Integrated Software Project
Calendar Tool Application

Imperiali Michele, Lorent Nicolas, Vanosmael Laurent and Mormont Romain
mjimperiali@student.ulg.ac.be, nicolas.lorent@student.ulg.ac.be,
laurent.vanosmael@student.ulg.ac.be, r.mormont@student.ulg.ac.be

Faculty of Applied Sciences – University of Liège – Belgium

Introduction

The **Calendar Tool Application** is a web-based scheduling application. On the one hand, it allows students to manage their personal and educational agenda. On the other hand, it provides to professors and faculty staff members a global view of students' schedule and allows them to schedule courses, projects, etc, in an optimal way. The application is available through a web interface and a mobile application.

Title

Block contnte :

Which part to optimize?

It was clear for us that we had to put our focus in optimizing the operation that was performed the most often. Obviously, the initialization is performed only once. The segmentation is done for each frame of the video, whereas, for each pixels in the background model, there is a probability to update it. Therefore, we decided to focus on the segmentation.

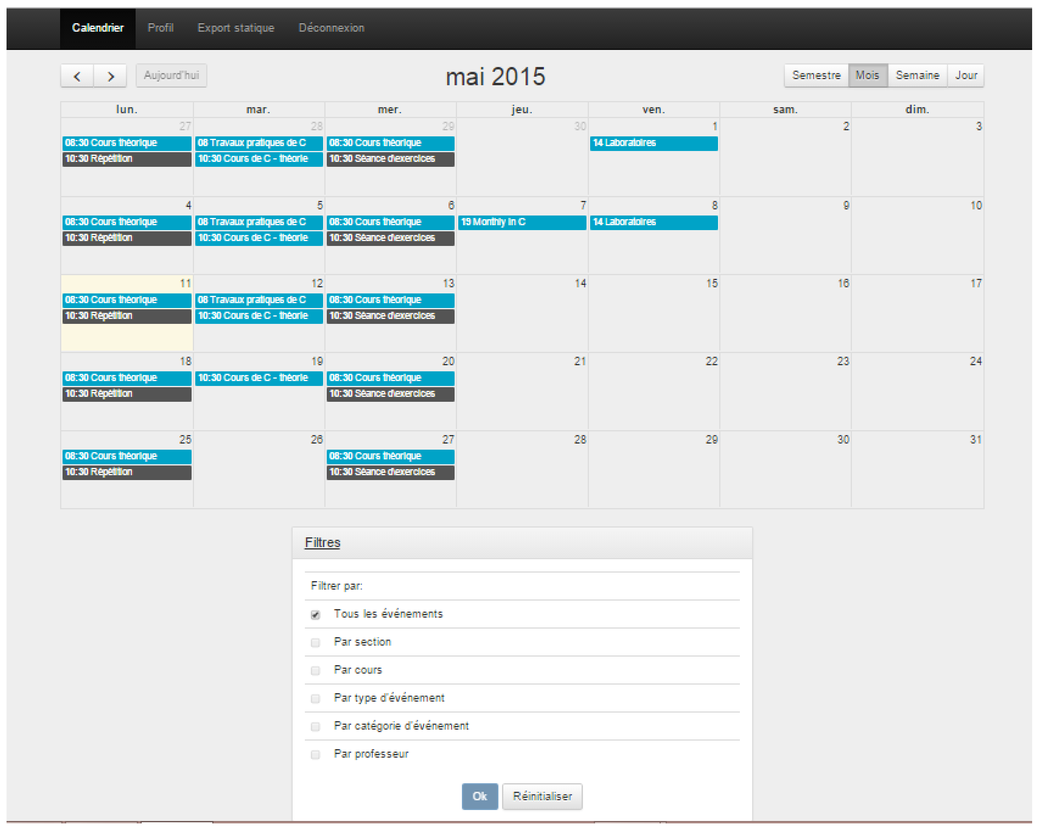
Sequential implementation

Without getting into the details, an optimization performed on the sequential version consists of maintaining the background model in two buffers. The first one contains the values that matched recently and the second contains the others. This allows to reduce the average complexity of the segmentation.

Vectorization

Our vectorization is targeted to speed up the segmentation. It performs the distance check for all pixels of a model at once. Since this distance check is performed in one-go, we decided to use one buffer instead of two as in the sequential version. This parallelism is obtained thanks to the NEON C intrinsics which exposes a lot of useful operations on and between vectors. Nevertheless, we didn't find many aggregation operation and had to implement ourself an accumulation (see the `accumulate` step in the Figure 1).

Figure 1: Steps of the vectorized implementation



Results

We have used the performances of the sequential implementation to assess our vectorization. The execution time of the segmentation and update steps are given in Figure . The chart highlights the quickness of the sequential version compared to our implementation. The former is actually two times faster than the latter and it even reaches an impressive framerate of 273 on one of the test video.

Figure 2: Execution time of the segmentation and update steps (μs)

We think that the slowness of our implementation is due the combination of the following factors :

(1) The accumulate operation shown on Figure 1 is quite expensive (it compiles in 10 assembly instructions).
(2) Our implementation uses every pixel of a model at each iteration (in the segmentation loop) while the sequential one first checks two pixels and only tests the others if necessary.
(3) On the test sequences, most of the pixels were in the background.

Improvements

Unfortunatly, the vectorized version is less efficient than the sequential one but there is some room for improvements.

A **first idea** would be to *find another way to aggregate the vector's lanes* (instead of accumulate). Indeed, reducing the number of assembly instructions associated with this step would drastically improve the algorithm as half of the segmentation loop instructions are dedicated to the aggregation.

Another idea would be to *keep the idea behind the sequential implementation while keeping the vectorized features* : keeping the most recent background pixel values in a 2-elements buffer and the rest of them in a 16-elements one. The buffers would still be vectors as the NEON architecture provides supports for such sizes. Moreover, a heavy aggregation wouldn't be needed anymore for the small buffer.

Conclusion