Impact of parallelization technologies on an optical flow-based motion interpolation algorithm

Juan Sebastián Rodríguez Castellanos
Departamento de Sistemas e Industrial
Universidad Nacional de Colombia
Bogotá - Colombia
juarodriguezc@unal.edu.co
GitHub account

***Abstract*—One of the applications of video processing is the Motion interpolation or the motion-compensated frame interpolation (MCFI), in these algorithms a pair of frames are used to generate an intermediate frame. This paper presents the implementation of a frame interpolation algorithm based on optical flow using different parallelization technologies (CPU, GPU, MPI). The algorithm used is presented in the document. Since the calculation of the Optical Flow is a process that requires a high computational capacity but can be executed in parallel, the impact of these technologies on the execution time will be analyzed. Different tests were performed, resulting in a speedup greater than 3 using CPU and above 34 using GPU with the implementation in CUDA. However, in all cases the execution times were considerably high, allowing only the CUDA implementation to interpolate an average of 3 frames per second for high video resolutions.**

Keywords: Motion interpolation, Optical flow, CPU, GPU, Speedup.

I. Introduction

Motion interpolation is a form of video processing with numerous applications, this type of algorithm is widely used by different technology companies to provide the effect of "smoothing" on different audiovisual content. The goal is to create an intermediate frame or number of frames between existing frames [1]. There are two main types of motion interpolation algorithms, in the first type all frames to be interpolated are known, in the second type only the current and previous frames are known.

There are many uses for this type of algorithm. In most current TVs there is software that produces the effect of "smoothing" on video, to achieve this effect the frame rate is increased to 60 or more Frames, depending on the frequency of the screen. Another of the main uses of video interpolation is in Virtual Reality (VR), being used by companies such as Oculus, Microsoft and Steam for their VR headsets [1].

In this paper an optical flow-based motion interpolation algorithm is presented which can interpolate frames using only the current and previous frames. To generate the intermediate frame, the Optical Flow is calculated, then linear interpolation is applied using the information obtained from the Optical Flow and finally a Gaussian Blur is applied to the areas with the highest motion.

The calculation of the Optical Flow is the most important task for the algorithm developed, the objective of the optical flow is to predict the pixel-level motion between video frames [2]. Classical approaches have been superseded by learning-based methods [2], however, for the development of the algorithm, a brute force approach is used to calculate the motion vectors.

II. Related work

1) Optical Flow:
The optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or camera [3]. It is a 2D vector where each vector is a displacement vector showing the movement of points from first frame to second [3]. To calculate this vector it is necessary to use the intensity or luminance of each pixel [4]. Consider a pixel $I(x, y, t)$ in the first frame, it moves by distance (dx, dy) in the next frame after dt time, so:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{1}$$

An example of the optical flow calculation is shown in Figure 1.

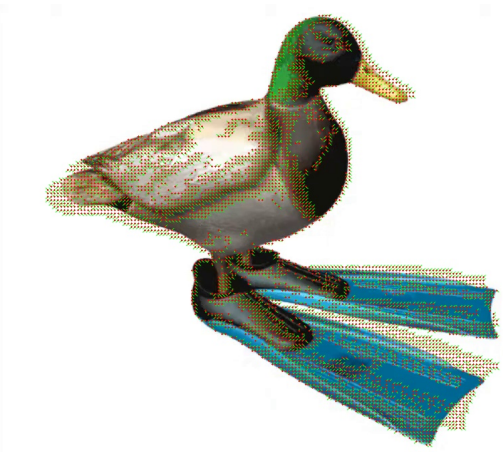

Fig. 1: Example of the Optical Flow

2) Luminance:
To calculate the Optical Flow it is necessary to use the intensity of each pixel. Intensity or luminance is a measure to describe the perceived brightness of a color [5], this value can be calculated using the following formula:

$$L = 0.2987R + 0.5870G + 0.1140B \quad (2)$$

Values for R, G and B must be between 0 and 1.

3) Gaussian Blur:

4) Linear Interpolation:

III. Algorithm Designed

1) Motion interpolation:

2) Parallel algorithm:

IV. Experiments

The main criteria to evaluate the algorithm designed

V. Results

1) OpenMP: asdasdasd

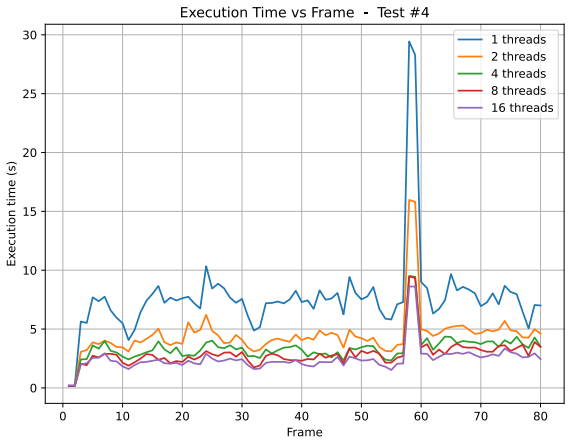

Fig. 2: Execution time by frame - OpenMP

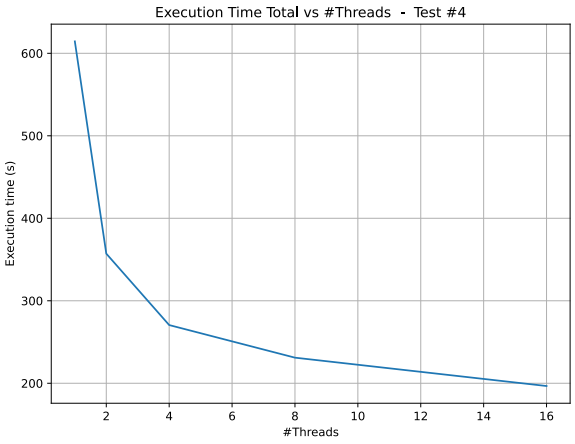

Fig. 3: Total Execution time vs #Threads - OpenMP

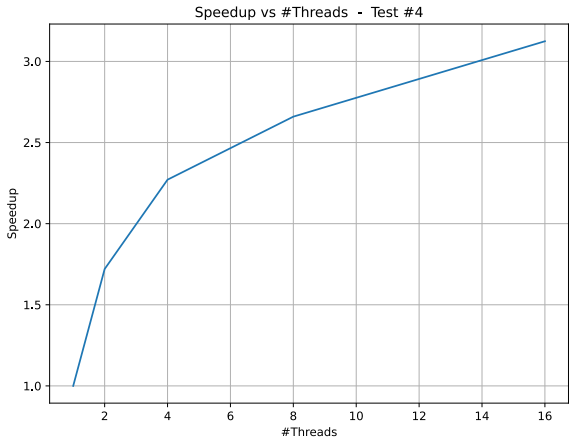

Fig. 4: Speedup vs #Threads - OpenMP

2) CUDA: asdasdasdasd

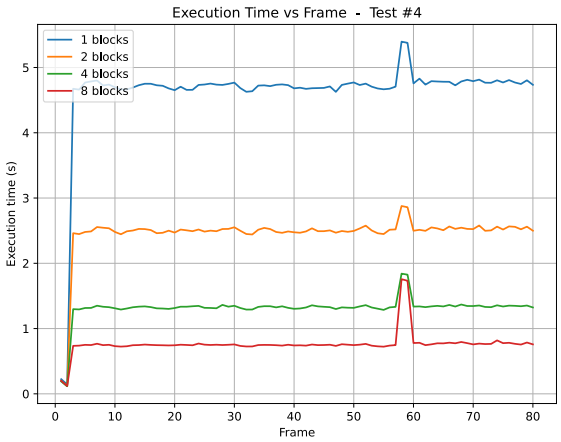

Fig. 5: Execution time by frame - CUDA

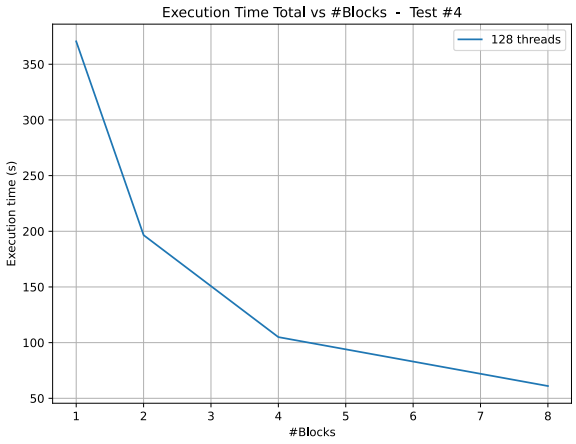

Fig. 6: Total Execution time vs #Threads - CUDA

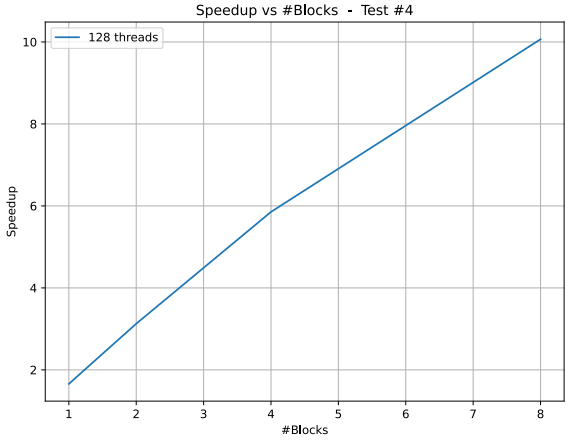

Fig. 7: Total Execution time vs #Threads - CUDA

VI. Conclusion

1) The Github repository is:
https://github.com/juarodriguezc/Sistemas-distribuidos

References

1) https://iopscience.iop.org/article/10.1088/1742-6596/1488/1/012024/pdf
2) https://arxiv.org/pdf/2204.08442v1.pdf
3) https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html
4) https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT12/node4.html
5) https://www.101computing.net/colour-luminance-and-contrast-ratio/