ICS-E4020: Week 6 - Challenge

Néstor Castillo García (472081)
nestor.castillogarcia@aalto.fi

May 24, 2015

1 Challenge Submissions

1.1 Description

The previous implementations were compared and results are shown in the same report. No extra assignments were implemented.

1.2 Hardware

The computers had the following specifications: Intel Xeon E3-1230v2, 4 cores, 8 thread, 3,3 GHz, RAM: 16 GB, GPU: Nvidia K2000.

1.3 Median Filter

A median filter was implmented for png images such that for each colour component, the value of each pixel is the median of all pixel values within a sliding window of dimensions (2k+1) x (2k+1).

An eight threaded solution solves the median filtering in an image in less than a quarter of the time of a single threaded solution. It is not an eight due to the fact that only four threads with hyperthreading are being used. The complexity of the median function is O(n). This implementation could be improved to minimise memory access and execute many calculation with minimum data read.

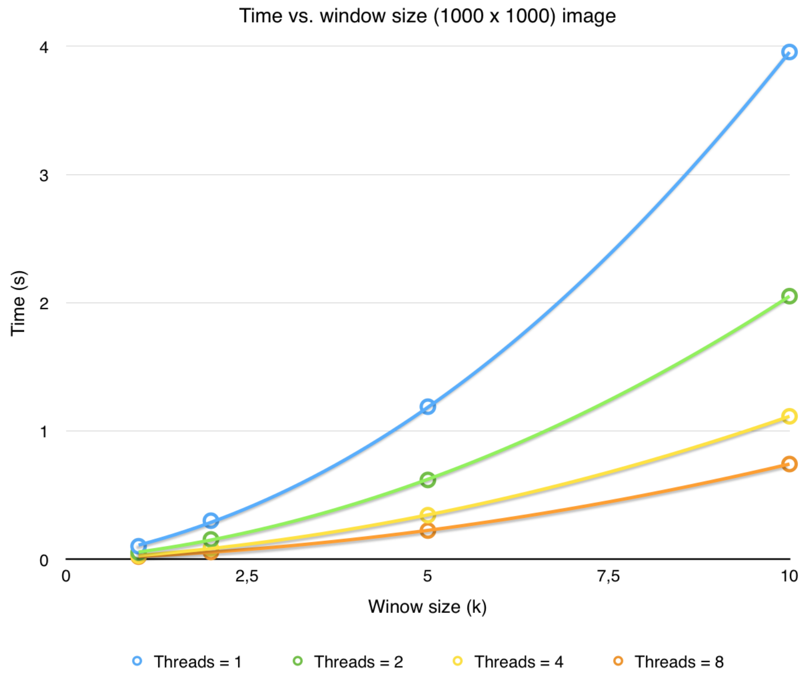

Figure 1: Meedian filtering window sizes time execution with different number of threads

1.4 Correlated Pairs

In this task, m input vectors with n numbers are given. The correlations between all pairs of input vectors is calculated. A red pixel at (i, j) indicates a positive correlation between vectos i and j, and a blue pixel indicates a negative correlation.

The GPU version did the correlated pairs task of a 4000 x 4000 image in less than 1s; 10 times faster than a eight-threaded. The focus of this approach was to maximise the number of arithmetic operation per memory transfer and also to used shared memory per block of threads because it is much more faster than the global memory.

In the CPU the use of vector registers is extremely useful to increase speed of arithmetic operations.

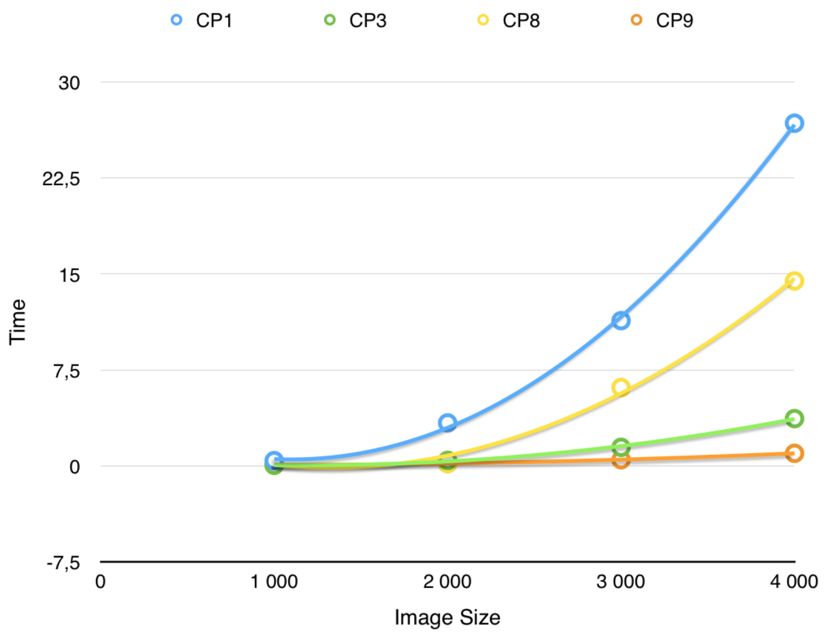

Figure 2: Scalability comparison between single threaded, multithreaded, multithreades vectorized and GPU implementations

The block size should be a careful selection and if shared mamory is used, the memory per block must be calculated to ensure a correct paralel execution according to the hardware.

```
cp      4000     4000     ==10263== NVPROF is profiling process 10263, command: ./cp-benchmark 4000 4000
1.123
==10263== Profiling application: ./cp-benchmark 4000 4000
==10263== Profiling result:
Time(%)       Time     Calls       Avg       Min       Max  Name
 97.50%  772.21ms         1  772.21ms  772.21ms  772.21ms  correlate_call(int, int, int, float const *, float*)
  1.26%  9.9742ms         1  9.9742ms  9.9742ms  9.9742ms  [CUDA memcpy HtoD]
  1.24%  9.7880ms         1  9.7880ms  9.7880ms  9.7880ms  [CUDA memcpy DtoH]
```

Figure 3: Detailed execution time in a 4000 x 4000 image in comparison to cp8 in cp9 memcopy appears to be a more relevant due to the increased efficiency in arithmetic intensity

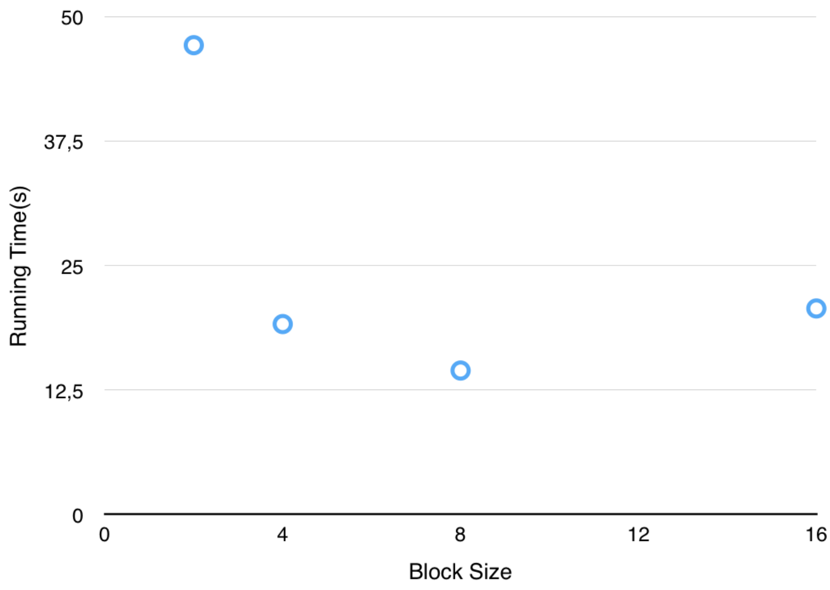

Figure 4: Time vs BlockSize (squared) in a 4000 x 4000 image

1.5 Sorting

In this task a parallel version of merge sort was implemented. In the first step the data is divided into n parts and each thread sorts a single part. Then the parts are merged in parallel by pairs until a single sorted chunk is obtained. The algorithm performs better if the number of threads is a power of two.

As expected, performance increased with the number of threads. The multithreded version was in average 3,8 times faster than the single threaded solution for the large random case values. In the other cases there was a slight increase in speed.

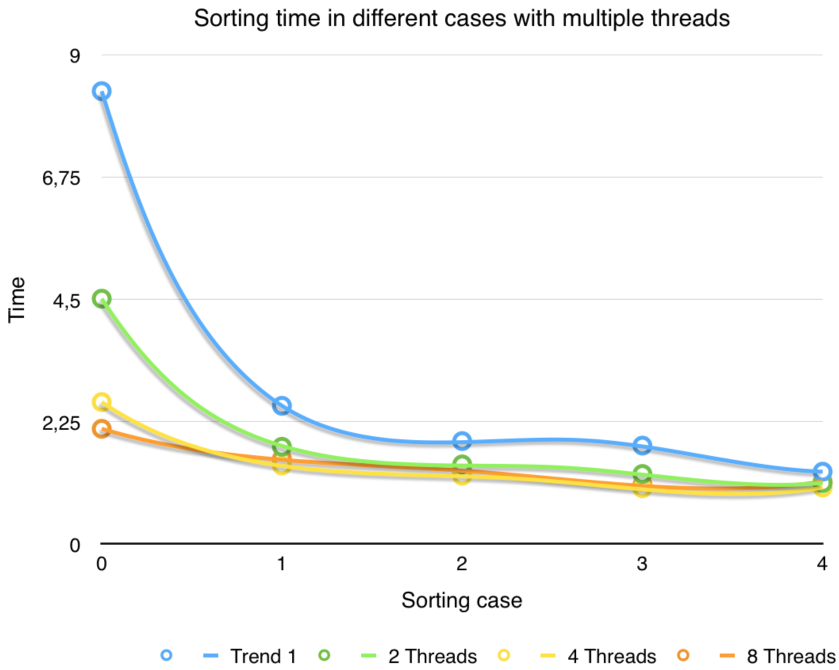

Figure 5: Multithreaded sorting time in cases: 0 = large random elements, 1 = small random elements, 2 = constant input, 3 = increasing values, and 4 = decreasing values.

1.6 Tables

The following table contains the benchmark results for different image sizes and number of threads in median filter.

img. width	img. height	k	Threads = 1	Threads = 2	Threads = 4	Threads = 8
100	100	1	0,003	0,001	0,001	0,001
100	100	2	0,007	0,002	0,002	0,002
100	100	5	0,023	0,006	0,004	0,006
100	100	10	0,038	0,02	0,013	0,017
200	200	1	0,004	0,002	0,002	0,001
200	200	2	0,012	0,007	0,006	0,003
200	200	5	0,048	0,026	0,016	0,011
200	200	10	0,156	0,081	0,044	0,03
500	500	1	0,025	0,013	0,01	0,006
500	500	2	0,075	0,039	0,028	0,017
500	500	5	0,297	0,155	0,088	0,057
500	500	10	0,986	0,511	0,277	0,187
1000	1000	1	0,102	0,053	0,028	0,021
1000	1000	2	0,301	0,155	0,085	0,057
1000	1000	5	1,19	0,619	0,344	0,224
1000	1000	10	3,956	2,052	1,115	0,743
2000	2000	1	0,407	0,212	0,133	0,08
2000	2000	2	1,198	0,621	0,328	0,226
2000	2000	5	4,764	2,479	1,336	0,898
2000	2000	10	15,867	8,229	4,346	2,982

CP3

		OMP_NUM_THREADS=1	OMP_NUM_THREADS=2	OMP_NUM_THREADS=4	OMP_NUM_THREADS=8
1	1	0	0	0	0
1	10	0	0	0	0
1	100	0	0	0	0
1	1000	0	0	0	0
1	2000	0	0	0	0
1	4000	0	0	0	0
10	1	0	0	0	0
10	10	0	0	0	0
10	100	0	0	0	0
10	1000	0	0	0	0
10	2000	0,001	0	0	0
10	4000	0,001	0,001	0	0
100	1	0	0	0	0
100	10	0	0	0	0
100	100	0,001	0,001	0	0
100	1000	0,006	0,003	0,003	0,001
100	2000	0,01	0,006	0,003	0,003
100	4000	0,011	0,013	0,007	0,006
1000	1	0,012	0,007	0,01	0,007
1000	10	0,013	0,009	0,01	0,003
1000	100	0,028	0,014	0,011	0,006
1000	1000	0,185	0,096	0,073	0,043
1000	2000	0,424	0,234	0,155	0,106
1000	4000	1,014	0,566	0,432	0,216
2000	1	0,048	0,026	0,015	0,015
2000	10	0,051	0,028	0,018	0,011
2000	100	0,113	0,057	0,042	0,024
2000	1000	0,853	0,48	0,294	0,184
2000	2000	1,835	0,997	0,57	0,427
2000	4000	4,223	2,272	1,229	0,859
4000	1	0,208	0,108	0,064	0,04
4000	10	0,217	0,117	0,076	0,042
4000	100	0,447	0,225	0,12	0,092
4000	1000	3,758	2,068	1,116	0,742
4000	2000	7,659	4,041	2,259	1,713
4000	4000	17,163	9,066	4,926	3,513

Figure 6: CP3 benchmark result with different number of cores

cp	1	1000	0.002
cp	1	2000	0.038
cp	1	2000	0.004
cp	1	4000	0.045
cp	1	4000	0.007
cp	10	1	0.032
cp	10	1	0.001
cp	10	10	0.032
cp	10	10	0.001
cp	10	100	0.033
cp	10	100	0.001
cp	10	1000	0.035
cp	10	1000	0.002
cp	10	2000	0.041
cp	10	2000	0.004
cp	10	4000	0.049
cp	10	4000	0.008
cp	100	1	0.031
cp	100	1	0.000
cp	100	10	0.033
cp	100	10	0.000
cp	100	100	0.035
cp	100	100	0.001
cp	100	1000	0.046
cp	100	1000	0.004
cp	100	2000	0.038
cp	100	2000	0.005
cp	100	4000	0.047
cp	100	4000	0.009
cp	1000	1	0.034
cp	1000	1	0.002
cp	1000	10	0.037
cp	1000	10	0.002
cp	1000	100	0.040
cp	1000	100	0.004
cp	1000	1000	0.065
cp	1000	1000	0.027
cp	1000	2000	0.088
cp	1000	2000	0.053
cp	1000	4000	0.137
cp	1000	4000	0.105
cp	2000	1	0.041
cp	2000	1	0.006
cp	2000	10	0.043
cp	2000	10	0.006
cp	2000	100	0.045
cp	2000	100	0.011
cp	2000	1000	0.107
cp	2000	1000	0.075
cp	2000	2000	0.184
cp	2000	2000	0.144
cp	2000	4000	0.322
cp	2000	4000	0.290
cp	4000	1	0.060
cp	4000	1	0.019
cp	4000	10	0.053
cp	4000	10	0.020
cp	4000	100	0.067
cp	4000	100	0.036
cp	4000	1000	0.268
cp	4000	1000	0.235
cp	4000	2000	0.483
cp	4000	2000	0.449
cp	4000	4000	0.934
cp	4000	4000	0.901

Figure 7: Benchmark results for GPU correlated pairs implementation CP9

I