FFMPEG GPU overlay

What can and can’t you do.

Regionales Rechenzentrum | Lira Del Canto, Maximiliano |

Index

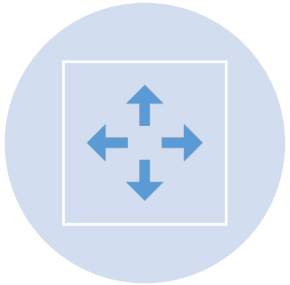
- What could do with the GPUs in FFMPEG
- The GPU overlay
  - What is the overlay filter
  - Tests
  - Advantages/Disadvantages over the traditional method
- How to implement in Opencast

What process can you do with GPU accelerated FFMPEG?

Resize

Crop

Re-encode



Transpose

We can’t…

- Apply filters (color filter, noise filter, etc)
- Generate noise, backgrounds, etc.

BUT NOW

¡We can use the Overlay filter!

Universität zu Köln

What is the overlay filter?

The overlay filter allows to put one video or image on top another.

Is a very useful filter when is needed to make a Picture-in-Picture video or to create a side-by-side video. Also useful when is needed to insert images or graphics on top a video.

Universität zu Köln

Tests: Side-by-Side video

- Opencast uses the overlay filter for the video editor
- The default opencast FFMPEG command that is used is:

```
ffmpeg \
-i $video_1 \
-i $video_2 \
-filter_complex \
'[0:v]scale=640:360,pad=1280:400:640:0:0x000000FF[lower];
[1:v]scale=640:360[upper];
[lower][upper]overlay=0:0[out];
[0:a][1:a]amix=inputs=2[aout]' \
-map "[out]" -map "[aout]" \
-c:v libx264 -preset veryfast -crf 23 -profile:v baseline \
-pix_fmt yuv420p -tune film -movflags faststart \
-c:a aac -ar 22050 -ab 64k "${out_video}_sw.mp4"
```

Tests: Side-by-Side video

- The command using GPU overlay is:

```
ffmpeg \
-hwaccel cuda -hwaccel_output_format cuda -i $video_1 \
-hwaccel cuda -hwaccel_output_format cuda -i $video_2 \
-filter_complex \
'[0:v]scale_npp='"640"':-2:format=yuv420p,hwdownload,
pad=w=2*iw:h=ih:x=0:y=0,hwupload_cuda,scale_npp=format=nv12[base];
[1:v]scale_npp=640:-2:format=nv12[overlay_video];
[base][overlay_video]overlay_cuda=x='"640"':y=0:repeatlast=false[out];
[0:a][1:a]amix=inputs=2[aout]' -map "[out]" -map "[aout]" \
-c:v h264_nvenc -preset fast -movflags faststart \
-c:a aac -ar 22050 -ab 64k "${out_video}_mixed.mp4"
```

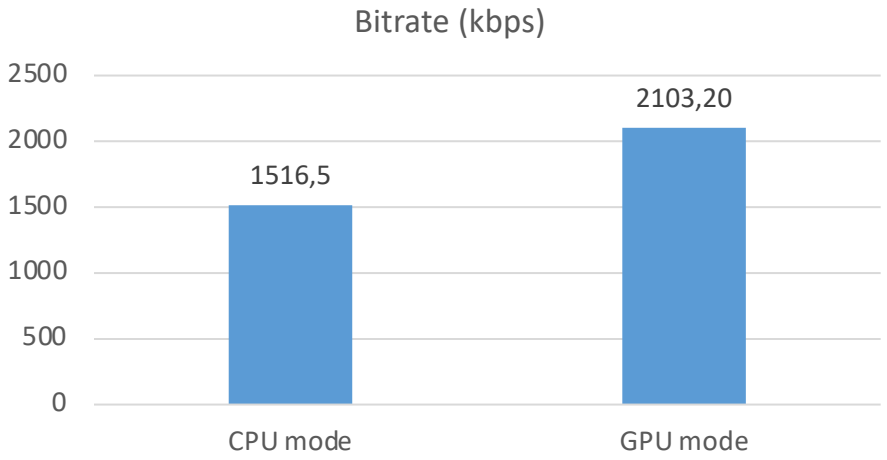
This command uses the CPU to create a “padding” for the video to insert it and create SBS.

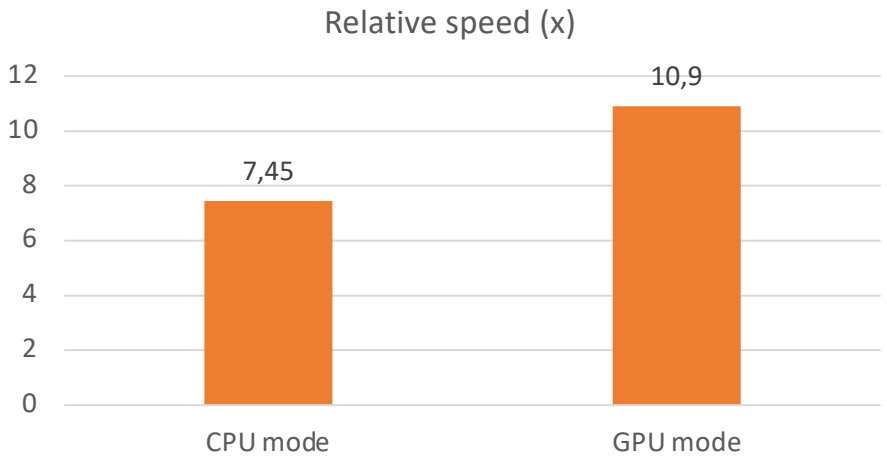
Tests: Side-by-Side video

CPU SBS video

GPU SBS video

Tests: Side-by-Side video





- The GPU mode with the fast profile, is less efficient than the CPU by 38%.
- The GPU is faster, but not so fast as the CPU counterpart.

Tests: Picture-in-Picture

- GPU (No background image):

```
ffmpeg -y -loglevel info \
-init_hw_device cuda=cuda -filter_hw_device cuda \
-hwaccel_output_format cuda -hwaccel cuda -i $video_1 \
-hwaccel_output_format cuda -hwaccel cuda -i $video_2 \
-i $input_logo \
-filter_complex \
'[0:v]scale_npp=1280:720:format=yuv420p,hwdownload,pad=w=iw+640:h=ih+360:x=20:y=180
:color=#DBE4ED,hwupload_cuda,scale_npp=format=nv12[beamer];
[1:v]scale_npp=512:-2:format=nv12[presenter];
[beamer][presenter]overlay_cuda=x=1360:y=40:repeatlast=false[merge];
[2:v]format=nv12,hwupload_cuda[logo];
[merge][logo]overlay_cuda=x=20:y=20:shortest=false' \
-c:a aac -ar 22050 -ab 64k -c:v h264_nvenc -preset fast -movflags faststart
$out_video
```

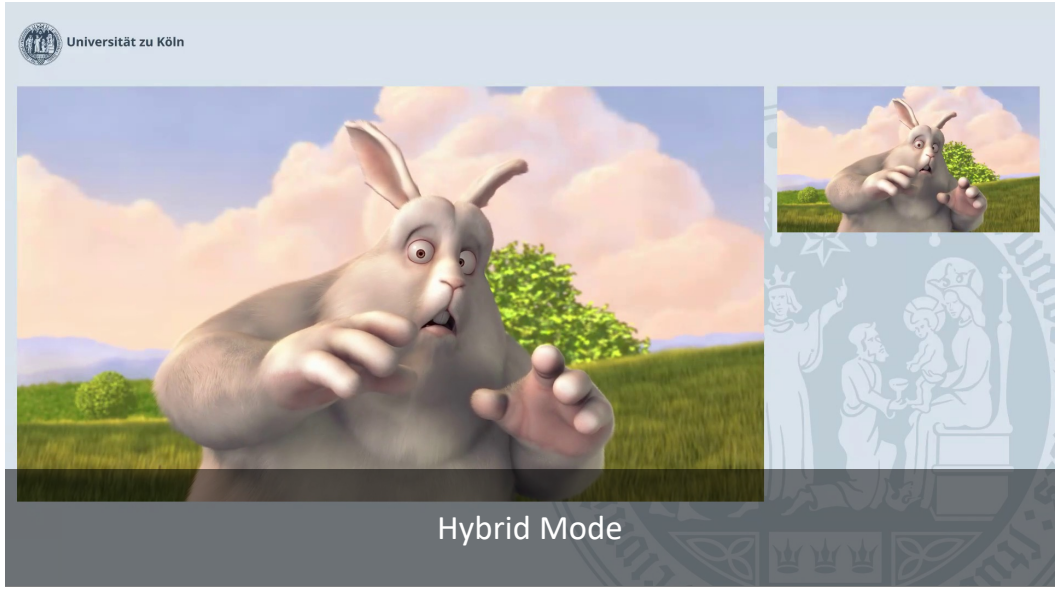
```
# How works:
## Beamer video
# 1.- Scales the video downs to 1280x720 (GPU)
# 2.- Adds a padding to be use for the other video.
(CPU)
# Initial canvas:
# Width: Input witdh + 640 pixels
# Height: Input height + 360 pixels
# Video position:
# X: 20 pixels to the left
# Y: 180 pixels down
# Canvas color: #DBE4ED
# 3.-Upload back to the GPU in NV12 pixel format
## Presenter Video
# 1.- Scales to 512 pixel with the pixel ratio (Take
care to be multiple of 16 if not green lines will
appear) in NV12 pixel format
## First overlay [Beamer and Presenter]
## Image upload to GPU
## Second overlay [merge and logo]
## Encoding in H264
```

Tests: Picture-in-Picture

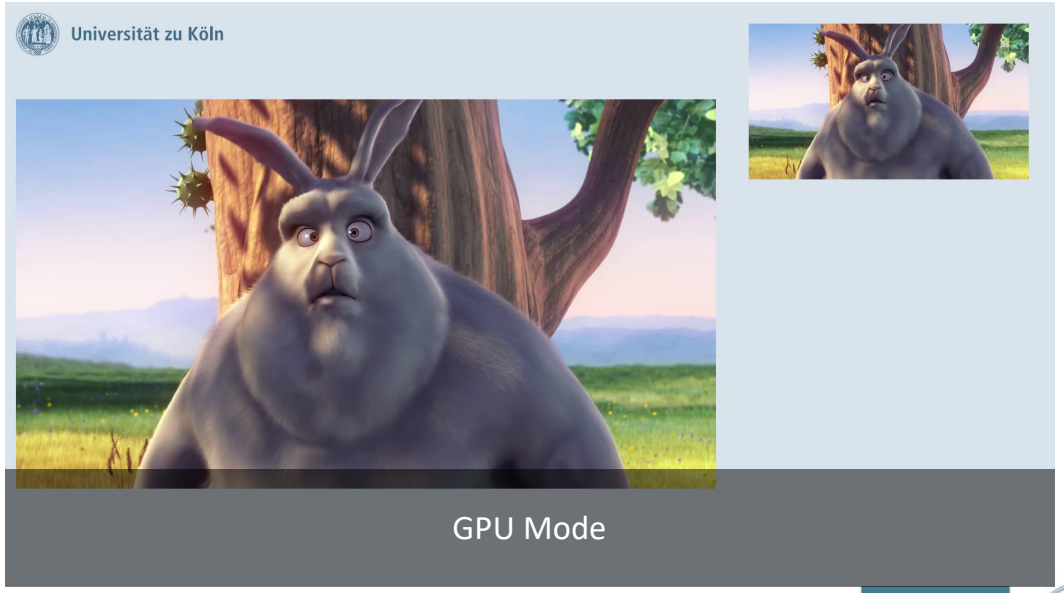
- Hybrid mode (GPU resizes and encodes, CPU applies overlay filter), Allows background image:

```
ffmpeg -nostats -y -i overlay-background.png -i banner_trans.png \
-hwaccel cuda -hwaccel_output_format cuda -i $video_1 \
-hwaccel cuda -hwaccel_output_format cuda -i $video_2 \
-filter_complex "\
[2:v] scale_npp=-2:768, hwdownload, format=nv12 [left]; \
[3:v] scale_npp=-2:270, hwdownload, format=nv12 [right]; \
[0:v][left]overlay=shortest=0:x=23:y=156 [first]; \
[first][right]overlay=shortest=0:x=1412:y=156 [second]; \
[second][1:v]overlay=shortest=0:x=23:y=38 ,hwupload_cuda [final]" \
-map "[final]" -map 2:a -c:a copy -c:v h264_nvenc -zerolatency 1 -rc:v vbr_hq -cq:v 20 \
-b:v 1000k -maxrate:v 1200k -r 25 -b_ref_mode 2 -max_muxing_queue_size 1000 -movflags
+faststart $out_video
```

Compararision

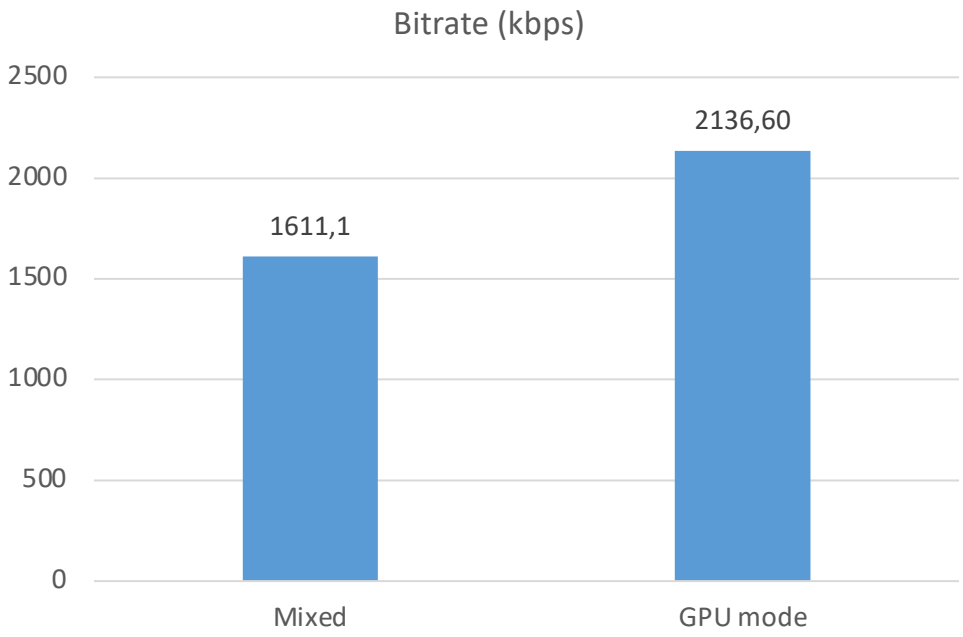


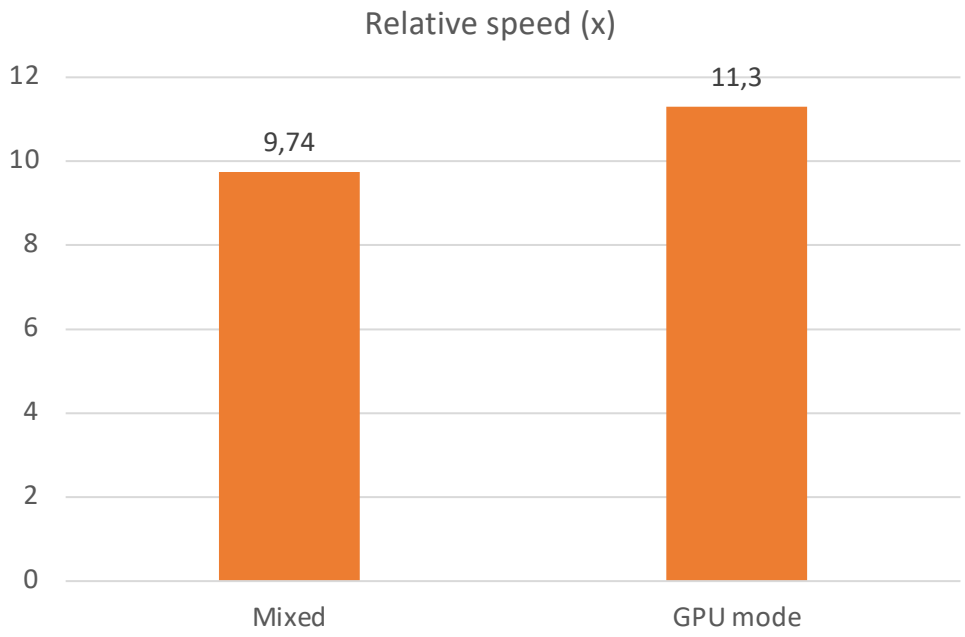
Hybrid Mode



GPU Mode

Tests: Side-by-Side video





The GPU overlay: Advantages / Disadvantages

Advantages

- Can use the high paralellism of GPU
- More performance compared to only CPU

Disadvantages

- Not all video formats supported
- (At the moment) Is not possible to use an image as a background
- Some transparency effects on images won’t work.

How to implement in Opencast

Compile or install	Insert	Add	Write
Compile or install FFMPEG version with CUDA drivers.	Insert the ffmpeg code as an executable shell code in a folder reachable for the “opencast” user.	Add to the list of allowed files in the opencast config the shell code. org.opencastproject.execute.impl.ExecuteServiceImpl.cfg	Write an execute WOH to the opencast workflow.

Conclusions

The GPU Overlay, can manage make overlays, some functions like use an image as background is not available at the moment.

While the performance is not so great as when is only resizing or encoding, this method can take advantage of the high paralelism that a GPU provides.

For advance compositions, like adding images, is better to use an hybrid method CPU + GPU.