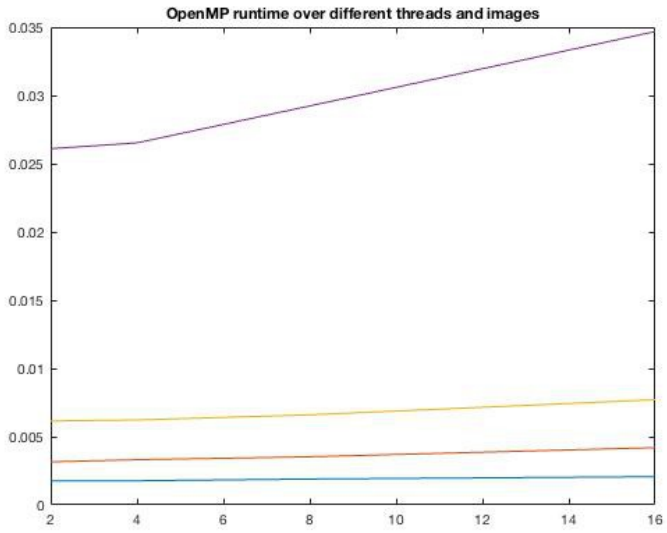
1.1 Threshold filter

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. The image data is saved in a global variable called **threadData.** Once done, the threads are created to run the threshold filter function. The process grabs the global variables and computes the average in its assigned area. The function **pthread_join()** is used to wait until each process is done before saving the newly created weighted image.

1.2 Blur filter

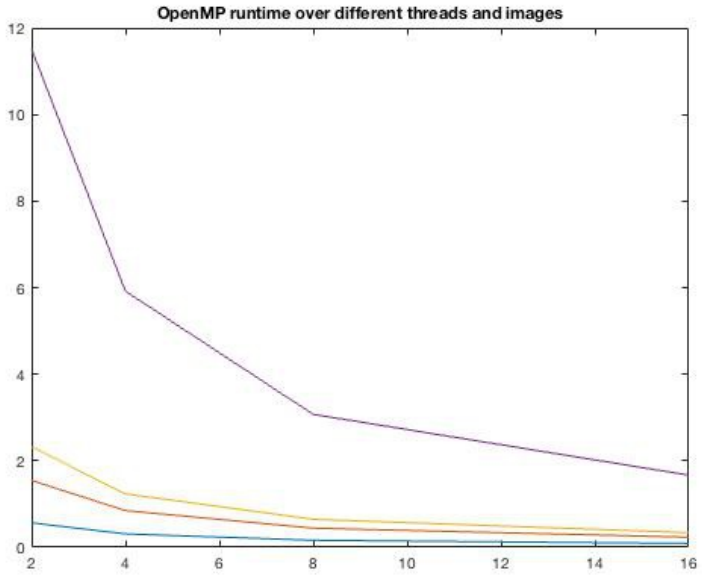
Instead of weighting the average value, the blur filter calculates the average value in a certain radius around a pixel to assign that color to the pixel. The result is a blurred picture. The bigger the radius is the more will it blur the image. The same process with saving the image data to a global variable used in the threshold filter is used in the blur filter as well. The image is split in equal horizontal parts for each process to handle and then sent in to a blur filter with **pthread_create()**. Once the processor is done blurring the image on the width it will wait until all processes are done with **pthread_join.** Once done it will repeat the exact same procedure is repeated with the height.

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). Most executions time had very similar result no matter the size of the problem or the amount of threads involved. Strangely enough did the result from execution the threshold filter on the biggest problem

(im4.ppm) with the highest amount of processes return the highest time. The amount of threads involved should not complicate things as it does with **MPI** since **OpenMP** uses a shared memory system. That means that the amount of extra communications between having for example 8 threads or 16 threads is small.



Second image shows different images and number of processes but with the blur filter program. Here the results looks more reasonable. It is clear that no matter the size of the problem does the blurfilter benefit from all amount of parallelization, being most effective with 16 processes.

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	3657	11969	12696	22775
img2	2723	8909	9494	17928
img3	3309	11206	11890	22731
img4	3135	10964	11752	21635

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