MAPPING VIDEO CODECS TO HETEROGENEOUS ARCHITECTURES

Mauricio Alvarez-Mesa | Techische Universität Berlin - Spin Digital | MULTIPROG 2015
Video Codecs

- 70% of internet traffic will be video in 2018 [CISCO]
- Video compression: significant improvements over the last years
- Growing demand/offer for higher quality: HD to UHD
- Video codecs require more and more computing power
Like them or not: they are available

- CPUs + GPUs + DSPs + HW accelerators + ...
Mapping: the Complete Application

− Video Player: Decoding + Rendering
Mapping: Where to Execute What?

**Figure**: Qualcomm Snapdragon 810 SoC
Outline

Introduction

Hardware Acceleration of Video Decoding

OpenCL Acceleration of Video Decoding

CPU Decoding + GPU Rendering

Conclusions
Hardware Accelerators

Mapping:

- Decoder in HW: ASIC for video decoding
- Rendering in GPU
High-level Accelerators

Mapping:

- Decoder in HW: ASIC for video decoding
- Rendering in GPU

Pros

- Low-power: 4K H.265 < 100 mW

Cons

- Flexibility: more than one codec
- Programmability issues
**Multiple APIs: accessing HW modules from SW player**

“Supported codecs depend on the chosen API: which is selected at runtime depending on what is available on the system”

Gstreamer Manual

<table>
<thead>
<tr>
<th>API</th>
<th>Vendor</th>
<th>MPEG2</th>
<th>MPEG4</th>
<th>H.264</th>
<th>VC1</th>
<th>H.265</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAAPI</td>
<td>Intel</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>VDPAU</td>
<td>NVidia</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>XVBA</td>
<td>AMD</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>?</td>
</tr>
<tr>
<td>DXVA</td>
<td>Microsoft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>VDA</td>
<td>Apple</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>?</td>
</tr>
</tbody>
</table>
Outline

Introduction

Hardware Acceleration of Video Decoding

OpenCL Acceleration of Video Decoding

CPU Decoding + GPU Rendering

Conclusions
### Mapping

- **Decoder**: accelerate decoding on the GPU using OpenCL
- **Rendering**: in GPU
OpenCL Video Decoding

Mapping

- Decoder: accelerate decoding on the GPU using OpenCL
- Rendering: in GPU

Pros

- Programmability
- Compatibility: write once, and compile/run everywhere (isn’t it?)
### OpenCL Video Decoding

#### Mapping
- Decoder: accelerate decoding on the GPU using OpenCL
- Rendering: in GPU

#### Pros
- Programmability
- Compatibility: write once, and compile/run everywhere (isn’t it?)

#### Cons
- Programming model mismatch
- Overheads
H.264 Video Decoding with GPUs

- H.264/AVC: one of the most widely used video codecs

<table>
<thead>
<tr>
<th>Kernels</th>
<th>Entropy Dec.</th>
<th>Inverse DCT</th>
<th>Intra-Pred.</th>
<th>Motion Comp.</th>
<th>Deblocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelism</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Divergence</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>
H.264 Inverse Transform on GPUs

Kernel Only

- Speedups over scalar execution in CPU
- Significant speedups: over 25×
H.264 Inverse Transform on GPUs

Kernel + Memory Transfers + OpenCL runtime

- Entire application slower than CPU + SIMD with 1 core

H.264 Motion Compensation on GPUs

- discrete GPU attains the highest performance, 1.7 speedup
- significant performance penalty when including the overhead
H.264 Motion Compensation on GPUs

B. Wang et al. Parallel H.264/AVC Motion Compensation for GPUs using OpenCL. IEEE TCSVT.
H.264: Complete Application

- GPU + frame-level pipelining: faster than single threaded CPU
- Required complete application redesign
HEVC/H.265 Video Coding Standard

- HEVC/H.265: 50% better compression compared to H.264/AVC
- Add high-level parallelization tools: WP, Tiles \(\rightarrow\) good for CPUs
- Remove dependencies in deblocking filter
- Add another filter: SAO

<table>
<thead>
<tr>
<th>Kernels</th>
<th>Entropy Dec.</th>
<th>Inverse T</th>
<th>Intra-Pred.</th>
<th>Motion Comp.</th>
<th>De-blocking</th>
<th>SAO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelism</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Divergence</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
H.265 In-loop Filters on Discrete GPUs

Kimono 1080p 240 Frames, GTX750Ti + i2500K

Time [s]

Decoder Version

Baseline
Baseline+Decouple
CPUOpenCL
GPUOpenCL

SAO
DEBLOCK
BUF_COPY
OTHERS
H.265 In-loop Filters on Integrated GPUs

Kimono 1080p 240 Frames, i5-4570R (iGPU: HD5200)

- Baseline
- Baseline+Decouple
- CPUOpenCL
- GPUOpenCL

Decoded Version

Time [s]

- SAO
- DEBLOCK
- BUF_COPY
- OTHERS

Mapping Video Codecs to Heterogeneous Architectures | M. Alvarez-Mesa | Multiprog 2015
Slide 19
Kimono 1080p 240 Frames, Cortex A15+A7 (iGPU: MaliT628)

Time [s]

<table>
<thead>
<tr>
<th>Decoder Version</th>
<th>Baseline</th>
<th>Baseline+Decouple</th>
<th>CPUOpenCL</th>
<th>GPUOpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time [s]</td>
<td>10</td>
<td>10</td>
<td>30</td>
<td>50</td>
</tr>
</tbody>
</table>

- SAO
- DEBLOCK
- BUF_COPY
- OTHERS
OPenCL Decoding: Lessons Learned

- Programming model mismatch:
  - not all kernels can be offloaded
  - for kernels that can be offloaded: data Locality loss
- Acceleration of kernels but not of complete application
- Overheads:
  - CPU GPU memory transfers
  - OpenCL runtime
- Effects on rendering of using the GPU for decoding not analyzed (yet)
Outline

Introduction

Hardware Acceleration of Video Decoding

OpenCL Acceleration of Video Decoding

CPU Decoding + GPU Rendering

Conclusions
## CPU Decoding

### CPU optimizations

- Multithreading: C++11 threads
- SIMD: compiler intrinsics

### Pros

- Programmability
- High performance: in high performance CPUs

---

Cons

- Low power efficiency: see Chi et al. TACO Jan 2015.
- Low performance: in low power CPUs
CPU Decoding

CPU optimizations

- Multithreading: C++11 threads
- SIMD: compiler intrinsics

Pros

- Programmability
- High performance: in high performance CPUs

Cons

- Low power efficiency: see Chi et al. TACO Jan 2015.
- Low performance: in low power CPUs
## 4K CPU Decoding Performance

<table>
<thead>
<tr>
<th></th>
<th>1-Thread</th>
<th></th>
<th>2-Threads</th>
<th></th>
<th>4-Threads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fps</td>
<td>sp.</td>
<td>fps</td>
<td>sp.</td>
<td>fps</td>
<td>sp.</td>
</tr>
<tr>
<td>Haswell</td>
<td>37.8</td>
<td>1.00</td>
<td>73.4</td>
<td>1.94</td>
<td>134.9</td>
<td>3.57</td>
</tr>
<tr>
<td>Bay Trail</td>
<td>6.2</td>
<td>1.00</td>
<td>12.2</td>
<td>1.95</td>
<td>23.6</td>
<td>3.79</td>
</tr>
<tr>
<td>Cortex-A9</td>
<td>2.8</td>
<td>1.00</td>
<td>5.3</td>
<td>1.90</td>
<td>8.6</td>
<td>3.05</td>
</tr>
<tr>
<td>Cortex-A15</td>
<td>6.0</td>
<td>1.00</td>
<td>11.4</td>
<td>1.91</td>
<td>19.9</td>
<td>3.33</td>
</tr>
</tbody>
</table>

- High performance CPUs fast enough for 4Kp50
- Low power CPUs not fast enough for 4Kp25

### References

- Chi et al. SIMD acceleration for HEVC Decoding, IEEE TCSVT. 2015
- Chi et al. Parallel Scalability and Efficiency of HEVC Parallelization Approaches. IEEE TCSVT. Dec 2012
- Chi et al. Parallel HEVC Decoding on Multi- and Many-core Architectures. JSPS 2013
## GPU Rendering using OpenGL

### Pros

- GPU should be efficient for simple 2D rendering
- Color conversion is just matrix vector multiplication
- GPUs have HW units for color conversion
- OPenGL is a mature language, well supported in many platforms
## GPU Rendering using OpenGL

### Pros
- GPU should be efficient for simple 2D rendering
- Color conversion is just matrix vector multiplication
- GPUs have HW units for color conversion
- OPenGL is a mature language, well supported in many platforms

### Cons
- 2D rendering is complex
- Color Conversion is not the bottleneck
- OPenGL has extensions, and has no performance guarantees
Video Rendering Stages

- **Texture Uploading**: upload/transform image from CPU to GPU
- **Color Model Conversion**: YUV into RGBA conversion
- **Displaying**: from framebuffer to screen
4K Video Play: Discrete GPU

4K Playback on GTX750-Ti Linux

Performance (millisecond/frame)

- Decoding + Rendering
- Decoding
- Rendering
- Texture Uploading
- CC + Displaying
- Color Conversion

Video Formats
- YUV420 8-bit 11.9MB
- YUV420 10-bit 23.7MB
- YUV422 10-bit 31.6MB
- YUV444 10-bit 47.5MB
4K Video Play: Integrated GPUs

4K Playback on HD4600 Linux

- Decoding + Rendering
- Decoding
- Rendering
- Texture Uploading
- CC + Displaying
- Color Conversion

Video Formats:
- YUV420 8-bit 11.9MB
- YUV420 10-bit 23.7MB
- YUV422 10-bit 31.6MB
- YUV444 10-bit 47.5MB

Mapping Video Codecs to Heterogeneous Architectures

M. Alvarez-Mesa

Multiprog 2015
Integrated GPUs + Linear Texture

4K Texture Uploading on HD4600 Win8.1

![Graph showing performance (millisecond/frame) vs. video formats: YUV420 8-bit (11.9MB), YUV420 10-bit (23.7MB), YUV422 10-bit (31.6MB), YUV444 10-bit (47.5MB). The graph compares different methods: glTexImage2D, PBO + memcpy, PBOMAP, INTEL_map_texture.]
Integrated GPUs + Linear Texture

4K Texture Uploading on HD4600 Win8.1

4K Playback on HD4600 Win8.1

Performance (millisecond/frame) vs. Video Formats

- glTexSubImage2D
- PBO + memcpy
- PBOMAP
- INTEL_map_texture

Mapping Video Codecs to Heterogeneous Architectures

M. Alvarez-Mesa
Multiprog 2015
Slide 29
### Discrete GPUs

- Performance drop: sharing memory for PCIe copy
- Can be solved by using integrated GPUs: remove copies
Decoding + Rendering: lessons learned

<table>
<thead>
<tr>
<th>Discrete GPUs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Performance drop: sharing memory for PCIe copy</td>
<td></td>
</tr>
<tr>
<td>- Can be solved by using integrated GPUs: remove copies</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integrated GPUs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Texture uploading takes more time than decoding</td>
<td></td>
</tr>
<tr>
<td>- Can be solved with an OpenGL extension</td>
<td></td>
</tr>
</tbody>
</table>
Decoding + Rendering: lessons learned

Discrete GPUs

- Performance drop: sharing memory for PCIe copy
- Can be solved by using integrated GPUs: remove copies

Integrated GPUs

- Texture uploading takes more time than decoding
- Can be solved with an OpenGL extension

OpenGL extensions

- INTEL_map_texture: remove memory layout conversion
- Performance loss due to memory contention
Outline

Introduction

Hardware Acceleration of Video Decoding

OpenCL Acceleration of Video Decoding

CPU Decoding + GPU Rendering

Conclusions
## Conclusions

**Video codecs and parallel architectures: conflicting objectives**

- Video codecs: remove redundancy → introduce dependencies
- Parallelism: remove dependencies → reduce compression
## Conclusions

**Video codecs and parallel architectures: conflicting objectives**

- Video codecs: remove redundancy → introduce dependencies
- Parallelism: remove dependencies → reduce compression

**Solve the system problem**

- Complete video decoding: not just kernels
- Rendering: is not just matrix multiplication
- Video player is Decoding + Rendering
Questions?

– Acknowledgements: Biao Wang, Haifeng Gao, Chi Ching Chi
– More info:
  – http://www.aes.tu-berlin.de/