4K Video Processing and Streaming Platform on TX1

Tobias Kammacher
Dr. Matthias Rosenthal

Institute of Embedded Systems / High Performance Multimedia Research Group
Zurich University of Applied Sciences
1. Live video streaming
   - In 5 minutes

2. Bottlenecks
   - GPU
   - Kernel
Live Video Streaming
Consumer-level YUV420. Seconds-latency.

Video Sources → Mixing → Encoding → Stream

- Color Space Conversion
- 4K → 4Gbps Scaling
- Target Bandwidth
- Transport Protocol
Live Video Streaming - Dynamic

Video Sources → Mixing → Encoding → Stream

- HLS
- RTP

Gbps  Mbps
Classic Approach

Video Sources  Mixing  Encoding  Stream

- 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

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
- libargus, V4L2 API

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

**Processing**
- GStreamer: mix, scale, convert, cuda, openGL
- Multimedia API: cuda, openGL
- Drivers / Modules: NvVideoEncoder, NvVideoDecoder

**CODECs**
- omx h264/h265, libav, mp3
- rtp, rtsp, hls, mpeg-ts

**Stream**
- RTP, RTSP, HLS, MPEG-TS

**High-Level:** VisionWorks/OpenCV, TensorRT, cuDNN, Custom Application

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

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

**Drivers / Modules**
- Sockets
- TCP/IP/UDP
- Eth Driver

**CPU**

**HW**
- VI (CSI)
- GPU
- CODECs
- PCIe Ctrl
- Eth PHY

**Video Source**
- Display Ctrl
- Convert
Modularity HW & SW

Sources
- v4l2, alsa, tcp/udp
- xvideo, overlay (omx), tcp/udp

Sinks
- GStreamer
  - mix, scale, convert, cuda, OpenGL

Processing
- CODECs
  - omx h264/h265, libav, mp3
  - rtp, rtsp, hls, mpeg-ts

Stream
- Sources
- Sinks
- Processing
- CODECs
- Stream

VI (CSI)
- Video Source
- Display Ctrl
- Convert
- CPU
- GPU
- CODECs
- PCIe Ctrl
- Eth PHY
Kernel Driver for 4K Input

Sources

- **GStreamer**
  - `v4l2, alsa, tcp/udp`
- **Linux Kernel**
  - `V4L2, videobuf2`
- **Modules / Drivers**
  - `v4l2-subdev`
- **Graphics Host**
- **CPU**
- **VI (CSI)**
- **Video Source**

Sinks

- **Processing**
  - **CODECs**
  - **Stream**

- **Nvidia Jetson TX1 Development Board**
- **HDMI2CSI module**
Video Input

Capture HDMI sources

- 4K* requires 8 CSI lanes in our case (hardware limitation)

* 2160p30 YUV422
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
Mixing two sources (4K and 1080p)
Video Processing
Example: Scaling, Mixing

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

TX1

CPU
L2 Cache

GPU
L2 Cache

Memory Controller

DRAM 4GB

CPU Buffer

GPU Buffer

TX1

CPU
L2 Cache

GPU
L2 Cache

Memory Controller

DRAM 4GB

Shared Buffer

TX1

CPU
L2 Cache

GPU
L2 Cache

Memory Controller

DRAM 4GB

Shared Buffer
## GPU Processing
### PiP Test (GPU Data Transfer and Kernel Execution)

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

Total: 30 ms

<table>
<thead>
<tr>
<th>Zero Copy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: cudaMemcpyHost(): Allocate memory on host **</td>
<td>-</td>
</tr>
<tr>
<td>Step 2: Execute kernel</td>
<td>23.5 – 25.7 ms</td>
</tr>
</tbody>
</table>

Total: 25 ms

<table>
<thead>
<tr>
<th>Managed Memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: cudaMemcpyManaged(): Allocate shared memory **</td>
<td>-</td>
</tr>
<tr>
<td>Step 2: Execute kernel</td>
<td>9-11 ms</td>
</tr>
<tr>
<td>Step 3: 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
- Ethernet Output

- Video Input
- Audio
- 2nd Video Source
- Gbps
- Mbps

- 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