SPM@Unipi-2016/2017

Project Report [Video Filtering]

By: Ahmad Alleboudy

Code repository:

https://github.com/alleboudy/spm

Problem being Solved
Applying either Sobel filter or a contrast stretching filter on a given video, implemented in different fashions.

Programming frameworks and tools used
FastFlow, OpenCV library, CMake

Architecture used to run the experiments
An AMD machine that has 24 real cores, no hyperthreading

What has been implemented in the project

1- A pipeline using fastflow's ordered farm [parallel2.cpp]
Utilizing an Emitter that reads the frames from the input video one by one and sending them out to the farm workers for processing where each worker processes a whole frame at a time and sends the result to the collector which handles writing the resulting frames to disk.
2- A pipeline using fastflow's Parallel For [parallel3.cpp]
A naïve implementation utilizing a fastflow parallel for loop skeleton over the frame's rows assigning each row in a frame to a worker.
3- Same filtering implemented in a sequential manner [sequential.cpp]
Where each frame is read, processed and saved to the disk within the main thread.
4- Same filtering implemented using threads [threaded.cpp]
Utilizing threads to process the frames, initiating a number of threads [given by the user] each one with a frame then joining the threads in the order they were invoked in then writing their results which were kept in a static array to the disk maintaining the order the frames were provided in.
5- Same filtering implemented using C++11 future and promise [future.cpp]
6- Others, but not reported due to obvious overhead [combining the parallel farm together with a parallel for inside the workers of the farm, allowing each worker of the farm to have numberOfCores-(NumberOfWorkers+2)/NumberOfWorkers cores in a parallel for that loops through the image pixels to carry out the worker's job], this method always utilized the 24 cores and showed too much overhead, so, it was omitted.

Included, as well, is a CMakeLists.txt file for building the source code along with the already built binaries under spm/build

Also, test.sh for running the binaries for different number of cores and reporting the time taken
And, clean.sh for cleaning the build folder removing *.avi files, cmake files and built binaries

Possible major FastFlow pipeline design choices

1- A fast flow Ordered Farm with a custom emitter and collector [would not have to worry about the frames ordering as it is already handled by the ordered farm]
2- A fast flow “Parallel for” that runs over the frame pixels
3- A combination [omitted due to obvious overhead]

How to compile the code

Will need to have OpenCV installed and the fastflow root folder present on the machine in the same directory where the project folder is residing. [please feel free to edit the CMakeLists.txt for better convenience]

Using CMake , navigate to the project directory spm

mkdir build

cd build

cmake ..

make

To run the compiled Binaries

For the sequential

./sequential path/to/input/video/file.mp4 path/to/output/video/file.mp4 sobel

For the others

./other path/to/input/video/file.mp4 path/to/output/video/file.mp4 NumberOfWorkers sobel

Where:

path/to/input/video/file.mp4 is where the input video resides on the desk

path/to/output/video/file.avi is where we wish to have the output video

NumberOfWorkers is the number of threads to use

sobel is for applying the sobel filter, other option is **stretch** for applying contrast stretching

Under spm/build is a simple small test video chaplin.mp4, the implementations work with colored and long videos as well

A small brief about the filters implemented:

Sobel Filter:
Is used to enhance the edges in an image which benefits computer vision and image processing applications where edge detection is required
https://en.wikipedia.org/wiki/Sobel_operator

Contrast Stretching:
Is used to limit the pixel intensities in an image between a given range [for simplicity here using min=0 and max =255, which forces the minimum intensity in the image to be 0 and the max to be 255 while stretching the intensities in between in the new range [0,255]]

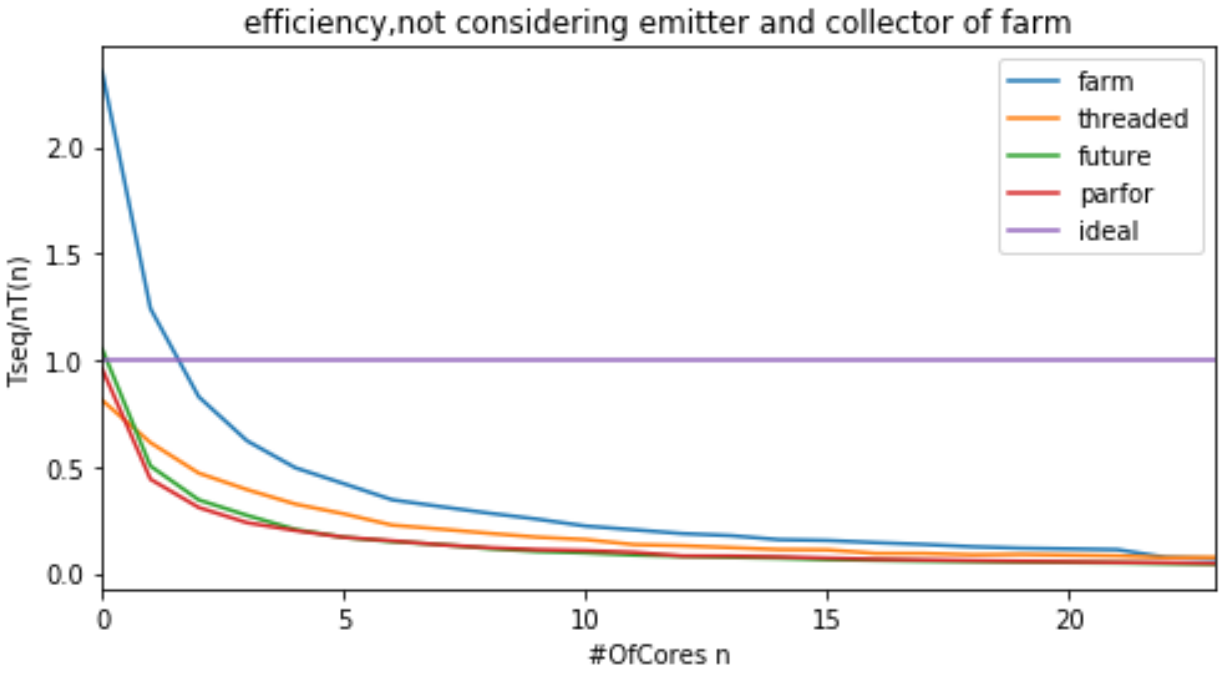
https://en.wikipedia.org/wiki/Normalization_(image_processing)

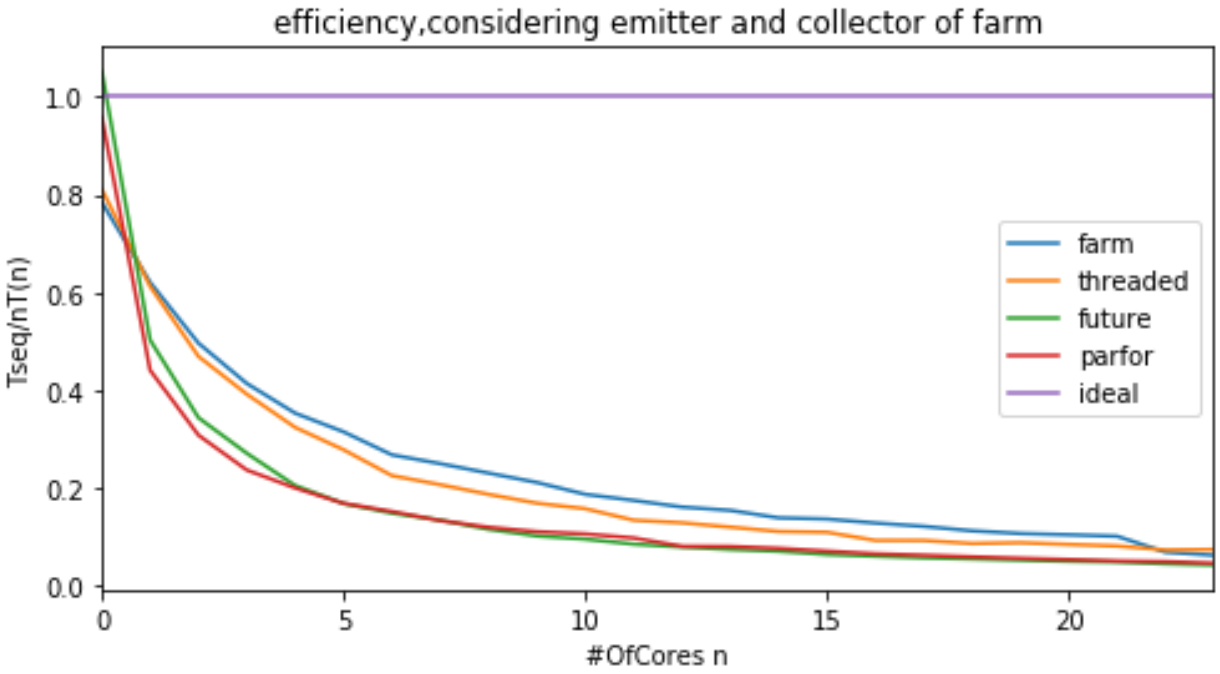
Performance Measures

[The used test video is spm/build/chaplin.mp4 of 190 frames with sobel filter]

1 - Efficiency,

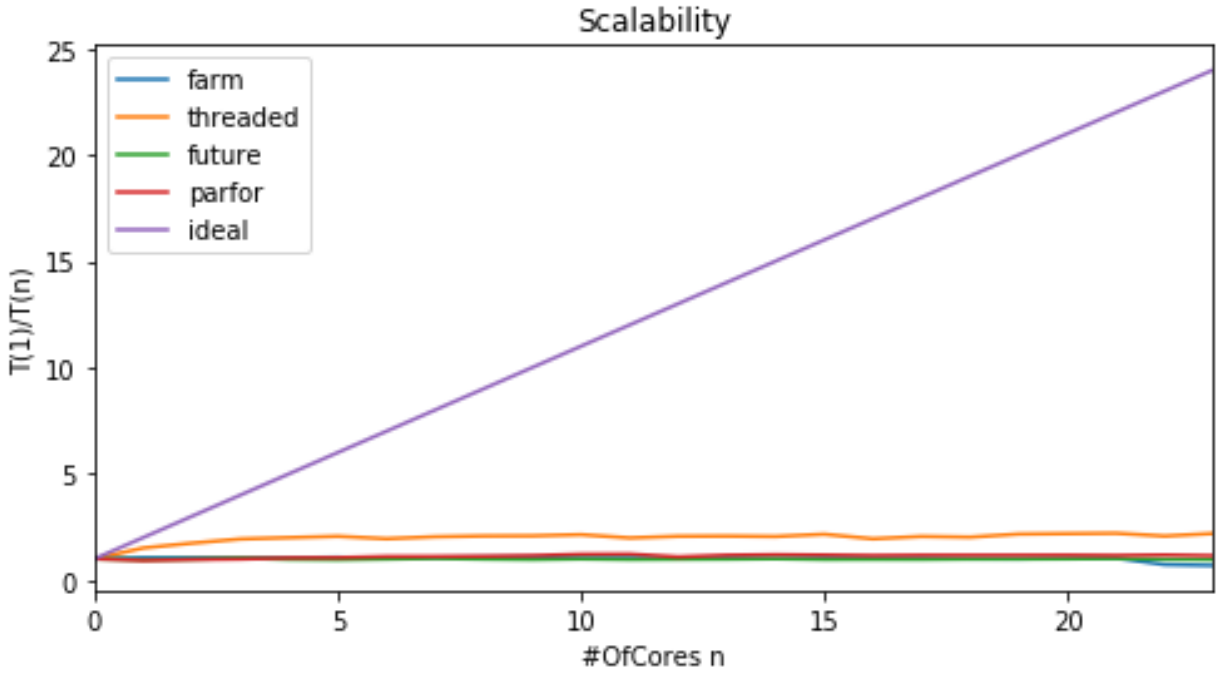
measuring how good is the implementation benefiting from the resources used by computing the ideal time for the resources usage Tseq/n divided by the parallel time Tpar(n)



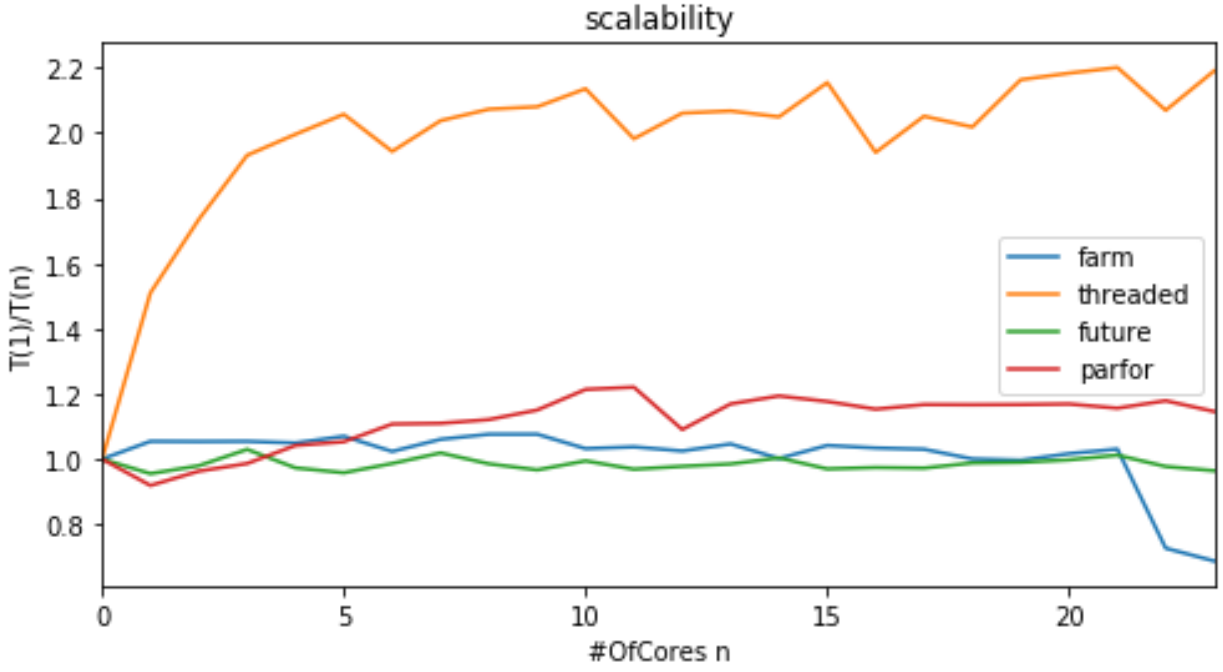


2 – Scalability,

Is Showing the ratio between the implementation's executing time with a single resource vs multiple resources [please note here the farm's emitter and collector are ignored from the calculations for simplicity]



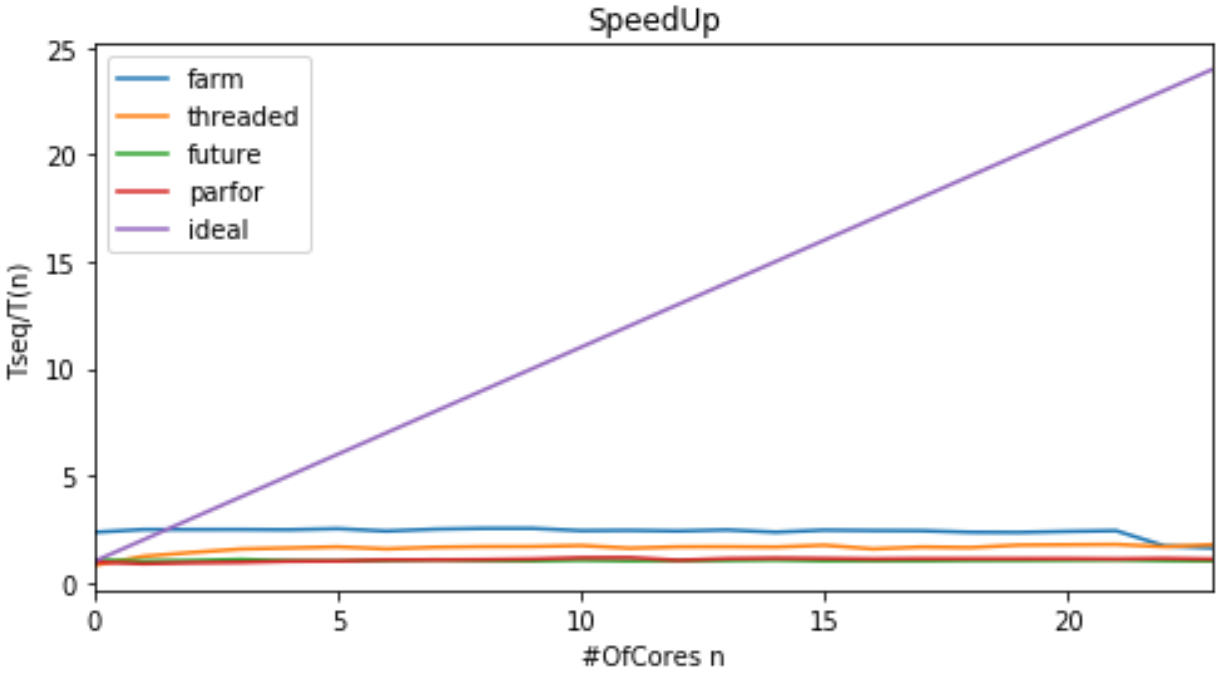
Omitting ideal for better comparison between the different implementations



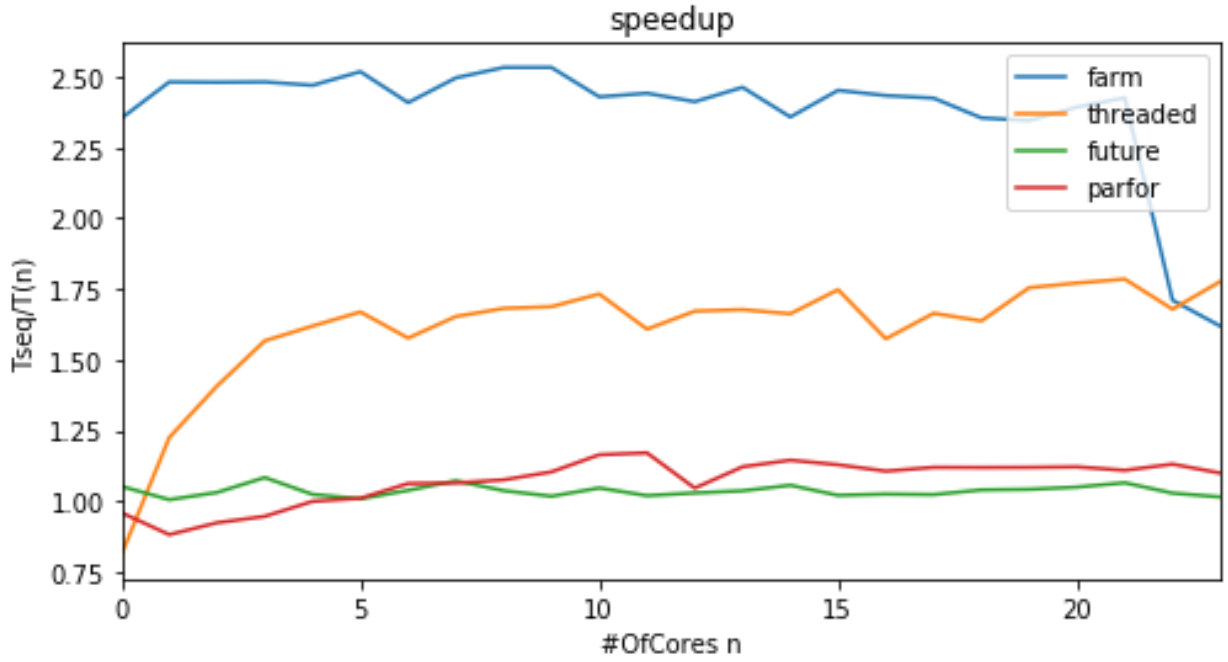
For the scalability, it seems the farm implementation is not showing much benefit from using more cores to process the frames.

3 – Speed Up,

Showing the gain in terms of speed when utilizing more resources



Omitting the ideal for better comparison between the implementations



We can easily notice a major drop in the farm implementation's performance measures when the number of cores exceeds 22 [since two are already dedicated to the emitter and collector]

Regarding the difference between the ideal case and the actual

My reasoning would be that it is due to the bottleneck when saving the frames to disk, meaning, whatever the speed we can achieve from dividing the frames between workers, we are limited by the inevitable sequential parts of the pipeline (Amdahl's Law).

[one frame write operation= ~12ms, x190 frames in chaplin.mp4 video = 2280ms which is around the time reported by the farm solution]

In a previous code version I tried to provide an output buffer to keep the frames in then flush a batch of frames together instead of writing them one by one to the disk, but unfortunately there is no batch write frames to disk in OpenCV.

And parallelizing the saving to disk part will definitely cause a race condition between the different writers trying to access and use the videowriter object at the same time which may no longer guarantee the frames order.

References

Unipi SPM lecture notes [2016/2017]

http://didawiki.cli.di.unipi.it/doku.php/magistraleinformaticanetworking/spm/spm1617lessons

OpenCV documentation

http://docs.opencv.org/

FastFlow tutorials and examples

http://calvados.di.unipi.it/dokuwiki/doku.php/ffnamespace:tutorial

Wikipedia

https://en.wikipedia.org/wiki/Sobel_operator

https://en.wikipedia.org/wiki/Normalization_(image_processing)