4K Video Processing and Streaming Platform on TX1
1. Live video streaming
   - In 5 minutes

2. Bottlenecks
   - GPU
   - Kernel
Live Video Streaming

Video Sources → Mixing → Encoding → Stream

- Color Space Conversion
- 4K → 4Gbps Scaling
- Target Bandwidth
- Transport Protocol

Gbps → Mbps
Live Video Streaming - Dynamic

Video Sources → Mixing → Encoding → Stream

- Video Sources
- Mixing
- Encoding (Gbps, Mbps)
- Stream (HLS, RTP)
Classic Approach

Video Sources → Mixing → Encoding → Stream

- **FPGA**
  - Pipeline
  - Encoding IP

- **CPU**
  - Streaming Application

Fixed Implementation → Interface
Approach with TX1

- **Processing**
  - GStreamer
  - MM API
  - CPU
  - GPU
  - DMAs

- **CODECs**
  - H.264
  - H.265
  - VP8

- **Streaming**
  - HLS
  - Mpeg-
  - TS
  - RT(S)P

- **Interaces**
  - CSI
  - PCIe
  - USB
  - Ethernet

---

Figure 1 NVIDIA Tegra X1 Mobile Processor
Software Frameworks

- **GStreamer**
  - Pipeline-based Multimedia Framework
  - Very easy to use (one-liner)
  - Open-Source

- **L4T Multimedia API** (since L4T 24.2)
  - Low-level APIs for application development

- **GPU Integration**
  - CUDA
  - OpenGL / EGL

GStreamer is free software available under the terms of the LGPL license
OpenGL® and the oval logo are trademarks or registered trademarks of Silicon Graphics, Inc
# Software Stack for Streaming

## Sources
- v4l2, alsa, tcp/udp
- xvideo, overlay (omx), tcp/udp
- libargus, V4L2 API
- NVOSD

## Sinks
- V4L2, videobuf2
- DRM/KMS/FB
- ALSA
- v4l2-subdev

## Processing
- GStreamer:
  - mix, scale, convert, cuda, openGL
- Multimedia API:
  - cuda, openGL

## CODECs
- Streams:
  - rtp, rtsp, hls, mpeg-ts
- Codecs:
  - omx h264/h265, libav, mp3

## High-Level: VisionWorks/OpenCV, TensorRT, cuDNN

## Libraries
- X11
- OpenGL, EGL, Vulkan
- CUDA
- OpenMAX (omx)

## Kernel Space
- V4L2, videobuf2
- DRM/KMS/FB
- ALSA
- v4l2-subdev
- GPU Driver
- Host1x / Graphics Host

## Modules / Drivers
- Linux Kernel
- Sockets
- TCP/IP/UDP
- PCIe Ctrl
- Eth PHY

## CPU
- Vi (CSI)
- GPU
- CODECs

## HW
- Video Source
- Display Ctrl
- Convert

## Userspace
- Buffers
- Utility
- High-Level: VisionWorks/OpenCV, TensorRT, cuDNN

## Eth Driver
- Eth PHY
Kernel Driver for 4K Input

**Sources**
- GStreamer
  - v4l2, alsa, tcp/udp

**Sinks**

**Processing**
- Nvidia Jetson TX1 Development Board
- HDMI2CSI module

**CODECs**
- V4L2, videobuf2
- v4l2-subdev
- Graphics Host

**Stream**
- CPU
- VI (CSI)
- Video Source

**Kernel Space**
- Linux Kernel
  - Modules / Drivers
    - v4l2-subdev
    - Graphics Host

**User Space**
- HW
- Video Source
Modularity HW & SW

Sources
- v4l2, alsa, tcp/udp

Sinks
- xvideo, overlay (omx), tcp/udp

Processing
- GStreamer: mix, scale, convert, cuda, OpenGL

CODECs
- OMX h264/h265, libav, mp3

Stream
- rtp, rtsp, hls, mpeg-ts

Video Source

Display Ctrl

Convert

CPU

GPU

CODECs

PCle Ctrl

Eth PHY
Simple Video Streaming Pipeline
HLS

Gstreamer Pipeline

V4L2 Source → Convert → Encode H.265 → MPEG-TS Mux → HLS Sink

WebServer (lighttpd)

$ gst-launch-1.0 v4l2src !
    videoconvert !
    omxh265enc bitrate=5000000 !
    mpegtsmux !
    hlssink

playlist-location=/var/www/playlist.m3u8
location=/var/www/segment%05d.ts
playlist-root=http://192.168.0.1
Video Processing
Scaling, Mixing

Gstreamer Pipeline

V4L2 Source → Format Convert → Scale → Mix (PiP) → Render HDMI

Mixing two sources (4K and 1080p)
Video Processing
Example: Scaling, Mixing

1080p Video

4K Video

Images: CC BY-SA Wikimedia
Mixing two sources (4K and 1080p)

- **CPU**: Using *compositor* element: **1.2 FPS**

  ```
gst-launch-1.0 v4l2src ! 'video/x-raw, format=UYVY, framerate=30/1, width=3840, height=2160' ! compositor name=comp sink_0::alpha=1 sink_1::alpha=0.5 ! xvimagesink sync=false videotestsrc pattern=1 ! 'video/x-raw,format=UYVY, framerate=30/1, width=1000, height=1000' ! comp.
  ```

- **OpenGL** (*glvideomixer* & *glimagesink*): **6.8 FPS**

- Need a solution with better performance => GPU
GPU Processing
GPU Memory Access Methods

Unified Virtual Addressing

Zero Copy

Managed Memory
### GPU Processing

**PiP Test**  (GPU Data Transfer and Kernel Execution)

### Unified Virtual Addressing

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1:</td>
<td><code>cudaMemcpy()</code> to GPU **</td>
<td>12.5 ms</td>
</tr>
<tr>
<td>Step 2:</td>
<td>Execute kernel</td>
<td>9-11 ms</td>
</tr>
<tr>
<td>Step 3:</td>
<td><code>cudaMemcpy()</code> to host **</td>
<td>7.2 ms</td>
</tr>
</tbody>
</table>

**Total: 30 ms**

### Zero Copy

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1:</td>
<td><code>cudaMallocHost()</code>: Allocate memory on host**</td>
<td>-</td>
</tr>
<tr>
<td>Step 2:</td>
<td>Execute kernel</td>
<td>23.5 – 25.7 ms</td>
</tr>
</tbody>
</table>

**Total: 25 ms**

### Managed Memory

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1:</td>
<td><code>cudaMallocManaged()</code>: Allocate shared memory**</td>
<td>-</td>
</tr>
<tr>
<td>Step 2:</td>
<td>Execute kernel</td>
<td>9-11 ms</td>
</tr>
<tr>
<td>Step 3:</td>
<td>synchronize with CPU</td>
<td>0.2 ms</td>
</tr>
</tbody>
</table>

**Total: 10 ms**

* Upload 4K + 1080p, Download 4K
  
** One time only operation
**GPU Processing**

**Results**

- **PiP** pipeline achieves 30 FPS
  - Using managed memory

**Additional:**

- Consecutive kernels executed faster
Conclusion
Hardware Mapping

Color Space Conversion -> Scaling -> Picture in Picture -> H.264/H.265 Encoder

Audio/Video Mux -> Encryption -> Transport Protocol Packer -> Forward Error Correction

Video Input -> Gbps

2nd Video Source -> Ethernet Output

Audio

Recorder

GPU

HW Block

CPU
Conclusion

- Streaming Pipeline in 5 Minutes!
- GStreamer is modular and easy to use
- Video Processing Bottleneck?
  - GPU to the rescue!
Get started with video streaming now!

Blog: https://blog.zhaw.ch/high-performance/
4K Driver: https://github.com/ines-hpmm
Hardware Board: http://pender.ch/products_zhaw.shtml

tobias.kammacher@zhaw.ch
matthias.rosenthal@zhaw.ch