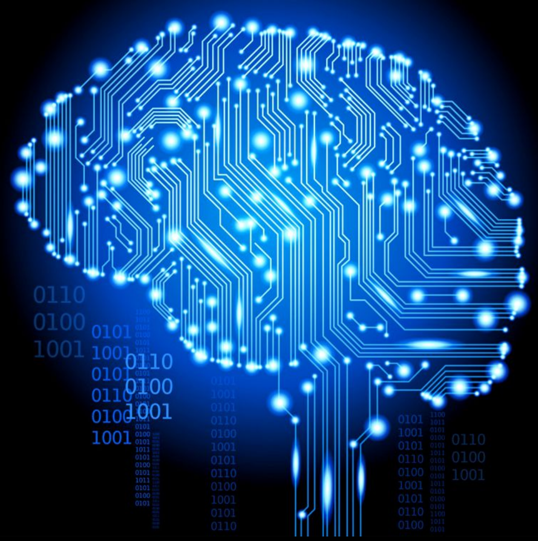


S9391
GstCUDA: Easy GStreamer and CUDA Integration

Eng. Daniel Garbanzo

MSc. Michael Grüner

GTC March 2019

Agenda

- About RidgeRun
- GStreamer Overview
- CUDA Overview
- GstCUDA Introduction
- Application Examples
- Performance Statistics
- GstCUDA Demo on TX2
- Q&A

About Us

- US Company - R&D Lab in Costa Rica
- 15 years of experience
- Embedded Linux and GStreamer experts
- Custom multimedia solutions
- Digital signal/image processing
- AI and Machine Learning solutions
- System optimization: CUDA, GStreamer, OpenCL, OpenGL, OpenVX, Vulkan
- Support for embedded and resource constrained systems
- Professional services, dedicated teams and specialized tools

Multimedia Is Everywhere

Medical Industry

Automotive Industry

Smart Devices

Computer Vision

- **Complex multimedia applications require a lot of processing resources**
- **GStreamer offers a flexible way for creating multimedia applications**
- **CUDA offers high performance accelerated processing capabilities**

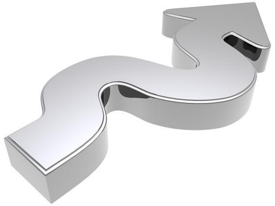
gstreamer

- Open source framework for audio and video applications
- Based on a pipeline architecture
- Extensible design based on plugins (more than 1000 freely available)
- Automatic format and synchronization handling
- Tools for easy prototyping

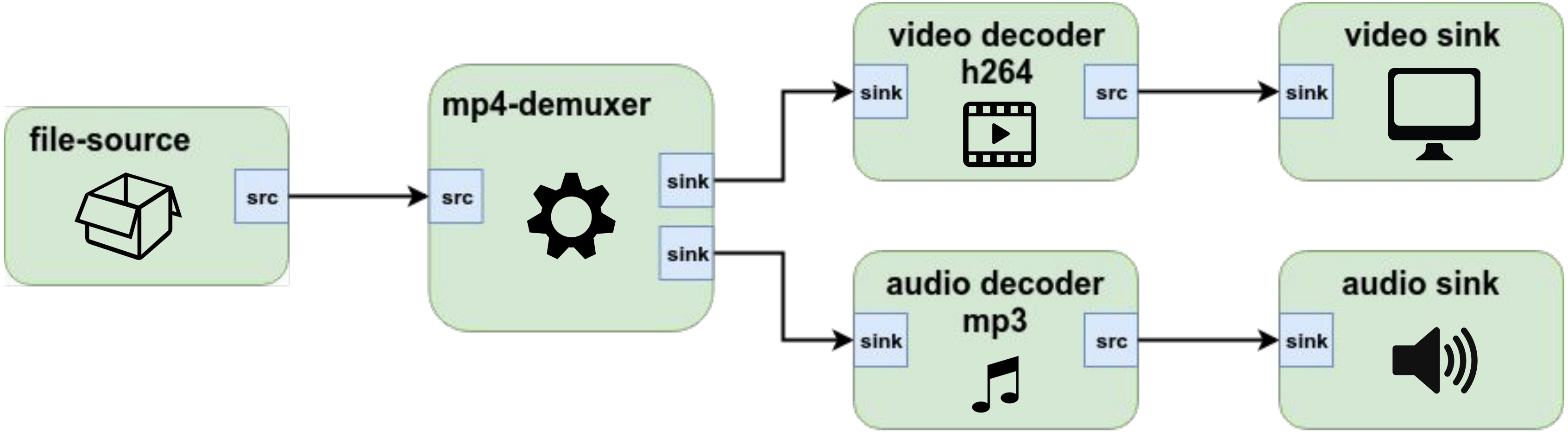
Modularity

Portability

Flexibility

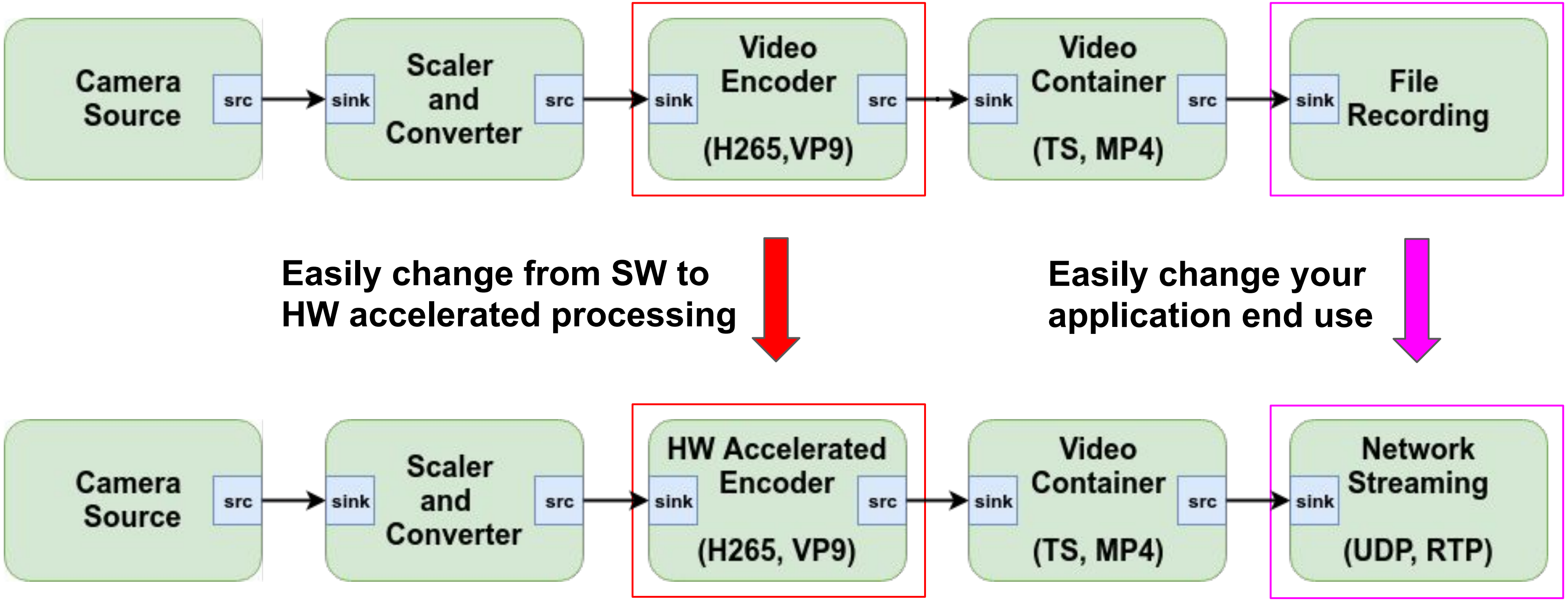


Basic MP4 player GStreamer Pipeline

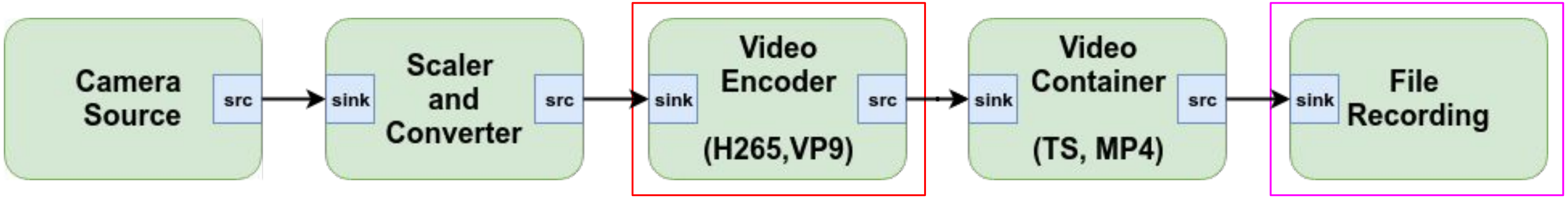


- Each plugin represents a different processing module
- The plugins are linked and arranged in a pipeline
- Freedom to build arbitrary pipelines for different applications

Modular design lets you change your application easily!

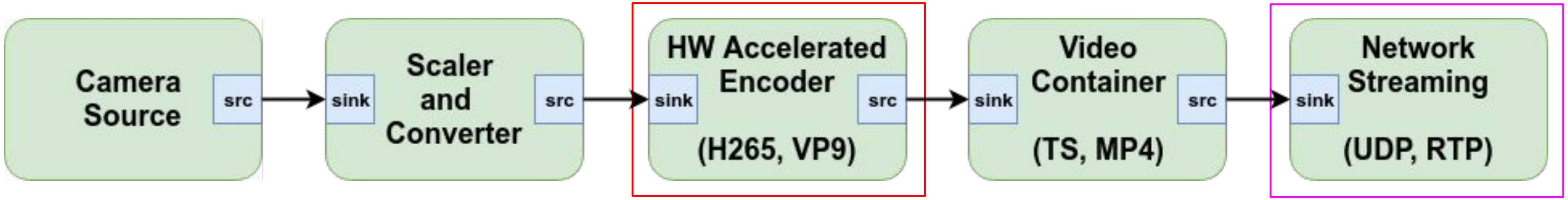


Modular design lets you change your application easily!



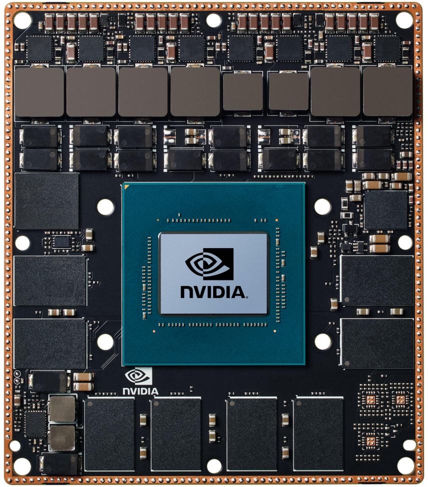
Code equivalent :

```
gst-launch v4l2src ! videoconverter ! x265enc ! mpegtsmux ! filesink
```



Code equivalent :

```
gst-launch v4l2src ! videoconverter ! omxh265enc ! mpegtsmux ! udpsink
```



Development environment for high performance GPU-accelerated applications

General purpose data processing via parallel algorithm execution on GPU

Extensive development, debugging and profiling set of tools

GstCUDA Integrates the Best of Both Worlds

GstCUDA

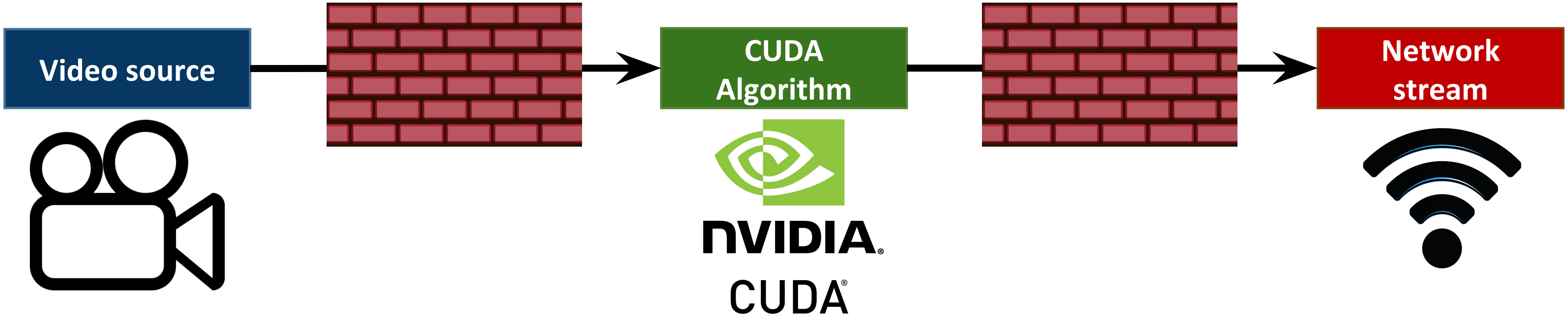
Framework enabling easy integration of CUDA algorithms into GStreamer pipelines

Eliminates the need to learn GStreamer internals

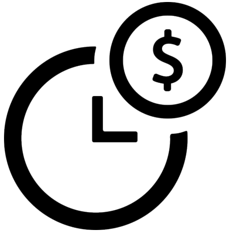
Focus on your CUDA algorithm to reduce time to market!

What Does GstCUDA Solve?

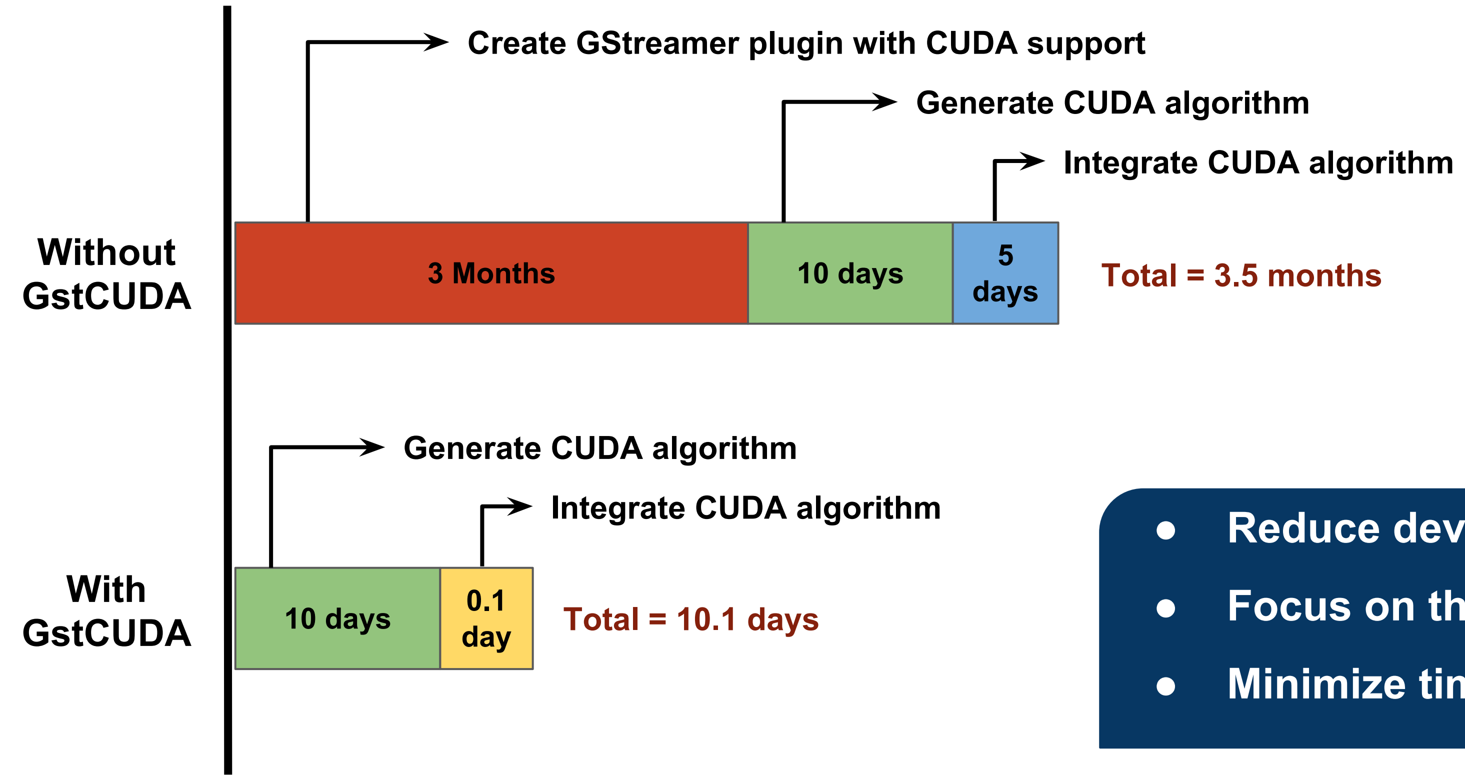
Integration Complexities



- **A lot of roadblocks between CUDA and GStreamer**
- **These are complex and time consuming**
- **Time is money!**

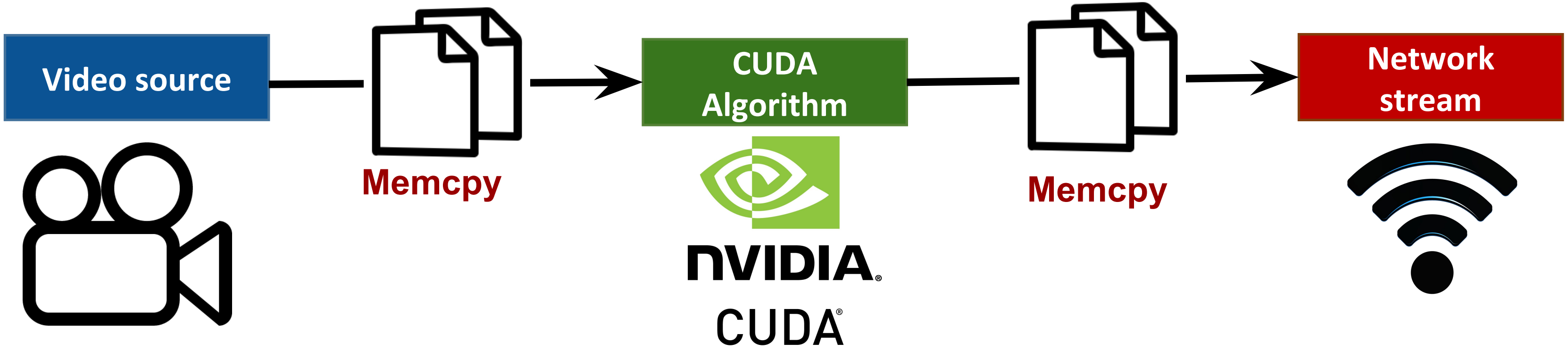


Development Time

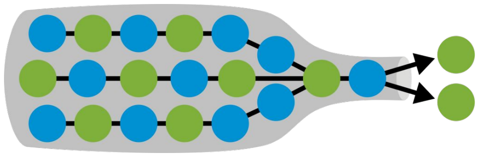


- Reduce development time
- Focus on the CUDA logic
- Minimize time to market

Performance Bottleneck



- Data transfers can be a bottleneck
- Memory copies can degrade performance
- Incompatibility between different memory types



Performance Bottleneck

Without GstCUDA

- Data transfers bottleneck cause poor performance
- Limited framerate at high resolutions

With GstCUDA

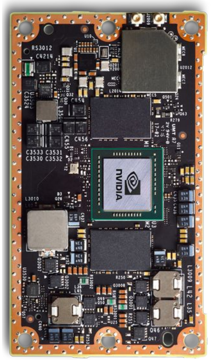
- Efficient memory handling improves performance
- Up to 2x 4K@60fps

Supported Platforms

RidgeRun
Embedded Solutions

- **Focused for NVIDIA Embedded Platforms**

Jetson TX1, TX2, TX2i and Nano



Jetson AGX Xavier

GstCUDA Key Features

Allows CUDA algorithm easy integration into GStreamer pipelines

Out of the box quick prototyping tools

Fine-grained control of image memory layout (planes, strides, etc ...)

Automatic efficient memory handling

GstCUDA Key Features

High performance for GStreamer/CUDA applications

Zero memory copy interface between CUDA and GStreamer

Direct handling of HW (NVMM) buffers

Unified Memory allocation mechanism

Framework Overview

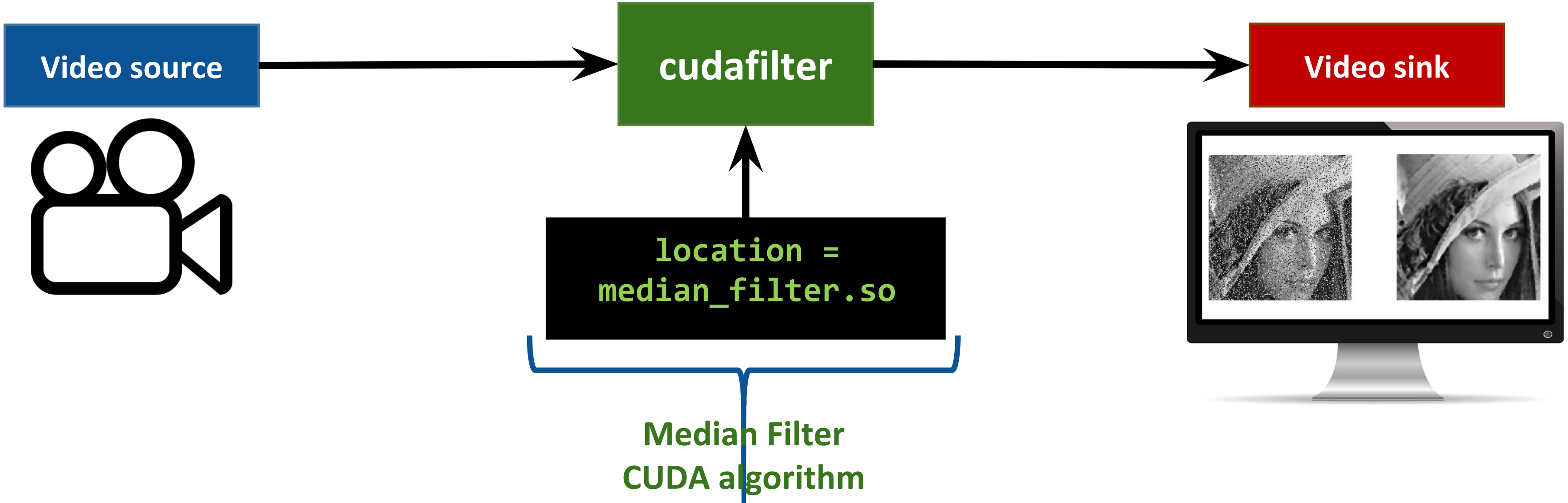
Quick Prototyping Elements

GStreamer elements for CUDA quick prototyping

Algorithms are loaded at runtime as plug-ins

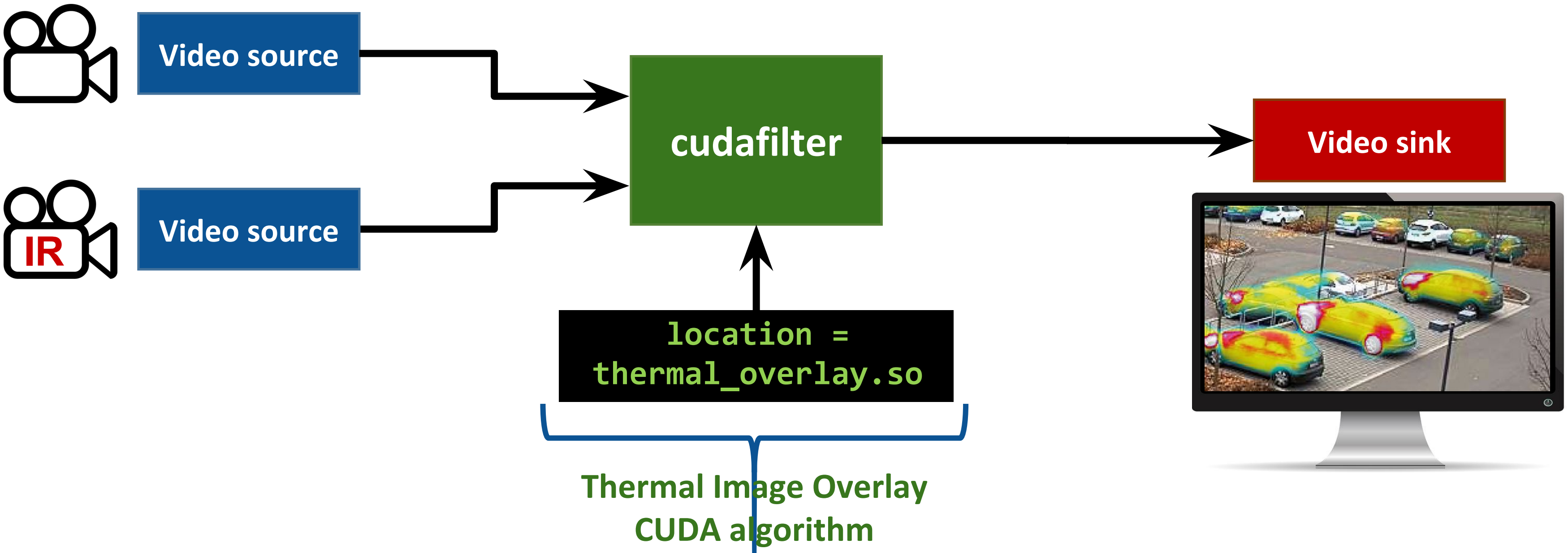
Change the algorithm on the fly

Cudafilter Element



Single input / Single output configuration

Cudamux Element



Multiple input / Single output configuration

CUDA Algorithm Interface

- **Make your CUDA algorithm compatible by implementing these interfaces**

Cudafilter Interface

```
bool open();

bool close();

bool process (const GstCudaData &inbuf,
  GstCudaData &outbuf);

bool process_ip (const GstCudaData
  &inbuf, GstCudaData &outbuf);
```

Cudamux Interface

```
bool open();

bool close();

bool process (vector<GstCudaData>
  &inbufs, GstCudaData &outbuf);

bool process_ip (vector<GstCudaData>
  &inbufs, GstCudaData &outbuf);
```

Buffer Processing Methods

process_ip
(In place)

Algorithm outputs are written to the input buffer

process
(Not in place)

Algorithm inputs and outputs are different buffers

Create Your Custom Element

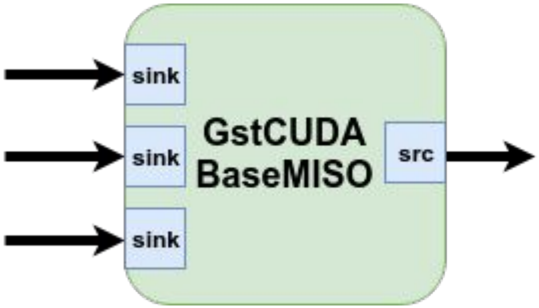
- Some applications may require specialized elements
- GstCUDA provides bases classes to simplify development

GstCUDABaseFilter:

- Single-input / Single-output topology

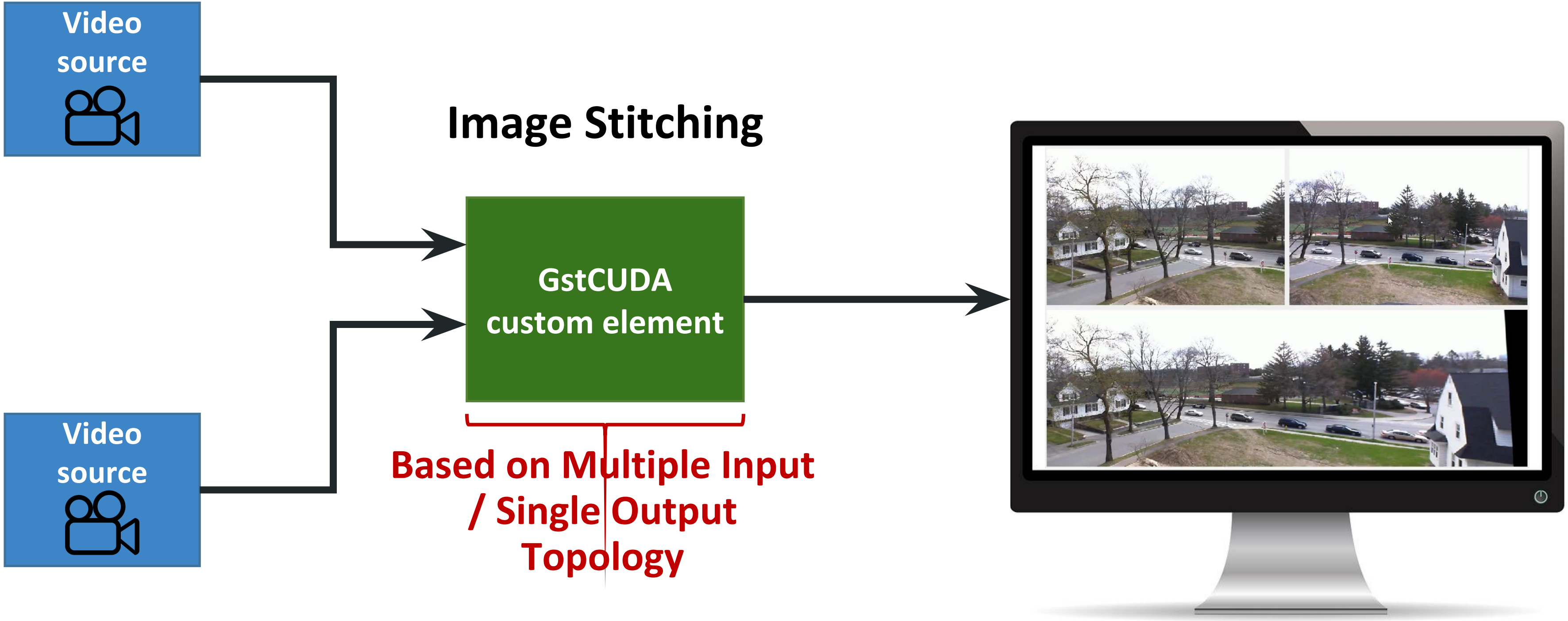
GstCUDABaseMISO:

- Multiple-input / Single-output topology



GstCUDA Framework Usage Example

- Inherit parent classes and focus on the algorithm!



GstCUDA Framework Summary

- **The framework includes:**

GstCUDA API

- Utils to handle memory interfaces
- GStreamer Unified Memory allocators
- Parent classes for different topologies

Quick prototyping elements

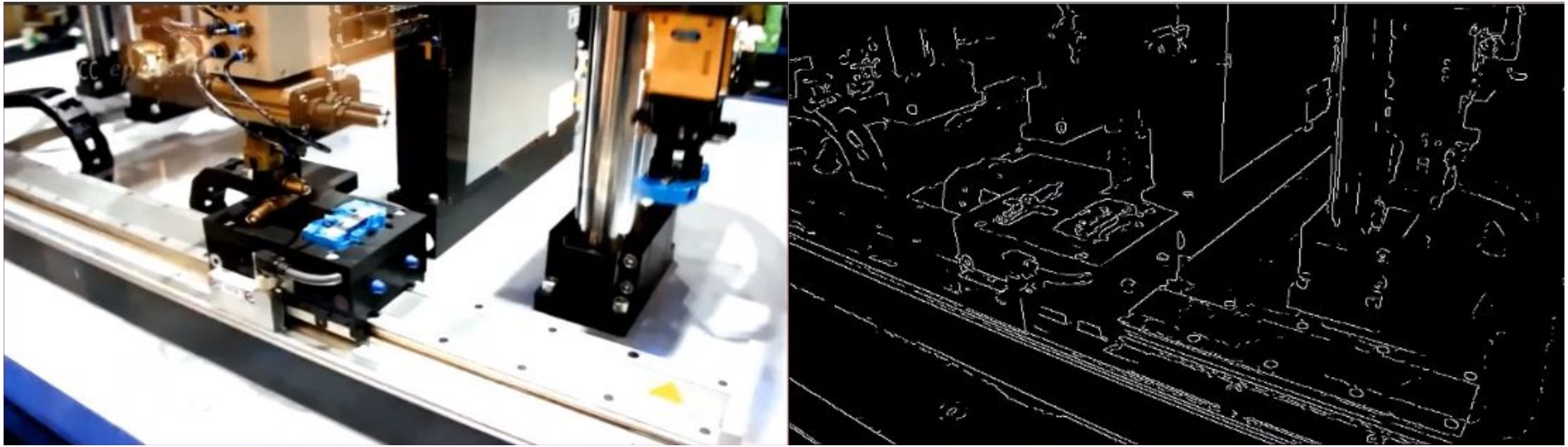
- Generic elements to evaluate custom algorithms
- Runtime loading of CUDA algorithms

Set of examples

- Complete GstCUDA element boilerplate
- CUDA algorithms for the prototyping elements

GstCUDA Application Areas Examples Video

Industrial Applications: Border Enhancement



Automation Applications: Hough Transform

Security Applications: Motion Detection/Estimation

Performance Statistics

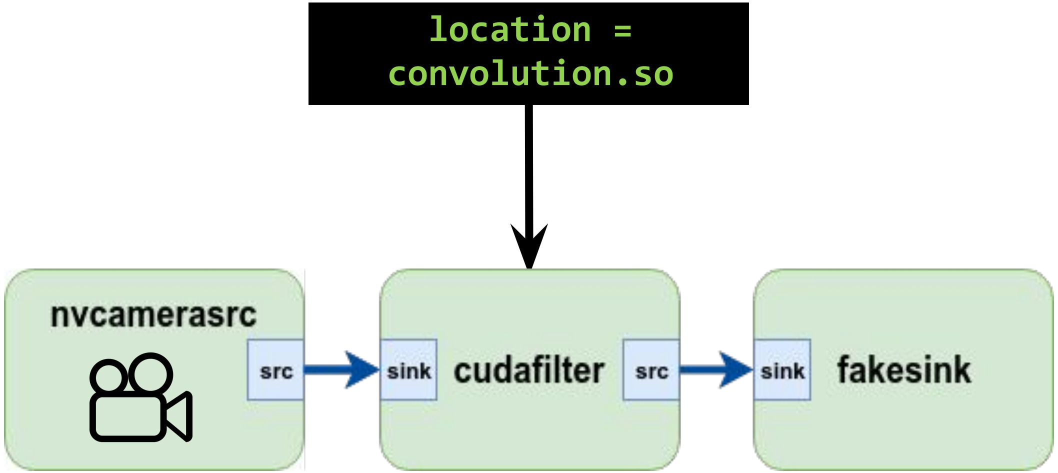
Varying Algorithm / Fixed Image Size

Test Conditions

- Image convolution algorithm

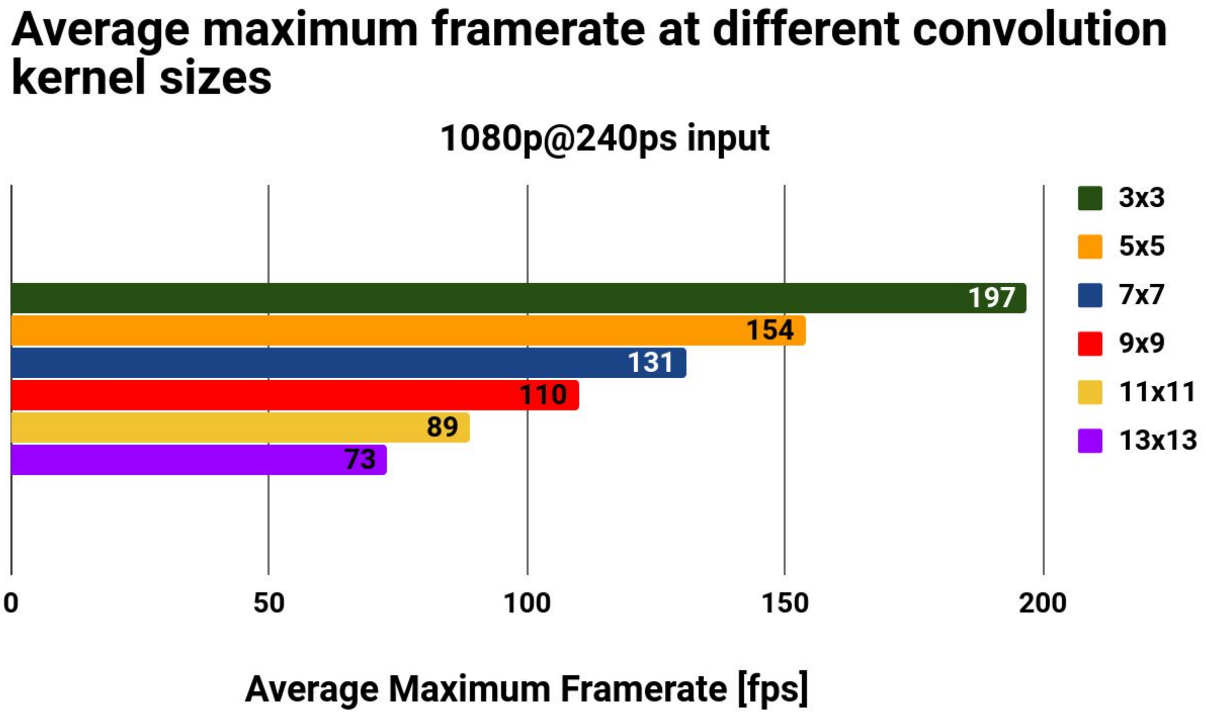
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

- Stressing compute capabilities
- Variable convolution kernel size
- 1080p@240fps / 1080p@60fps stream input
- Cudafilter element
- Unified Memory allocator
- Jetson TX2 platform
- Not In-place



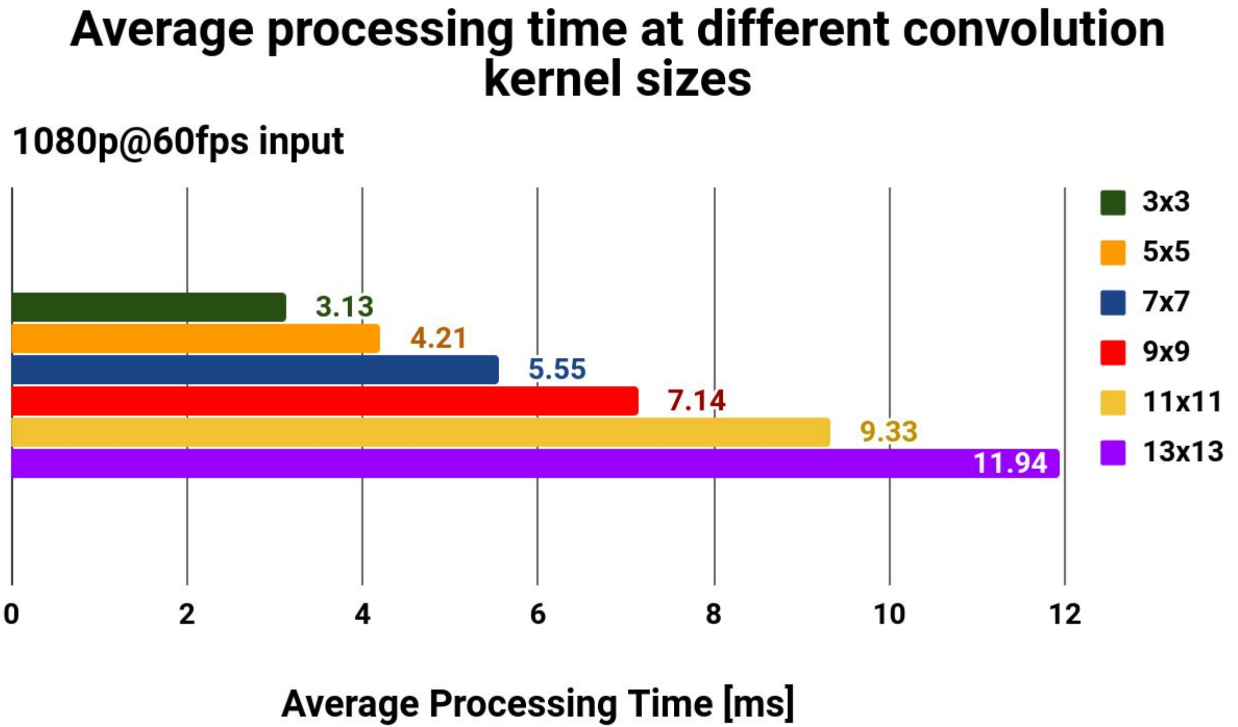
Varying Algorithm / Fixed Image Size

Framerate Stats



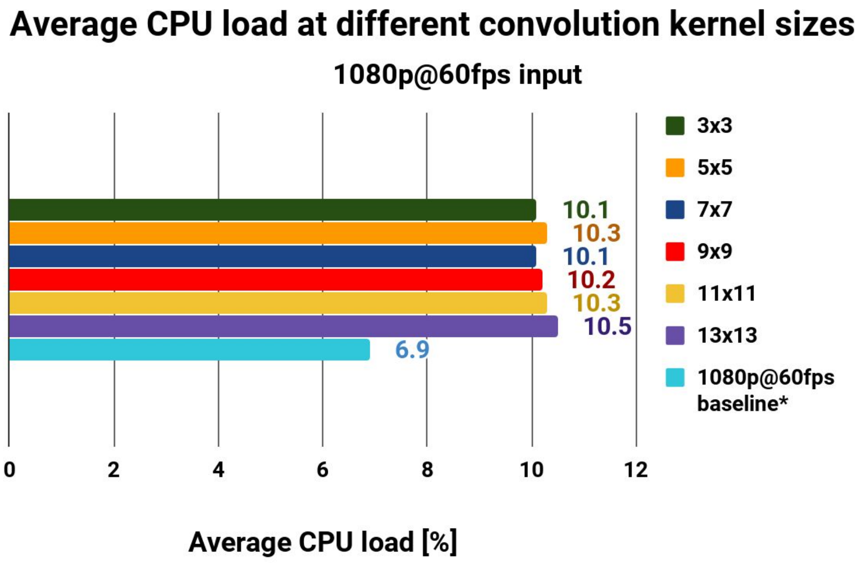
Varying Algorithm / Fixed Image Size

Processing Time Stats

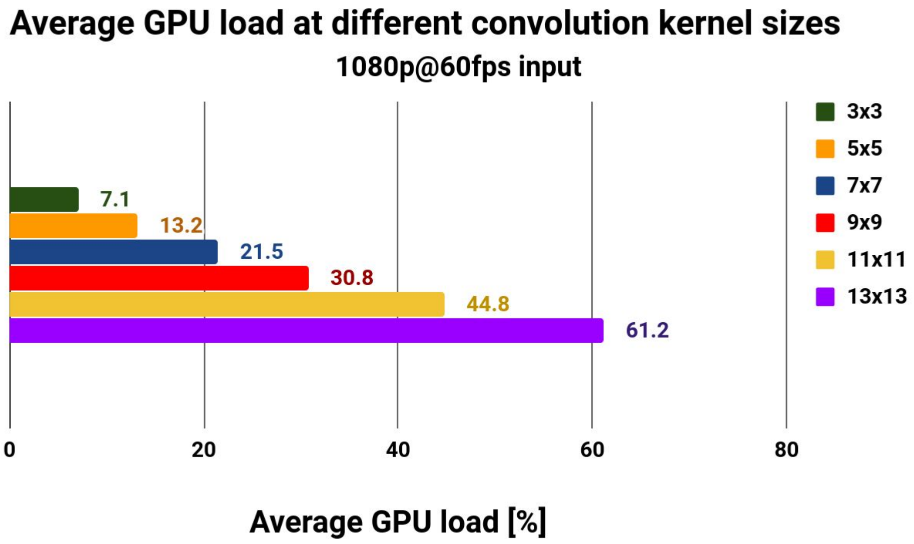


Varying Algorithm / Fixed Image Size

CPU Load Stats



GPU Load Stats



*baseline = simple capture pipeline (without GstCUDA)

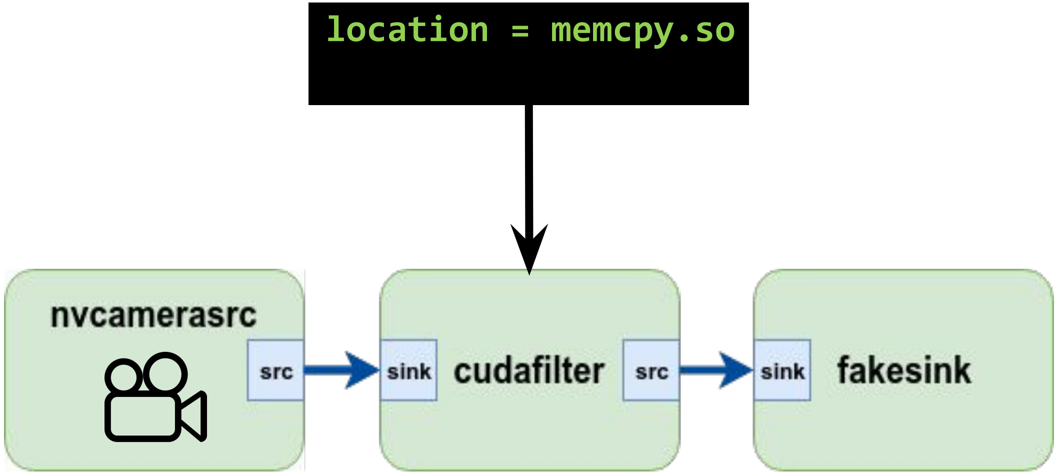
Fixed Algorithm / Varying Image Size

Test Conditions

- **Memory copy algorithm**

$$y_{(n,m)} = x_{(n,m)}$$

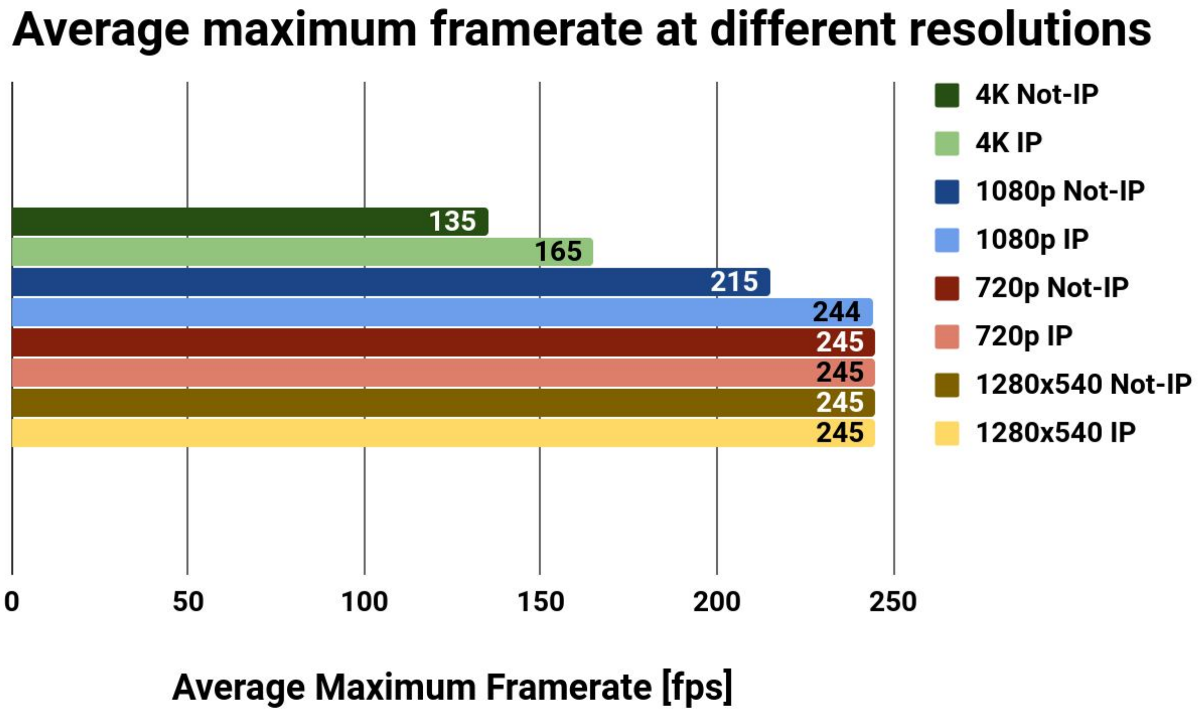
- **Stressing data transfer**
- **Variable input resolution**
- **Cudafilter element**
- **Unified Memory allocator**
- **Jetson TX2 platform**
- **In-place vrs not In-place**



Fixed Algorithm / Varying Image Size

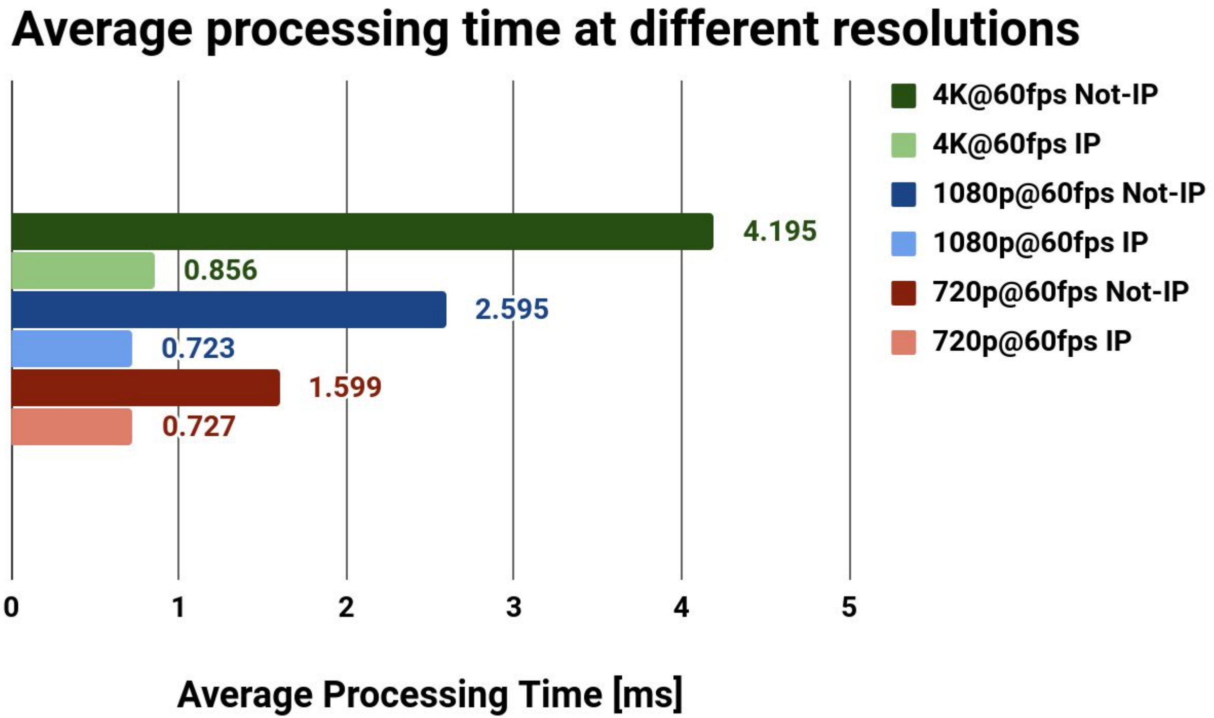
Framerate Stats

Note: Maximum Framerate limited to 245 fps by the video source



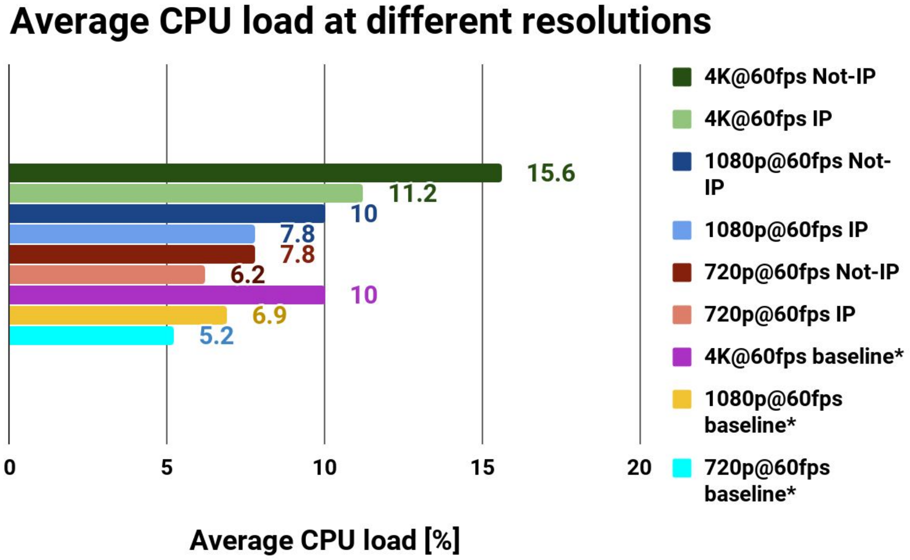
Fixed Algorithm / Varying Image Size

Processing Time Stats

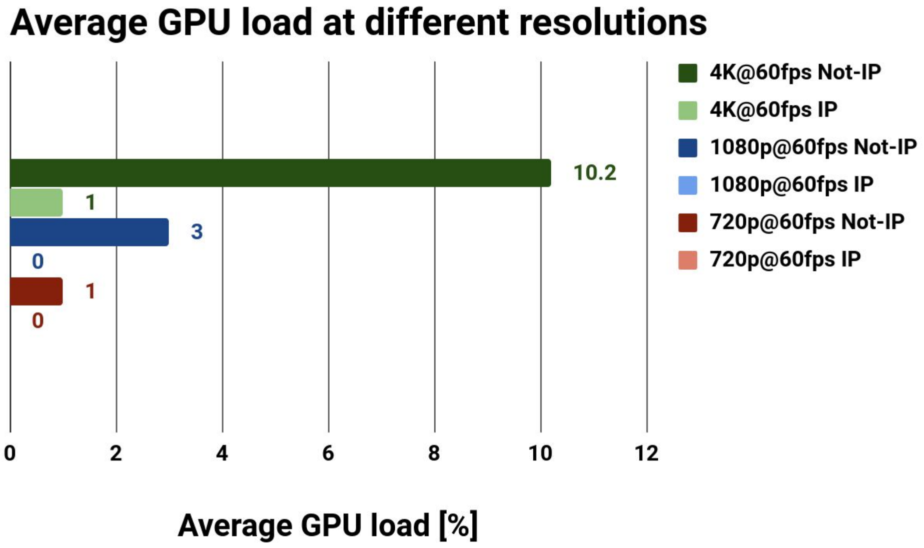


Fixed Algorithm / Varying Image Size

CPU Load Stats



GPU Load Stats



*baseline = simple capture pipeline (without GstCUDA)

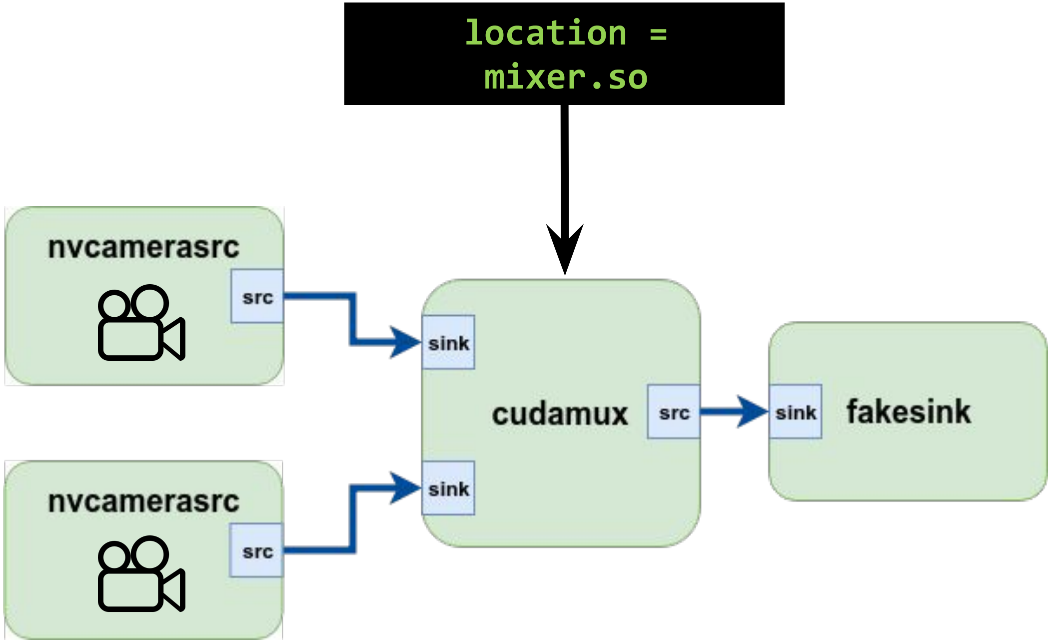
Fixed Algorithm / Varying Image Size

Test Conditions

- Simple image mixing algorithm

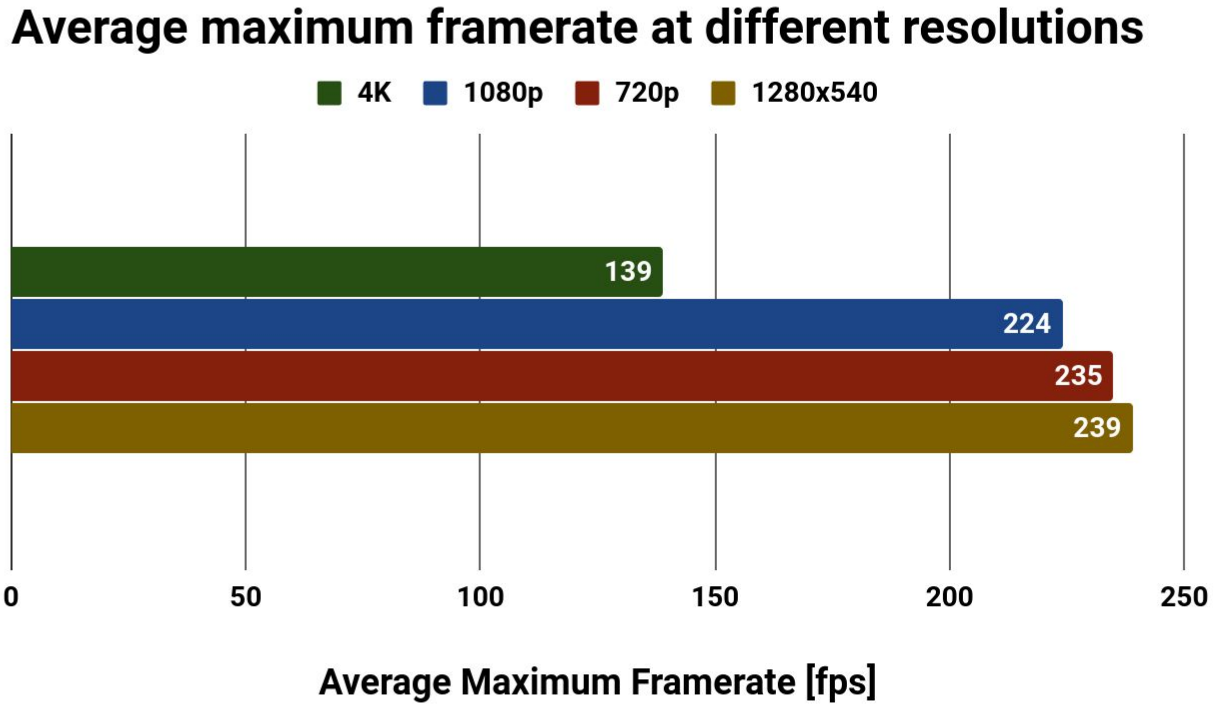
$$y_{(n,m)} = 0.5(x_1(n,m) + x_2(n,m))$$

- Stressing data transfer
- Variable input resolution
- Cudamux element
- Unified Memory allocator
- In-place=True
- Jetson TX2 platform



Fixed Algorithm / Varying Image Size

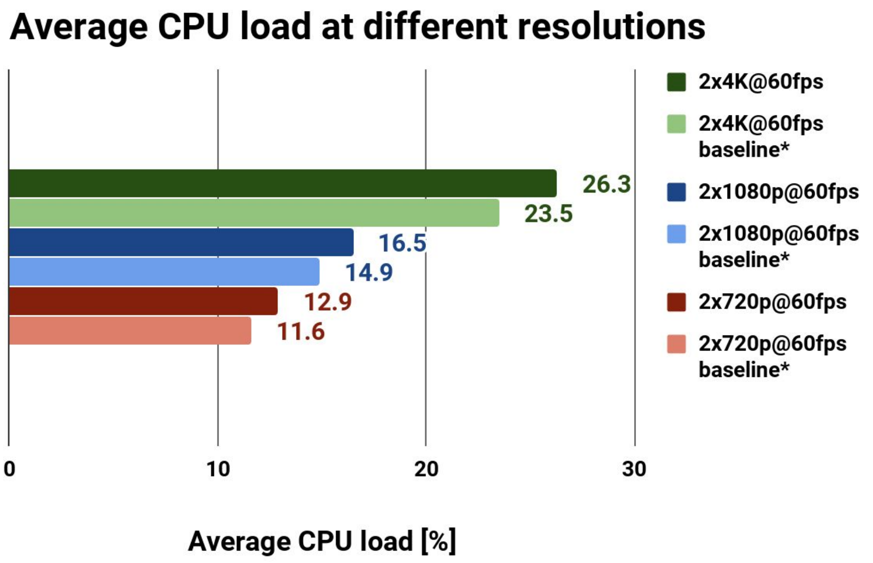
Framerate Stats



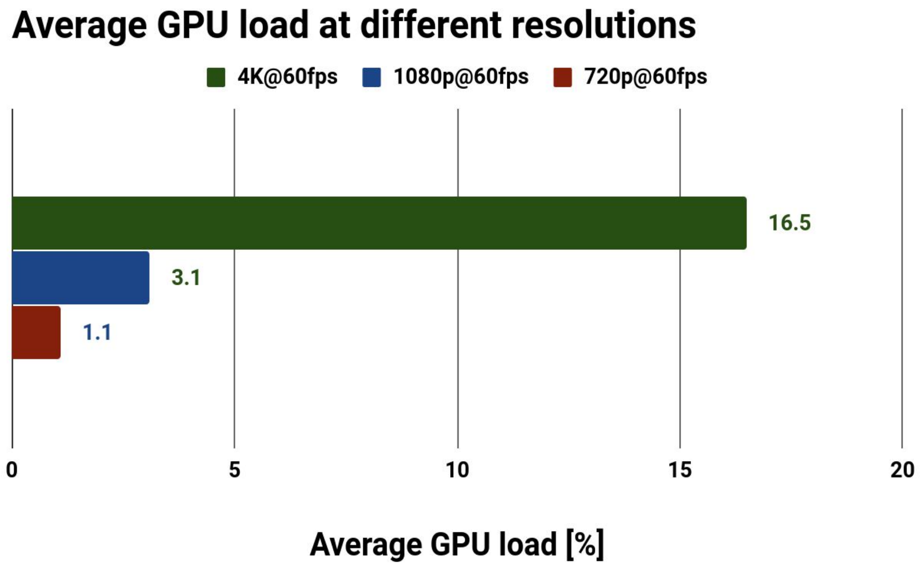
Note: Maximum Framerate limited to 240fps by the video source

Fixed Algorithm / Varying Image Size

CPU Load Stats



GPU Load Stats



*baseline = simple capture pipeline (without GstCUDA)

GstCUDA Live Demo on Jetson TX2

Sobel Filter 1080p60fps

Code equivalent :

```
gst-launch-1.0 nvcamerasrc sensor-id=2 fpsRange=60,60 ! "video/x-raw(memory:NVMM),width=1920,height=1080,framerate=60/1,format=I420" ! nvvidconv ! "video/x-raw" ! queue ! cudafilter in-place=false location=/borders.so ! queue ! nvoverlaysink
```

Resources

- GstCUDA wiki page:
  - gstcuda.ridgerun.com
- RidgeRun Website:
  - ridgerun.com
- RidgeRun Contact:
  - ridgerun.com/contact