h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Dr. D. Barrientos

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Software optimization

Introduction

In this exercise, we will focus on different software optimization techniques. As we will be changing only the software, the bitstream generated in the previous exercise (in Quartus) can be used to program the FPGA. In principle, also the BSP file will be the same for all the exercises. You will have to change only the software files (in Eclipse).

Profiling

In order to measure the performance of our system, we will use the *ProfileTimer* included in the hardware. As we defined it as the *hal.timestamp_timer* in the BSP configuration in the previous exercise, we can now exploit it for profiling.

In the *main.c* file :

1. Include the following files : *"sys/alt_timestamp.h"* and *"alt_types.h"*
2. Define a few variables to save timestamp values (type : *alt_u32*).
3. Initialize the timer by calling the function *alt_timestamp_start()*.
4. Place calls to the function *alt_timestamp()* before and after the regions you want to monitorize. For example :

```
start_sobel_x = alt_timestamp();
sobel_x(grayscale);
end_sobel_x = alt_timestamp();
```

5. Print the difference (in cycles) between the recorded counter values in the different regions of interest.

Record the number of cycles spent in the functions *conv_grayscale*, *sobel_x*, *sobel_y* and *sobel_threshold*. Reproduce the results presented during the lecture including the number of cycles and time spent by function and for all functions. Generate and compare the results with the compiler options : *-O1*, *-O2* and *-O3*.

Loop unrolling

For these tests, use the *-O0* option in the compiler.

1. Remove the inner loop of *sobel_mac*. What improvement in performance do you reach ?
2. Remove the outer loop of *sobel_mac*. What is the performance now ?

In-lining

Use the *-O0* option in the compiler.

1. Replace the *sobel_x* and *sobel_y* functions by "in-line" versions (without calls to *sobel_mac*). How much improvement in performance do you get ?
2. Merge the filtering functions of the Sobel algorithm within a single *sobel_complete* function. Do you get any further improvement ? Why ?

Code improvement

- Could you change the code in *sobel.c* to improve further the processing time ? What would you change ?
- What about *grayscale.c* ?

At the end of these optimizations you may get about 2230 CPU cycles/pixel in *-O0* and about in 385 cycles/pixel *-O3*. How many do you get ?