# Project 1 - Image filter

## DV1463 – Performance Optimization

Emiliano Casalicchio, Sai P. Josyula

# The Image filter project

This lab is about the performance evaluation and optimization of a short program that blurs an eight-bit gray scale image. The code in Annex 1 assumes that you provide it with a file containing a 251x256 gray scale photographic image (RAW)[1],[2].

The data structure used by the program is as follows:

```
Image: array [0..250, 0..255] of byte;
```

Each byte contains a value in the range 0..255 with zero denoting black, 255 representing white, and the other values representing even shades of gray between these two extremes.

The blurring algorithm averages a pixel with its eight closest neighbors. A single blur operation applies this average to all interior pixels of an image (that is, it does not apply to the pixels on the boundary of the image because they do not have the same number of neighbors as the other pixels). The Pascal program implements the blurring algorithm and lets the user specify the amount of blurring (by looping through the algorithm the number of times the user specifies).

The Pascal program[3] in Annex 1 compiles with Free Pascal[4]. The compiler is available for windows, Linux and OSX but I strongly suggest you to work with Linux. Under Linux you can also use the GDB[5] debugger, very useful when you develop assembly programs and performance profiling tools.

To complete the project, you should accomplish the following tasks and write a final report.

## Task 1 – pascal and performance evaluation

The first task is

- To compile and to run the pascal program in Annex 1

[1] An image compliant with the input requirement can be downloaded from the Project page on itslearnig. If you don't like it, feel free to generate your own image.

[2] See Annex 3 for how to visualize a RAW image

[3] Tutorials about pascal programming are available here:

- https://www.tutorialspoint.com/pascal/index.htm
- http://www.freepascal.org/docs-html/3.0.0/ref/ref.html

[4] http://www.freepascal.org/

[5] https://www.gnu.org/software/gdb/

- To measure the execution time of the program, i.e. the real, user and sys time. The execution time, hereafter referred also as performance, can be measured using the linux's command `time` with the resolution of milliseconds (`bash` version).

You should answer the following questions:

- What are the performance of the pascal code on your platform?
- What is the impact of the blur factor (number of iterations) on the execution time of the image filter program? Do the performance evaluation for a significant number of blur factor values, use plots to represent the results, and provide your comments.

## Task 2 – C, performance comparison and optimization

The second task is to implement a C version of the image filter algorithm, to compile it with the `gcc` compiler and to measure again the performance. Then, compare the performance of the C and Pascal version. You should answer the following questions:

- Does the C version outperform the pascal version? Do the comparison for the same set of blur factor's values used in Task 1. Use plots to represent the results and provide comments.
- Try to compile the C program with the options –o, -o2 and –o3. Do you find any difference in the execution time and or memory utilization? (To measure the memory usage you can use the valgrind[6] / massif[7] tool). Provide a discussion of performance results/comparison (represent the results with plots)

## Task 3 – assembler and performance comparison

The third task is to write an assembly version of the image filter, to measure and to compare the performances and compare again with the previous two implementations. Annex 2 shows an example of not optimized assembly code. It is a crude translation from pascal code directly into assembly language. The assembly syntax used is the Intel assembly[8]. Remember that you will use the `gcc` compiler and, therefore, you should use the AT&T syntax.

[6] http://valgrind.org/

[7] http://valgrind.org/docs/manual/ms-manual.html

[8] A comparison between intel syntax and AT&T syntax can be found here http://www.imada.sdu.dk/Courses/DM18/Litteratur/IntelnATT.htm

First, you should run the not optimized version of the assembly program and measure the performance (execution time and memory usage) for different values of the blur factor. Then you should answer the following questions:

- What is the faster image filter implementation? Assembly, C or Pascal? Do the comparison for the blur factors used in Task 1 and 2. Provide a discussion of the results and represent results using plots.
- What is the difference among the three different implementations in term of memory usage? Does the blur factor impact the memory usage?

# Task 4 – assembler, optimization and performance comparison

Finally, you should try to optimize the code using the following techniques:

1. Use `movsd` instruction rather than a loop to copy data from `DataOut` back to `DataIn`.
2. Use a `repeat-until` style for all loops.
3. Unroll the innermost two loops

You can apply all the techniques in a single step, that means you are not required to produce three different versions of the program.

Optimization 1 refers to the possibility to copy a block of memory words with a single instruction rather than implementing the memory transfer with a loop. Have a look at: the instruction **movsd** that moves the memory location addressed by `DS:SI` into the memory location addressed by `ES:DI;` the `rep` modifier that repeat the execution of an instruction for the number of time specified in `cx`): e.g. the following code execute the `movsd` instruction for 256 times

```
…
mov cx 255
rep movsd
…
```

Optimization 2 is related to the implementation of the loops using the `repeat-until` style. In the `repeat-until` style you first execute the action, then you check the exit conditions; in the `for-loop` style you do the opposite. The `repeat-until` style provides many advantages when used in assembly coding style. Among them, it allows also to reduce to a single loop the two DataIn to DataOut and DataOut to DataIn copy loops, that are:

```
for i := 0 to 250 do
        for j := 0 to 255 do
                dataout^ [i][j] := datain^ [i][j];
```

and

```
for i := 0 to 250 do
        for j := 0 to 255 do
                datain^ [i][j] := dataout^ [i][j];
```

Of course, that requires to change the logic in the management of `datain` and `dataout` variables.

Optimization 3 is the most useful. The innermost loops

```
for k := -1 to 1 do
        for l := -1 to 1 do
                sum := sum + datain^ [i+k][j+l];
```

are repeated for 249x251 times. Unrolling the loops means to execute in a sequential order the nine instructions that are executed by the for-loops. Of course, you cannot use the index `i` and `k` but their values should be made explicit by using constants. The code will occupy more space (line of code) but loop unrolling allows to save the time needed for: comparison, jump and loop counter initialization and increment.

You should answer the following questions:

- Why the use of a `repeat-until` style should allow to achieve better performances? (Apart from the possibility above mentioned)
- Why the loop unrolling techniques should allow to achieve better performances? (Apart from the possibility above mentioned)
- What are the performance of the assembly-optimized version with respect to the performance of the assembly-not-optimized, C, C-optimized and pascal version? (do the comparison for the same set of blur factors used in the previous tasks, provide comments, and report the results using plots).

# Final Report

At the end of the lab you have to submit a report on itslearning. The lab is evaluated 1.5 hp. If the report is not satisfactory you will be requested to submit an improved version.

The final report should be minimum 2000 and maximum 5000 words long (excluded plots). The final report must be writen in english and sbmitted in pdf format. The final report should contain all the sections described in what follow.

The code must be duly commented and provided in separate files.

## Task 1.

In this section you should provide comments on your experience in running and compiling the pascal code and on the performance evaluation of the pascal code. Do the performance evaluation for a significant number of blur factor values and represent the performance metrics with a plot.

Moreover, In this section you should provide an answer to all the question specified in the above section Task 1 – pascal and performance evaluation. Don't forget to represent performance results using plots.

## Task 2

This section should contain:

- Comments on your experience in developing, compiling and running the C code. The C code must be provided and duly commented.
- Comments on the performance comparison between the C version and the Pascal version.
- Comments on the results of the optimization of the compiled code.

Moreover, In this section you should provide an answer to all the question specified in the above section Task 2 – C, performance comparison and optimization. Don't forget to represent performance results using plots.

## Task 3

This section should contain

- Comments on your experience in developing, assembling and running the assembly code. The assembly code must be provided and duly commented.
- Comments on the performance comparison among the three versions of the image filter program.

Moreover, In this section you should provide an answer to all the question specified in the above section Task 3 – assembler and performance comparison. Don't forget to represent performance results using plots.

### Task 4

This section should contain

- Comments on your assembly optimization choices and on their implementation. (Comment both the design and on the implementation of the optimization).
- Comments on the performance evaluation of the Assembly code.
- Comments on the performance comparison with the other implementations.

Moreover, In this section you should provide an answer to all the question specified in the above section Task 4 – assembler, optimization and performance comparison. Don't forget to represent performance results using plots.

# Annex 1 – Pascal code

The code is available in the file `imagefilter.pas`

# Annex 2 – Assembly code

The code is available in the file `imagefilter.s`

# Annex 3 – Image visualization

The input image is in the file `roller1.raw`

To visualize the image, you can use any image processing application that support RAW or you can use http://rawpixels.net/ with the following options:

```
Predefined format: Grayscale 8bit
Pixel Format: Grayscale
Dimensions: 251x256
```