Efficient Video Processing on Embedded GPU

Tobias Kammacher
Armin Weiss
Matthias Frei

Institute of Embedded Systems
High Performance Multimedia Research Group
Zurich University of Applied Sciences (ZHAW)
1. Share Experiences
2. Benefits of Embedded GPU
3. Bottlenecks
Experience with Gstreamer on Embedded Devices

- Live Video Stream
  - HW / SW
  - Embedded + 4K
  - Drivers
Experience with Gstreamer on Embedded Devices

- Live Video Stream
  - HW / SW
  - Embedded + 4K
  - Drivers

Nvidia Jetson TX1 Development Board

4K HDMI Capture Module
Experience with Gstreamer on Embedded Devices

- **Live Video Stream**
  - HW / SW
  - Embedded + 4K
  - Drivers

- **Multi Camera Capture**
  - Debayer on GPU

- **GPU is powerful**
  - Realtime?
Experience with Gstreamer on Embedded Devices

- **Live Video Stream**
  - HW / SW
  - Embedded + 4K
  - Drivers

- **Multi Camera Capture**
  - Debayer on GPU

- **GPU is powerful**
  - Realtime?

- **Live Video Processing**
  - Computer Vision
  - Deep Learning
Embedded: Nvidia TX1/TX2

Interfaces
- CSI
- PCIe
- USB
- Ethernet

Image: nvidia.com
Embedded: Nvidia TX1/TX2

Interfaces

- CSI
- PCIe
- USB
- Ethernet

Image: nvidia.com
Embedded: Nvidia TX1/TX2

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

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

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

- **Interfaces**
  - CSI
  - PCIe
  - USB
  - Ethernet

Image: nvidia.com
Software Frameworks on TX1/TX2

• OS: Linux for Tegra (L4T) by Nvidia
  – Kernel 4.4.15
  – Video Input: V4L2 drivers (e.g. for CSI)
  – Video Output: Xorg or proprietary framebuffer

• Multimedia APIs
  – GStreamer
    • Hardware Scaling, CODECs (omx)
    • Video Input, Display
    • ISP hidden
  – L4T Multimedia API (Nvidia)
    • Video input, V4L2 API, Buffer management
  – OpenCV, Deep Learning Frameworks (TensorRT, Yolo, ..)

• GPU Integration
  – CUDA
  – OpenGL (ES) / EGL
  – Vulkan

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

**Sources**
- v4l2, alsa, tcp/udp
- xvideo, overlay (omx), tcp/udp
- libargus, V4L2 API
- Buffer utility

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

**Processing**
- **VisionWorks**
- **TensorRT**
- **OpenCV (-> AI)**
- **GStreamer**
  - mix, scale, convert, cuda, openGL
- **Multimedia API**
  - cuda, openGL

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

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

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

**Libraries**
- V4L2, videobuf2
- DRM/KMS/FB
- ALSA

**User Space**
- libargus, V4L2 API
- NVOSD
- Buffer utility

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

**Hardware**
- VI (CSI)
- GPU
- Convert
- CODECs H.264/265/VP8
- PCIe Ctrl
- Eth PHY

**CPU**
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
Video Processing
Example: Scaling, Mixing

Gstreamer Pipeline

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

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

Mixing two sources (4K and 1080p)

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

```bash
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.
```
Video Processing
Example: Scaling, Mixing

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**
Video Processing
Example: Scaling, Mixing

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
Use GPU with GStreamer

- **GStreamer Plugin**
  - From nvidia: nvivafilter
    - CUDA processing
    - NVMM frame format (Nv internal)
    - EGLImage type
    - Only 1 input and 1 output pad
  - Our own plugin (internal)
    - CUDA processing
    - Multiple input pads, 1 output pad
    - Allocate managed memory from GPU and pass to src plugin
    - Support Userptr io-mode
- Alternatives?
Unified Virtual Addressing

TX1

CPU
L2 Cache
Memory Controller

GPU
L2 Cache

DRAM 4GB

CPU Buffer
GPU Buffer
GPU Processing
GPU Memory Access Methods

Unified Virtual Addressing

Zero Copy

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

CPU ➔ GPU

CPU ➔ GPU

CPU ➔ GPU

CPU ➔ GPU

CPU ➔ GPU

GPU Processing
GPU Memory Access Methods

Unified Virtual Addressing

Zero Copy

Managed Memory
## Unified Virtual Addressing

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</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

* Upload 4K + 1080p, Download 4K
## GPU Processing

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

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

**Total:** 30 ms

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

**Total:** 25 ms

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

PiP Test  (GPU Data Transfer and Kernel Execution)

### Unified Virtual Addressing

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

Total: 30 ms

### Zero Copy

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cudaMallocHost(): Allocate memory on host**</td>
<td>-</td>
</tr>
<tr>
<td>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>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cudaMallocManaged(): Allocate shared memory**</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Execute kernel</td>
<td>9-11 ms</td>
</tr>
<tr>
<td>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
- Audio
- Ethernet Output
- Mbps
- 2nd Video Source

- GPU
- HW Block
- CPU
Conclusion

• Live 4K on Embedded

• GPU and HW-accelerated blocks
  – Enable Desktop -> Embedded

• Bottlenecks and Solutions
  – Allocate GPU Managed Memory for Capture
  – Gst GPU Plugin
Get started with embedded GPU now!

Blog: [https://blog.zhaw.ch/high-performance/](https://blog.zhaw.ch/high-performance/)
4K Drivers: [https://github.com/ines-hpmm](https://github.com/ines-hpmm)
Hardware Board: [http://pender.ch/products_zhaw.shtml](http://pender.ch/products_zhaw.shtml)

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