NVIDIA VIDEO TECHNOLOGIES
Abhijit Patait, 3/26/2018
AGENDA

NVIDIA Video Technologies Overview
Video Codec SDK Updates
Perf/Quality Optimization
Benchmarks
Roadmap
NVIDIA VIDEO TECHNOLOGIES
VIDEO CODEC SDK
A comprehensive set of APIs for GPU-accelerated video encode and decode

**NVENCODE** API for video encode acceleration

**NVDECODE** API for video & JPEG decode acceleration (formerly called NVCUVID API)

Independent of CUDA/3D cores on GPU for pre-/post-processing

- Gamestream
- Cloud transcoding
- Remote desktop & visualization
- Intelligent video analytics
- Video archiving
- Video editing
NVIDIA VIDEO TECHNOLOGIES

SOFTWARE

VIDEO CODEC SDK
Video Encode and Decode for Windows and Linux
CUDA, DirectX, OpenGL interoperability

CUDA TOOLKIT
APIs, libraries, tools, samples

cuDNN, TensorRT, cuBLAS, cuSPARSE

NVIDIA DRIVER

HARDWARE

NVENC
Video encode
H.264 MPEG-4/AVC
H.265 HEVC

NVDEC
Video decode
H.264 MPEG-4/AVC
H.265 HEVC
 VP8
VP9

CUDA
High-performance computing on GPU
NVIDIA GPU VIDEO CAPABILITIES

**Decode HW***
- Formats:
  - MPEG-2
  - VC1
  - VP8
  - VP9
  - H.264
  - H.265
  - Lossless
- Bit depth:
  - 8/10/12 bit
- Color**
  - YUV 4:2:0
- Resolution
  - Up to 8K***

**Encode HW***
- Formats:
  - H.264
  - H.265
  - Lossless
- Bit depth:
  - 8 bit
  - 10 bit
- Color**
  - YUV 4:4:4
  - YUV 4:2:0
- Resolution
  - Up to 8K***

---

* See support diagram for previous NVIDIA HW generations
** 4:2:2 is not natively supported on HW
*** Support is codec dependent
VIDEO CODEC SDK UPDATE
VIDEO CODEC SDK UPDATE

SDK 6.0
- ARGB
- Quality+
- Dec+Enc
- ME-only

2015

SDK 7.x
- Pascal
- 10-bit encode
- FFmpeg
- ME-only for VR
- Quality++

2016

SDK 8.0
- 10-bit transcode
- 10/12-bit decode
- OpenGL
- Dec. optimizations
- WP, AQ, Enc.
- Quality

2017

SDK 8.1
- B-as-ref
- QP/emphasis map
- 4K60 HEVC encode
- Reusable classes & new sample apps

Q1 2018

SDK 8.2
- Decode + inference
- optimizations

Q2 2018
B-FRAMES AS REFERENCE

Non-ref B-frames

➢ Improved visual quality - up to 0.6 dB PSNR (BD-PSNR = 0.3 dB)
➢ Negligible performance penalty
➢ Ensure decoder support

B-frames as reference
WITHOUT B-AS-REF

1080p @3 Mbps
WITH B-AS-REF

1080p @3 Mbps
Problem

- Desktop content is challenging to encode
- Thin-line text, wireframes, high-detail textures
- If severely bitrate constrained, recovery is difficult without IDR.
- QP modulation requires knowledge of complexity
  - Rate control in NVENC firmware
Original Image
EMPHASIS MAP
Region of Interest Encoding

Solution

➢ Identify “high-detail” areas within the captured image (NVFBC)
➢ Provide feedback to encoder to treat these areas differently (NVENC)
EMPHASIS MAP
Region of Interest Encoding

- Generated by NVFBC
- Interpreted by NVENC as ΔQP
- Encoder translates to ΔQP
- ΔQP depends on absolute QP

5 = High detail areas
0 = Low detail areas

5 5 4 5 3 2 1 0
5 5 5 3 3 2 2 0
5 5 4 4 2 1 0 0
3 2 4 3 2 1 1 2
1 1 0 3 2 4 0 0

--- --- --- --- --- - -
--- --- --- --- --- - -
--- --- --- --- --- - -
-- -- -- -- -- -- --
-- -- -- -- -- -- --
-- -- -- -- -- -- --
REDESIGNED SDK SAMPLES
Reusable Encoder/Decoder Classes

➢ Reusable base classes, easy-to-understand, end-user focused
➢ Sample apps re-designed
➢ Encode base classes: NvEncoderD3D9, NvEncoderD3D11, NvEncoderCUDA, NvEncoderD3GL
➢ Decode base class: NvDecoder
➢ Abstraction over low-level enc/dec APIs
  ➢ init(), run(), destroy()
➢ FFmpeg demux
<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppDec</td>
<td>Basic Decoding</td>
</tr>
<tr>
<td>AppDecD3D</td>
<td>Decode and Display using D3D9 and D3D11</td>
</tr>
<tr>
<td>AppDecGL</td>
<td>Decode and Display using OpenGL</td>
</tr>
<tr>
<td>AppDecImage</td>
<td>Decoding and Color Conversion to a specific format (BGRA, BGRA64)</td>
</tr>
<tr>
<td>AppDecLowLatency</td>
<td>Low-latency decode</td>
</tr>
<tr>
<td>AppDecMem</td>
<td>Decode from memory buffer</td>
</tr>
<tr>
<td>AppDecMultiInput</td>
<td>Use-case: Surveillance, multiple videos on screen</td>
</tr>
<tr>
<td>AppDecPerf</td>
<td>Multi-threaded, perf measurement</td>
</tr>
</tbody>
</table>
REDESIGNED SDK SAMPLES

Encode Applications

- **AppEncCUDA**: Encoding CUDA surfaces
- **AppEncD3D9**: Encoding using D3D9 surfaces
- **AppEncD3D11**: Encoding using D3D11 surfaces
- **AppEncDec**: Encoding & decoding in different threads, HDR streaming
- **AppEncLowLatency**: Low-latency encode, intra-refresh, slices etc.
- **AppEncME**: ME-only mode
- **AppEncPerf**: App for Encoder performance measurement
- **AppEncQual**: Encoding & quality measurement (PSNR)
OPTIMIZATION STRATEGIES
OPTIMIZATION STRATEGIES

General Guidelines

➢ Minimize PCIe transfers
  ➢ Eliminate, if possible
  ➢ Use CUDA for video pre-/post-processing
➢ Multiple threads/processes to balance enc/dec utilization
  ➢ Monitor using nvidia-smi: nvidia-smi dmon -s uc -i <GPU_index>
  ➢ Analyze using GPUView on Windows
➢ Minimize disk I/O
➢ Optimize encoder settings for quality/perf balance
SW TRANSCODE

```
ffmpeg -c:v h264 -i input.mp4 -c:a copy -c:v h264 -b:v 5M output.mp4
```
SW TRANSCODE + SCALE

ffmpeg -c:v h264 -i input.mp4 -vf scale=1280:720 -c:a copy -c:v h264 -b:v 5M output.mp4

System Memory

Bitstream → SW Decode → YUV

Preprocess (e.g. scaling) → YUV

YUV → SW Encode → Bitstream

29 fps*

*1:2 transcode, fps per session
4 GHz Intel i7-6700K
GPU UNOPTIMIZED TRANSCODE

```
ffmpeg -vsync 0 -c:v h264_cuvid -i input.mp4 -c:a copy -c:v h264_nvenc -b:v 5M output.mp4
```

*288 fps*

*1:2 transcode, fps per session
GP104 GPU*
**GPU UNOPTIMIZED TRANSCODE + CPU SCALE**

```
ffmpeg -vsync 0 -c:v h264_cuvid -i input.mp4 -c:a copy -vf scale=1280:720 -c:v h264_nvenc -b:v 5M output.mp4
```
HIGH-PERF GPU OPTIMIZED TRANSCODE

ffmpeg -vsync 0 -hwaccel cuvid -c:v h264_cuvid -i input.mp4 -c:a copy -vf scale_npp=1280:720 -c:v h264_nvenc -b:v 5M output.mp4

*1:2 transcode, fps per session
GP104 GPU
HIGH-PERF GPU OPTIMIZED TRANSCODE

ffmpeg -vsync 0 -hwaccel cuvid -c:v h264_cuvid -resize 1280x720 -i input.mp4 -c:a copy -c:v h264_nvenc -b:v 5M output.mp4

*1:2 transcode, fps per session
GP104 GPU
FFMPEG VIDEO TRANSCODING

Tips

➢ Look at FFmpeg users’ guide in NVIDIA Video Codec SDK package
➢ Use \texttt{–hwaccel} keyword to keep entire transcode pipeline on GPU
➢ Run multiple 1:N transcode sessions to achieve $M:N$ transcode at high perf
CUDA FILTERS IN FFmpeg

- resize option with NVDEC (e.g. -c:v h264_cuvid -resize 1280x720 ...)
- scale_npp: Built-in CUDA library filters
- Custom CUDA filter examples in FFmpeg
  - scale_cuda
  - thumbnail_cuda
  - Build your own using above as guide
- If you must use CPU and GPU filters, minimize PCIe x’fers
MIXING CPU & GPU FILTERS

Fade (CPU) + Scale (GPU)

Why doesn’t this work?

```bash
ffmpeg.exe -y -c:v h264_cuvid -i input.264 -vf "fade, scale_npp=1280:720" -c:v h264_nvenc output.264
```

This works

```bash
ffmpeg.exe -y -c:v h264_cuvid -i input.264 -vf "fade, hwupload_cuda, scale_npp=1280:720" -c:v h264_nvenc output.264
```
MIXING CPU & GPU FILTERS

Scale (GPU) + Fade (CPU)

Why doesn’t this work?

ffmpeg.exe -y -c:v h264_cuvid -i input.264 -vf "hwupload_cuda, scale_npp=1280:720, hwdownload, fade" -c:v h264_nvenc output.264

One solution

ffmpeg.exe -y -c:v h264_cuvid -i input.264 -vf "hwupload_cuda, scale_npp=1280:720, hwdownload, format_nv12, fade" -c:v h264_nvenc output.264

Optimal solution

ffmpeg.exe -y -hwaccel cuvid -c:v h264_cuvid -i input.264 -vf "scale_npp=1280:720, hwdownload, format_nv12, fade" -c:v h264_nvenc output.264
OPTIMIZATION TIPS

➢ Write your own CUDA filters

➢ **Combine CUDA filters**; e.g. scaling + color space conversion in a single filter

➢ For systems with multiple CPU sockets, avoid accesses to *local* system of one CPU from another CPU. Find the local **NUMA** node and **localize** the storage *per CPU*.
BENCHMARKS
P4: 5X MORE H.264 ENCODE THAN 2S CPU SERVER
Up to 5x more throughput, up to 10x better efficiency at ~ quality

**H.264 hq Encode Throughput (Streams)**
- Tesla P4
- Dual Intel Xeon E5-2660v3 @ 2.6 GHz

**H.264 hq Encode Efficiency (Streams / Watt)**

**H.264 hq Encode Quality (PSNR YUV)**
P4: REAL-TIME HEVC 4K60 ENCODE

Up to 15x more throughput, up to 30x better efficiency at ~ quality

H.265 hq Encode Throughput
(Streams)

H.265 hq Encode Efficiency
(Streams / Watt)

H.265 hq Encode Quality
(PSNR YUV)

Tesla P4
Dual Intel Xeon E5-2660v3 @ 2.6 GHz
GPU ENCODE REDUCES CAPEX 7X, OPEX 17X

Transcoding 20,000 720p30 Streams + 20,000 1080p30 H.264 Streams, hqslow

Nodes Needed

7,609

243

CapEx

$33.6M

$4.8M

4 Year OpEx

$49.9M

$2.9M

CPU Nodes

2xE5-2660v3, 128GB DDR4, 512GB SSD, 25 GE. Node price including core network $4500

GPU Nodes

2xE5-2660v3, 8xP4 PCIe, 128GB DDR4, 512GB SSD, 25 GE
ROADMAP
Video Codec SDK 8.2

➢ Q2 2018
➢ Decode + inference optimizations
➢ Reconfigure decoder without reinitialization
   ➢ No init time, reuse context, lowers memory fragmentation
➢ Report decoder errors
   ➢ Inference can continue up to error slice
➢ HEVC I-frame only decoding (H.264 already supported) - Q3 2018
   ➢ Lower memory, IVA use-case
RESOURCES


FFmpeg GIT: https://git.ffmpeg.org/ffmpeg.git

FFmpeg builds with hardware acceleration: http://ffmpeg.zeranoe.com/builds/

Video SDK support: video-devtech-support@nvidia.com

Video SDK forums: https://devtalk.nvidia.com/default/board/175/video-technologies/

Connect with experts (CE8107): Today, 26th March at 3:00 pm