Authors: Filip Källström filka786; Emil Lindberg emili903 Date: 2013-05-20

Report lab 2 – Image Filter (pthreads)

Description

Blur

We start by reading the image and calculate displacements for the given number of threads. Each thread will have a range of rows it's supposed to blur with the given radius.

When the displacements is calculated the only thing we have to do is to create the threads and let them call the blurfilter function. Since we have shared memory every thread will write to the same target image so we don't have to do any gathering of the image that we had to do in the MPI version of blur. When the threads are done with the blur all we have to do is join the threads and then write the blurred image to file.

We have also created a couple of structs that contains the arguments each thread need in the blurfilter function. One struct which contains thread private data is created for each thread and another struct is just created once and contains the shared data.

Threshold

As before we start by reading the image and calculate displacements for the given number of threads. The displacements and work amount will in this case point out which pixels each thread is supposed to apply the threshold filter to. Of course we let each process get the same amount of pixels to work with. In the cases where this is not possible we distribute the left over pixels to to as many processes as possible.

We now create the threads and let them call the filter function. Since we now have to calculate the total sum and we are working with shared memory we use a conditional lock on the sum variable to make sure that sum is calculated correctly and that no thread is starting to calculate the average intensity before the sum calculation is completed.

With the average intensity calculated the threads can just loop through their part of the image and apply the threshold filter on the pixels.

At the end we just join the threads together and writes the image o file.

As before we have a couple of structs for thread private data and shared data to store the arguments to the filter.

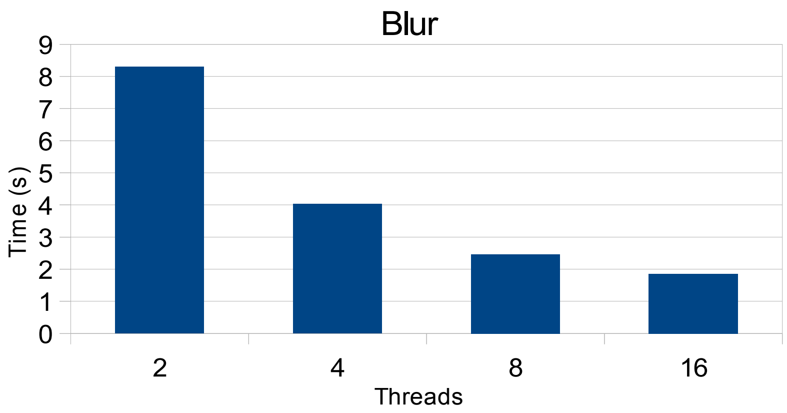
Execution times

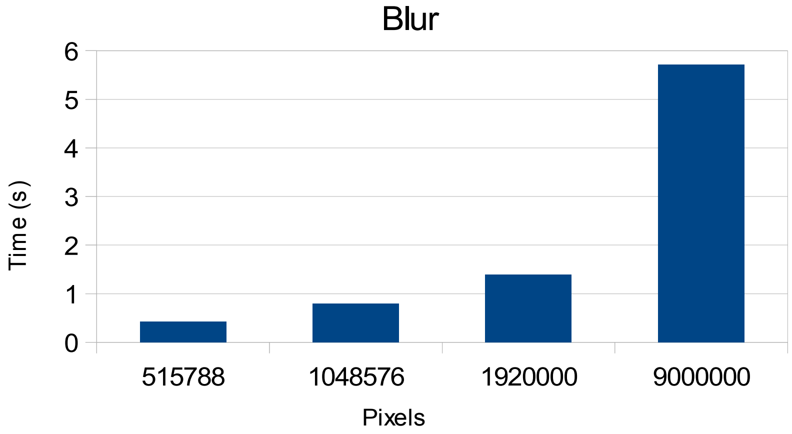
The data for the diagrams below is averages of 5 runs and can be seen in the appendix at the end of this report.

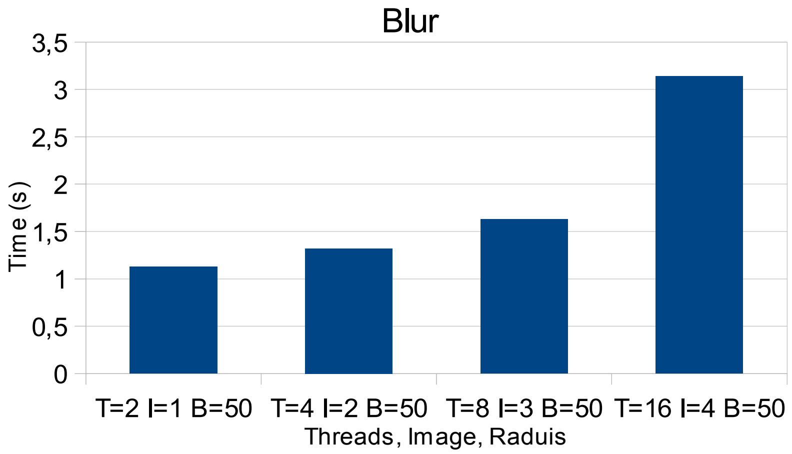
Default values for the variables if they are not changed in the diagrams are:

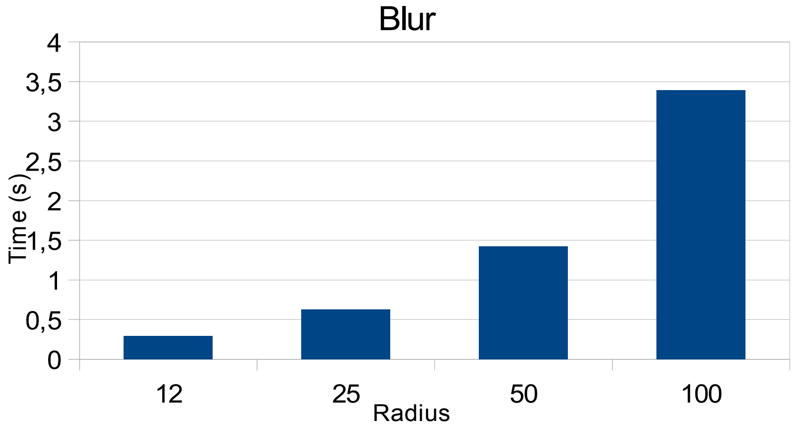
- Threads: 8
- Image 3: 1920000 pixels
- Radius: 50

Blur


Blur
Time (s)
0
1
2
3
4
5
6
7
8
9
2
4
8
16
Threads

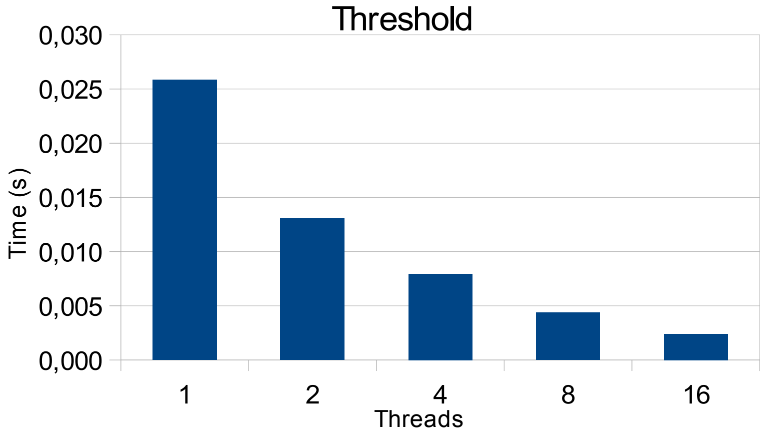

Blur
Time (s)
0
1
2
3
4
5
6
515788
1048576
1920000
9000000
Pixels

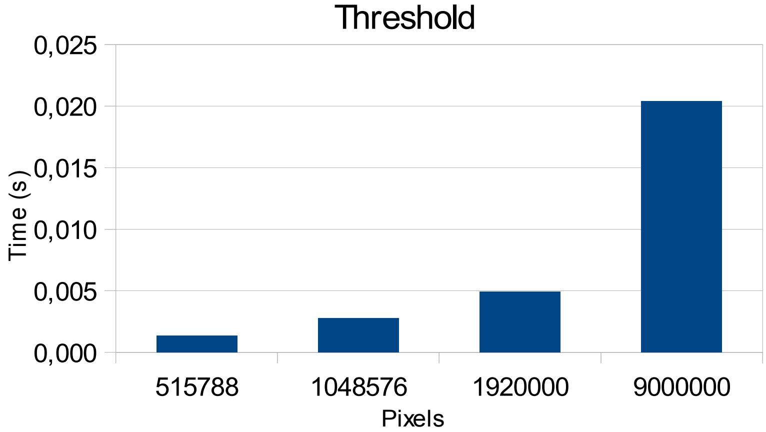

Blur
Time (s)
0
0,5
1
1,5
2
2,5
3
3,5
T=2 I=1 B=50
T=4 I=2 B=50
T=8 I=3 B=50
T=16 I=4 B=50
Threads, Image, Raduis

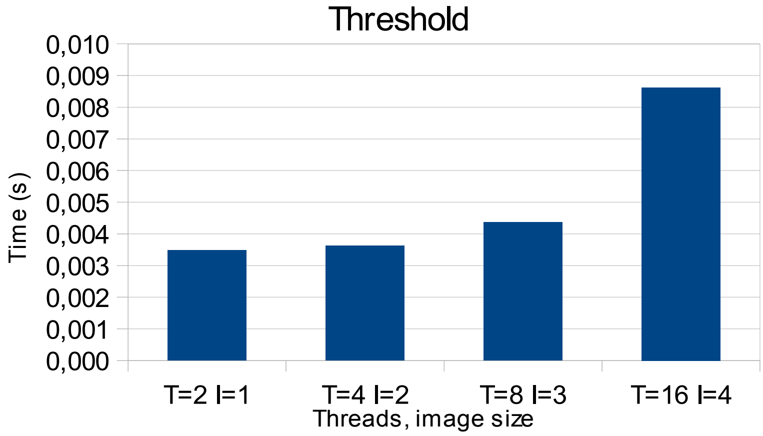

Blur
Time (s)
0
0,5
1
1,5
2
2,5
3
3,5
4
12
25
50
100
Radius

The above results are very good and we see almost no unusual behavior. What can be mentioned is that in the threads diagram it can be seen that 8 threads performs almost as good as 16 threads. Our best guess is that this is because the speedup of having 16 threads working on different parts of the image instead of 8 doesn't make up for the overhead in the communication with 16 threads.

Threshold







These diagrams for threshold are also very good. The time is expected to half when the threads doubles. As the image doubles the time is expected to double, in the last run the image is more than 4 times as big which is also reflected on the execution time. In the last diagram the first 3 should be almost the same since we double the processes and double the image size, the last run has double the processes but more than 4 times as big image and therefore is expected to take about double the time as the others.

Flops

Blur

The blur algorithm have 2 loops where both loop of all pixels and have an inner loop that loops over the radius. There is 18 flops that is performed for all pixels then there is 28 flops in the inner most loop which also loops over the radius. This leads to the following equation.

$$2*(numPixels*18+numPixels*radius*28)$$

Threshold

The threshold filter algorithm has 2 parts. First it sums up all red green and blue values for each pixel to a total sum of the pixel. This is 3 times the number of pixels flops. Then we have to check for each pixel if it's supposed to be black or white which is again a sum of the red green and blue value of the pixel, that is 2 times number of pixels flops. All in all we get the following number of flops.

$$3*numPixels+2*numPixels=5\text{numPixels}$$

Calculation of MFLOPS

Calculation of MFLOPS with an image size of 1048576 pixels and execution time of 0.00277339 seconds.

$$totalFLOPS=5*1920000=5242880$$

$$\frac{5242880\,FLOPS}{0.00277339\,sec}=1890.4\,MFLOP/s$$

Appendix:

Blur

Threads	Time
2	8,296
4	4,03
8	2,464
16	1,864

Image	Time
515788	0,421
1048576	0,794
1920000	1,392
9000000	5,714

Radius	Time
12	0,292
25	0,627
50	1,421
100	3,393

All	Time
T=2 I=1 B=50	1,129
T=4 I=2 B=50	1,32
T=8 I=3 B=50	1,629
T=16 I=4 B=50	3,14

Threshold

Threads	Time
1	0,0258119
2	0,0130509
4	0,00793929
8	0,00436211
16	0,00240389

Image	Time
515788	0,00134643
1048576	0,00277339
1920000	0,0049191
9000000	0,0204225

All	Time
T=2 I=1	0,00348221
T=4 I=2	0,00362514
T=8 I=3	0,00436875
T=16 I=4	0,00862182