Heterogeneous Computing Approach for High Performance Video Resampling and Resizing

Final Presentation
July 6th, 2018

Author: José Pedro Soares João Pereira
Academic Supervisor: Jorge Manuel Gomes Barbosa
Supervisor: Alexandre Ulisses Silva

Mestrado Integrado em Engenharia Informática e Computação
Faculdade de Engenharia da Universidade do Porto

exceeding media challenges

Heterogeneous Computing Approach for High Performance Video Resampling and Resizing

Proposed Solution

Video Resampling and Resizing

Results

Context and Goals

Future Work

Author: José Pedro Soares João Pereira

Mestrado Integrado em Engenharia Informática e Computação
Faculdade de Engenharia da Universidade do Porto

1

Context and Goals

What does this project want to achieve?

Context

- Increasing popularity of high resolutions of video
- Videos of high resolution are made of an high number of pixels per frame

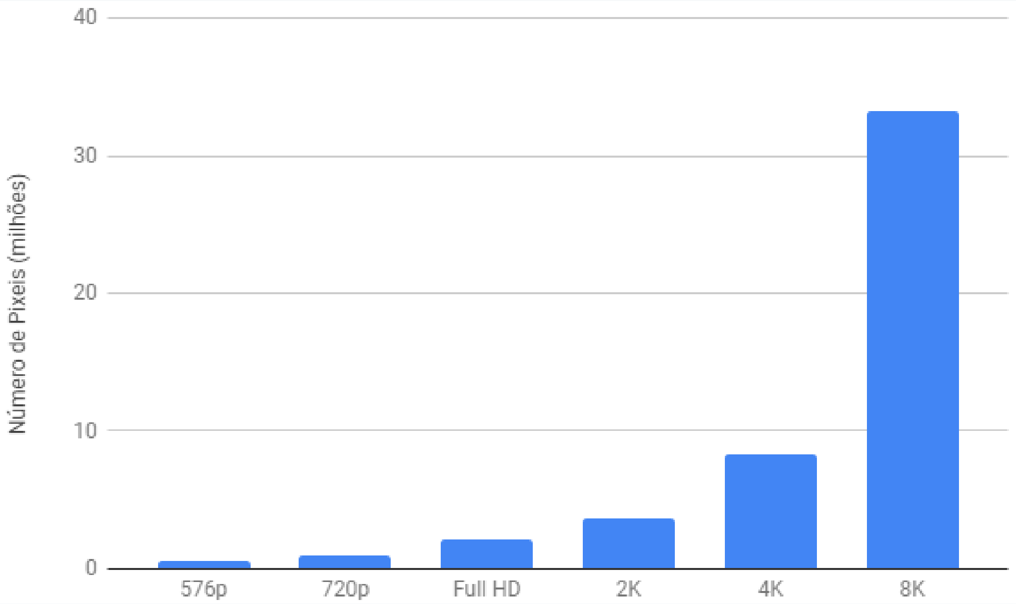


Fig 1: Millions of pixels per frame by video resolution

Problem

- Video production processes operate over each pixel of each frame
- The computational resources needed are proportional to the number of pixels
- Video production solutions must follow the ever stricter customer demands
- Systems should be restructured considering processing scalability

Problem

- Video production processes operate over each pixel of each frame
- The computational resources needed are proportional to the number of pixels
- Video production solutions must follow the ever stricter customer demands
- Systems should be restructured considering processing scalability
- Heterogeneous computing approach uses different types of processing units
  - Different CPU models, in the same machine or in machines of the same network
  - CPU and FPGA, high performance integrated circuit for arithmetic operations
  - CPU and GPU, processing unit destined to render graphic components

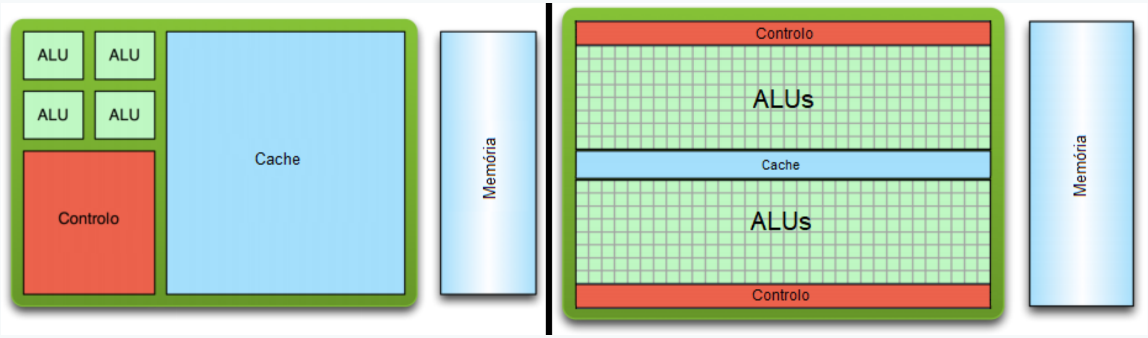


Fig 2: CPU vs GPU architecture

Goals

- Take advantage of the existent hardware in *mxfSpeedRail* products
- Implement an own heterogeneous solution for video resampling and resizing
  - Using a graphics processing unit to speed up the process
- Decrease the execution time of the process in comparison to *FFmpeg*
- Develop a scalable solution to real time video resampling and resizing
  - Of high resolution uncompressed video

Current Solution

- *FFmpeg* based solution
- Uncompressed video resampling and resizing is a single-thread process

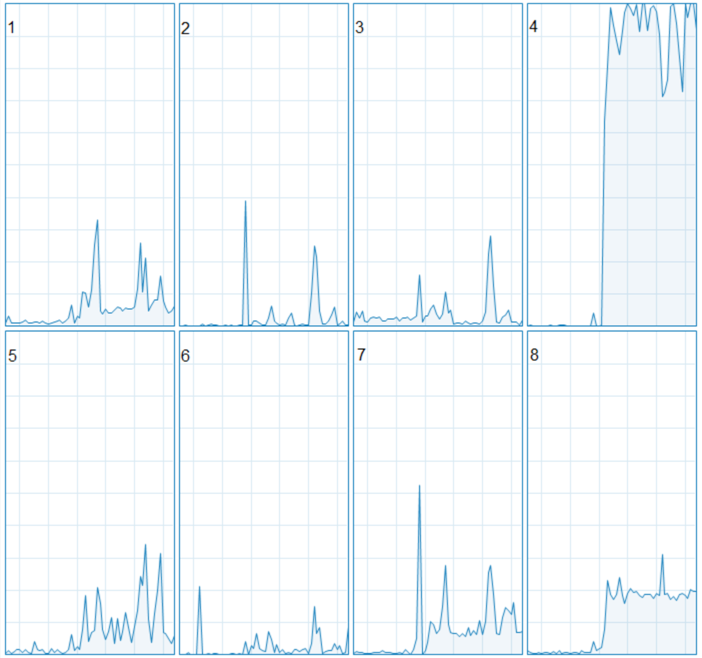


Fig 3: CPU usage of *FFmpeg* video resampling by thread

Current Solution

- Vectorization is used to accelerate the processing speed
- Vectorization allows the implicit parallelization of a block of code
- Using low level code instructions
    - As an example, SSE, AVX and AVX2 intrinsics

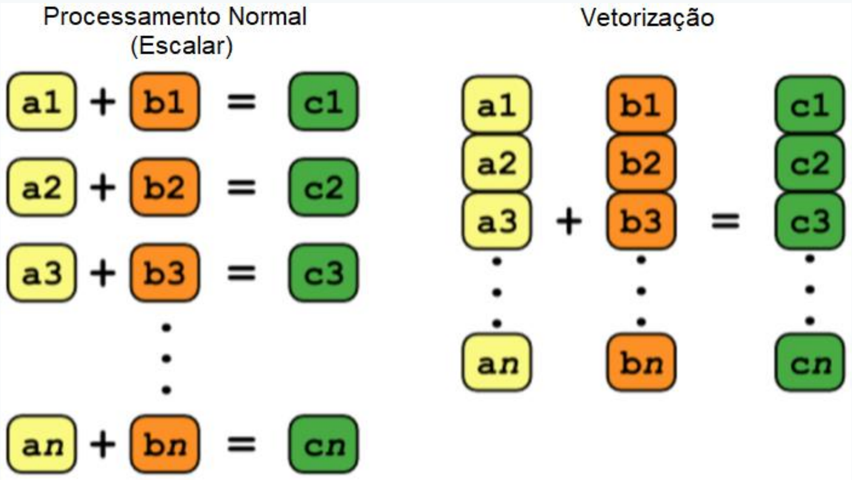


Fig 4: Vectorization of a sum operation

Other Options

- *FFmpeg* already support graphic acceleration of video resampling and resizing
  - BUT only using the Nvidia hardware encoder, NVENC
- The *FFmpeg* implementation is an hardware solution
  - As an example, by a SDI source
- This project solution is purely software

2

Video Resampling and Resizing

What is video resampling and resizing?

YUV Color Space

- Frequently used in digital video representation
- Additive color model constituted by three different components
    - A luma component and two different chrominance components

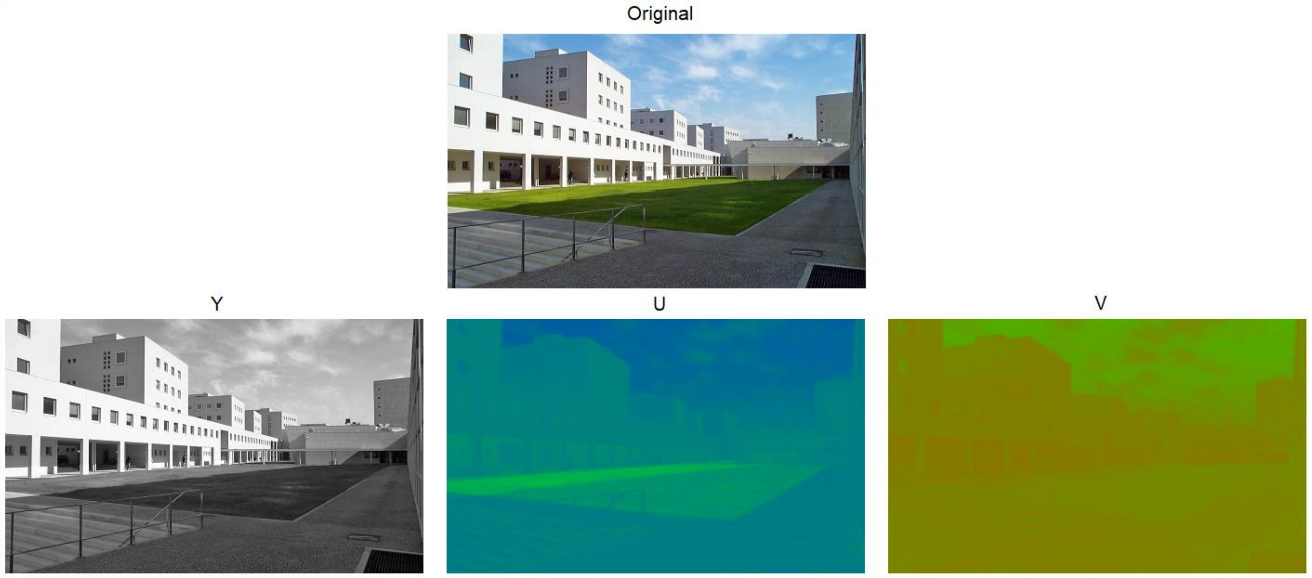


Fig 5: YUV color space components

YUV Color Space – Chrominance Subsampling

- The model takes advantage of the weak visual perception to chrominance changes
- Chrominance subsampling reduces the number of color samples
  - Smaller chrominance components implies a smaller video file size

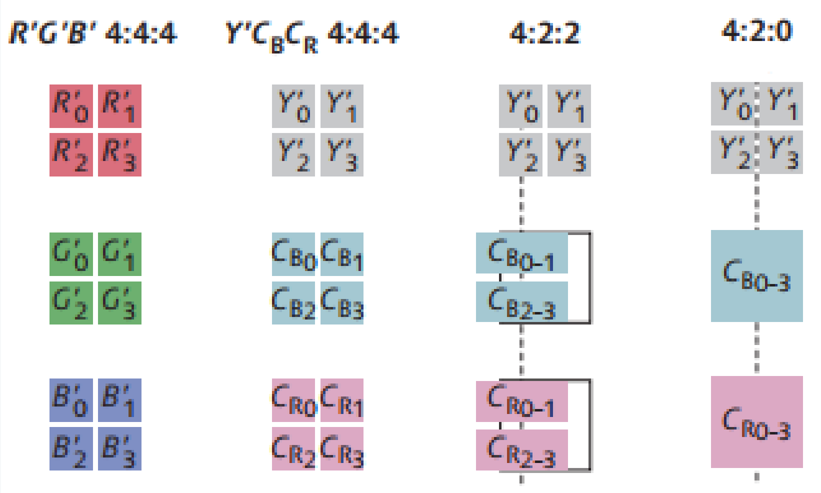


Fig 6: Chrominance subsampling types

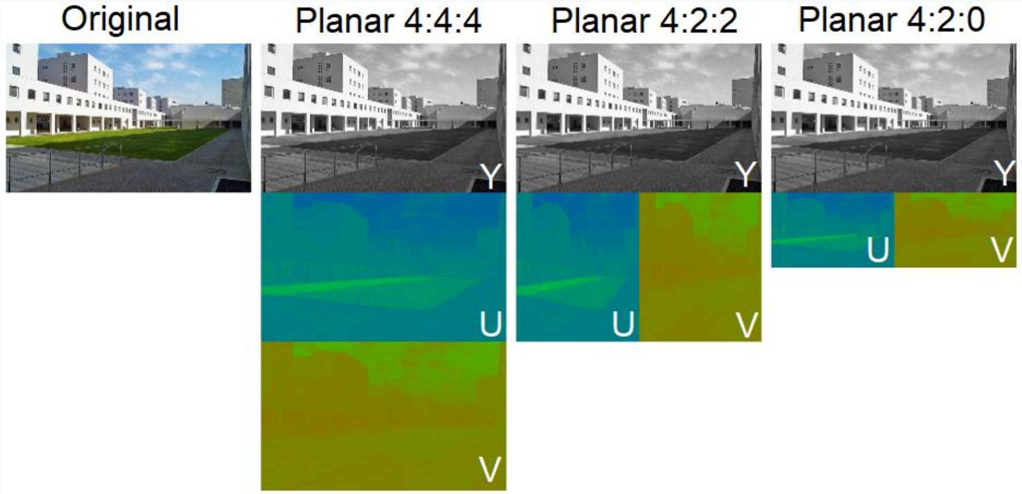


Fig 7: Size of an image's components by type of subsampling

YUV Color Space – Pixel Formats

- This color space pixel formats are divided into different types:

Plano A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7

Plano B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
U0	U0	U0	U0	U0	U0	U0	U0	U2	U2	U2	U2	U2	U2	U2	U2
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
U4	U4	U4	U4	U4	U4	U4	U4	U6	U6	U6	U6	U6	U6	U6	U6

Plano C

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V0	V0	V0	V0	V0	V0	V0	V0	V2	V2	V2	V2	V2	V2	V2	V2
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
V4	V4	V4	V4	V4	V4	V4	V4	V6	V6	V6	V6	V6	V6	V6	V6

Fig 8: YUV422p planar format

Plano A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
U0	U0	U0	U0	U0	U0	U0	U0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	V0	V0	V0	V0	V0	V0	V0	V0	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1

Fig 9: UYVY packed format

Plano A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7

Plano B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
U0	U0	U0	U0	U0	U0	U0	U0	V0	V0	V0	V0	V0	V0	V0	V0	U2	U2	U2	U2	U2	U2	U2	U2	V2	V2	V2	V2	V2	V2	V2	V2

Fig 10: NV12 semi-planar format

Resampling and Resizing

- Resampling an image implies the change of its dimensions
  - It is the insertion or remotion of the number of pixels of each of the video's frames
- This process is achieved by different algorithms
  - Each algorithm trades off execution speed by the quality of the operation's results
  - Each pixel's color of a resampled image is obtained by the corresponding original pixels

$$k_{nn}(x) = \begin{cases} 1 & |x| < 0.5 \\ 0 & \textit{senão} \end{cases}$$

Fig 11: Algorithm *Nearest Neighbor*

$$k_{linear}(x) = \begin{cases} 1 - |x| & |x| < 1 \\ 0 & \textit{senão} \end{cases}$$

Fig 12: Algorithm linear

$$k_{spline}(x) = \begin{cases} 1.4 \times |x|^3 - 2.4 \times |x|^2 + 1 & |x| < 1 \\ -0.6 \times |x|^3 + 3 \times |x|^2 - 4.8 \times |x| + 2.4 & 1 \leq |x| < 2 \\ 0 & \textit{senão} \end{cases}$$

Fig 13: Algorithm by *spline*, or cubic

Resampling and Resizing

- The different algorithms differ in terms of execution speed and the quality of the resampled image results

Fig 14: Quality of results of each resampling algorithm

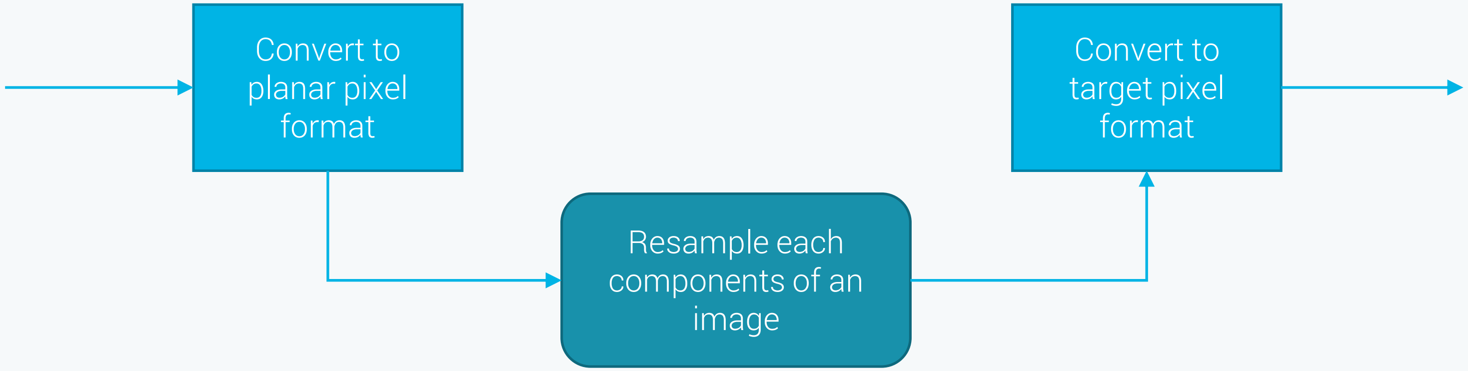
3

Proposed Solution

How was the solution implemented?

Proposed Solution

- The resampling process is divided into two operations
  - Due to the different combinations of input and output pixel' formats
- The pixel format conversion operation
- The resampling operation



Pixel Format Conversion

- This operation is exclusively executed by the CPU
- The parallelization was achieved by OpenMP

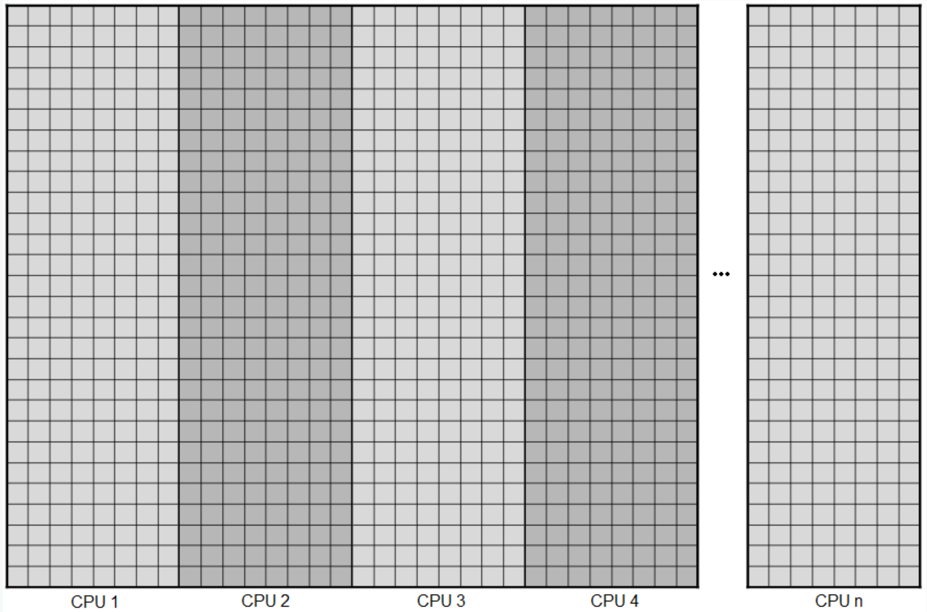


Fig 15: Work division of the conversion operation by different CPU threads

Resampling

- This operation is exclusively executed by the GPU
- Implemented by the Nvidia framework CUDA

Fig 16: Texture memory access

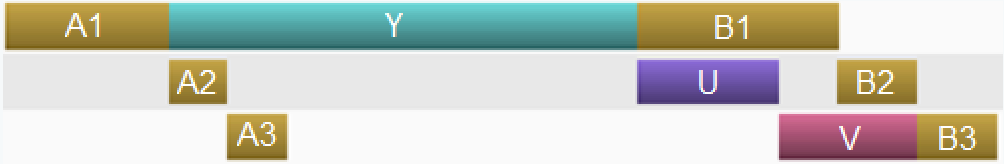


Fig 17: Data transfer parallelization with resample operation

4

Results

What are the results achieved by the proposed solution?

Pixel Format Conversion

- Tested with two different CPU models
- Performance gain of 51% in comparison to *FFmpeg*

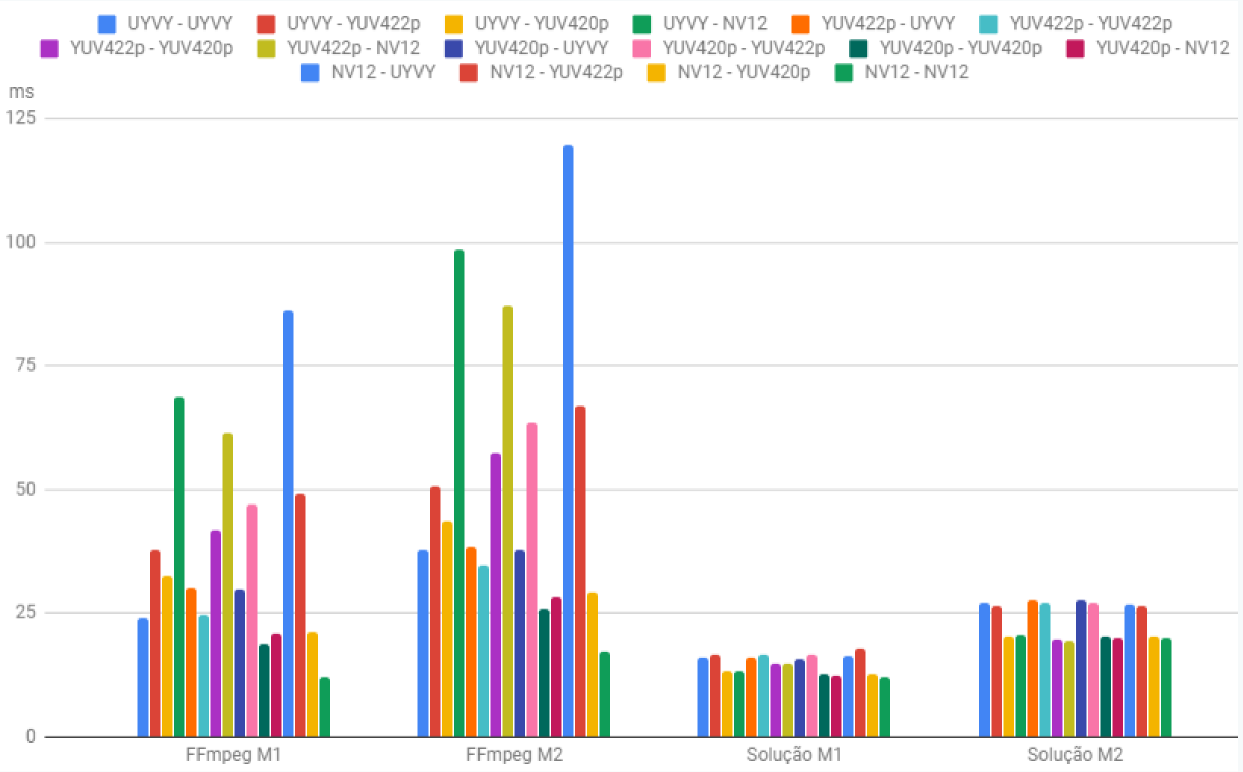


Fig 18: Execution time of the pixel format conversion operation

Resampling

- Tested with two different GPU models (Nvidia Quadro P600 and P2000)
- Performance gain of 58% in comparison to *FFmpeg*

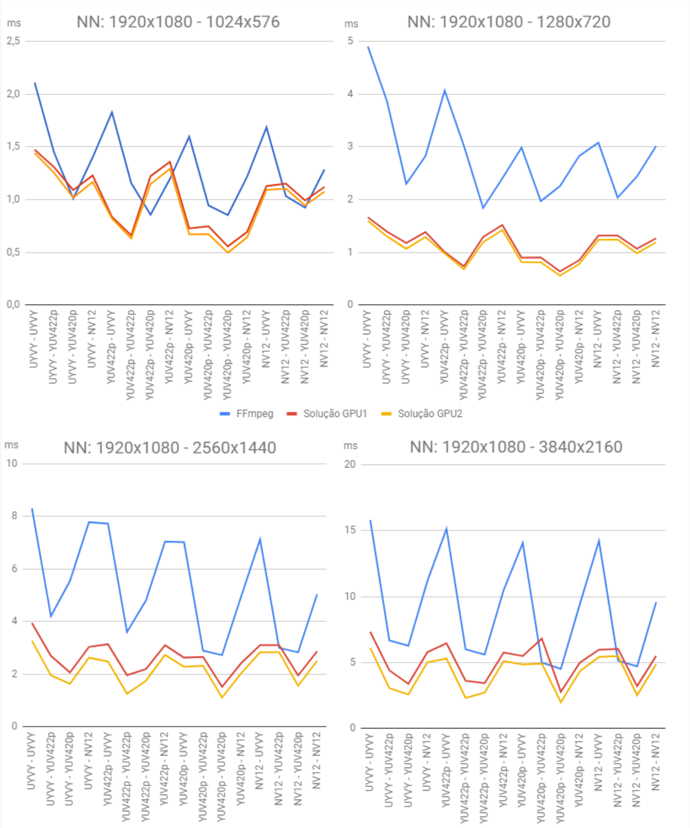


Fig 19: Execution time of the NN algorithm

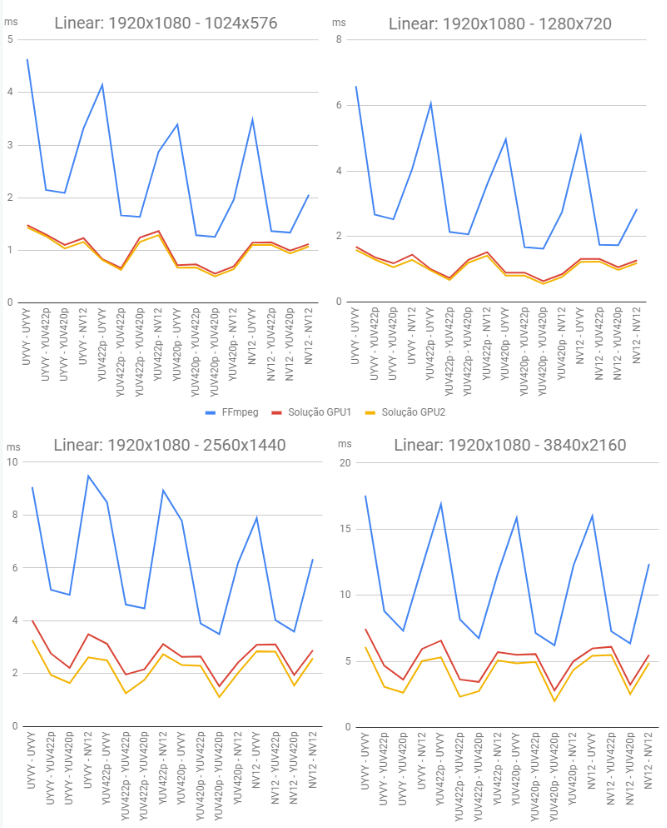


Fig 20: Execution time of the linear algorithm

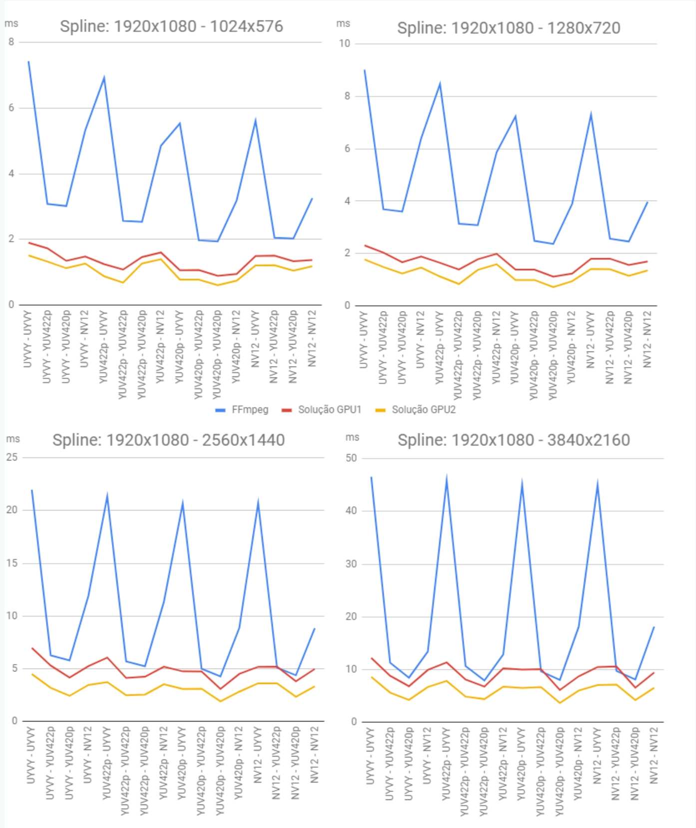


Fig 21: Execution time of the spline algorithm

Video Resampling and Resizing

- Execution time of resampling and resizing a *Full High Definition* video

	Planar				Não Planar			
NN	1024x576	1280x720	2560x1440	3840x2160	1024x576	1280x720	2560x1440	3840x2160
FFmpeg	1,027	2,463	3,701	5,486	1,539	3,262	6,874	12,491
Solução GPU1	-6,08%	-56,86%	-38,75%	-25,86%	-30,48%	-62,13%	-55,93%	-52,73%
Solução GPU2	-11,66%	-60,34%	-51,33%	-41,81%	-33,44%	-64,39%	-62,26%	-58,98%
Linear								
FFmpeg	1,598	2,023	4,282	7,247	3,233	4,488	8,016	14,325
Solução GPU1	-39,44%	-47,54%	-46,45%	-43,10%	-66,81%	-72,20%	-61,37%	-58,48%
Solução GPU2	-42,83%	-51,64%	-57,95%	-55,71%	-68,37%	-74,24%	-67,39%	-64,28%
Spline								
FFmpeg	2,398	2,925	5,206	9,216	5,263	6,520	15,707	30,616
Solução GPU1	-45,69%	-45,51%	-17,06%	-13,61%	-73,63%	-73,22%	-65,93%	-66,44%
Solução GPU2	-58,08%	-60,66%	-48,36%	-44,88%	-78,72%	-79,57%	-77,74%	-77,20%

Fig 22: Performance gain of video resampling and resizing operation in comparison to *FFmpeg*

5

Future Work

What can be done to improve the proposed solution?

Future Work

- Implement vectorization in the pixel format conversion operation and compare with the current solution
- Integrate the proposed solution in MOG Technologies' product
- Create a wider support of pixel formats and resampling algorithms
- Scale the proposed solution to a processing distributed system

exceeding media challenges

Any questions?

Thank you!

Author: José Pedro Soares João Pereira
Academic Supervisor: Jorge Manuel Gomes Barbosa
Supervisor: Alexandre Ulisses Silva

Mestrado Integrado em Engenharia Informática e Computação
Faculdade de Engenharia da Universidade do Porto