ELEC-H-473 Microprocessor architecture
Lab report : SIMD 3

Youri Hubaut, Antoine Vandevenne

May 6, 2017

1 Introduction

For this project, we had to compare the difference of performances while using multi-threading on previously used image-processing algorithms. One consisted in applying a threshold function and the other in applying a min/max filtering. The image was given as a raw file of size 1024 $\times$ 1024 in greyscale (8bit). We had to compare the difference of performance between the plain C version and the ASM one with SSE registers.

2 Code

The code is straight-forward. We simply use the win32 implementation of threads and we hide it through a more abstract point of view. The trick consists to create a functor and give it to the newly create thread in such way that it can call the function with the arguments that we want. We first create a "thread-pool" and we start our threads at the same time using a condition variable.

3 Benchmark

These results are obtained after performing the function 10000 times, time is expressed in seconds:

SIMD 1			
Single threaded			
Debug		Release	
C	ASM	C	ASM
62.967	1.101	12.048	1.091
Multi threaded (4)			
Debug		Release	
C	ASM	C	ASM
31.124	0.624	11.237	0.634

And these results are obtained after performing the function 100 times:

SIMD 2			
Single threaded			
Debug		Release	
C	ASM	C	ASM
19.239	0.092	1.37	0.098
Multi threaded (4)			
Debug		Release	
C	ASM	C	ASM
8.828	0.031	1.653	0.027

The commands used were:

- "gcc -g main.c -o main" for debug

- "gcc -O3 -march=native main.c -o main" for release

For the first SIMD project, we see that using four threads does not reduce the speed by four, but only by a factor of two (except for the C in release version). This may be due to the fact that I use a 2 core + "2 hyper-threads". It means that we are limited by the memory that cannot provide data quick enough.

For the second case, the same phenomenon happens. We can thus conclude that I need a new PC =). Multithreading should be used through a thread pool to avoid cost of thread creation which is humongous in comparison to classical operations. And the benefit that we can obtain is not what we could theoretically expect despite the fact that we have no data clash. Input are read-only, output are write-only and "separated" in memory.