MULTIMEDIA PROCESSING
Real-time H.264 video enhancement by using AMD APP SDK

Wei-Lien Hsu
AMD
SMTS

Gongyuan Zhuang
AMD
MTS
OUTLINE

- Motivation
- OpenDecode
- Video deblurring algorithms
- Acceleration by clAmdFft
- Experimental results
- Conclusion and Remarks
MOTIVATION

- In order to support high-quality video playback with very low power consumption, AMD Unified Video Decoder (UVD) technology has completely offloaded H.264/AVC, VC-1, MPEG2 and MPEG4 video codec tasks from CPU.
- UVD engine can be accessed by using DXVA API on Windows or XVBA API on Linux.
- For OpenCL developers to use the hardware video decoder, AMD APP SDK introduces OpenDecode API. (OpenDecode can be directly used my application, does not require plug-in)
- OpenDecode is fully interoperable with the OpenCL, post-processing of the decompressed video can be efficiently performed directly inside the GPU memory without the need to copy data back and forth between CPU and GPU.

This presentation demonstrates a working example for using
- OpenDecode to decode H.264 video content
- OpenCL™ FFT kernel library routines to enhance the visual quality in shader processors.
**OPENDECODE**

- OpenDecode is part of AMD OpenVideo implementation that provides a way for OpenCL video applications to leverage the powerful UVD unit inside AMD Radeon graphics cards.

- **AMD OpenVideo Decode API design goals:**
  - Be interoperable with the OpenCL API, which allows for shared surface memory between the two domains
  - Provide platform-independent video codec functions
  - Support multiple video codec standards
  - Support full decoding acceleration
  - Extendable to support hardware encoding functions in the future
Decoding framework

- Ineroperability of OpenCL
  - The AMD extensions to OpenCL provide functions for obtaining the platform context handle for creating:
    - OpenDecode sessions
    - Output surface buffers
    - Command buffer queues
    - Kernel objects
  - OpenDecode API is a thin layer, it is designed to pass parameters to multimedia driver (in CL-UVDLib) for executing decoding tasks.
Decoding pipeline

- Decompression takes place on the GPU, it operates concurrently with CPU, GPU compute and GPU graphics.
OPENDECODE

The essential steps in performing a decode by using the Open Video Decode API can be summarized as follows:

- Initialization
- Context creation
- Session creation
- Decode execution
- Process Decode Output
- Session and context destruction
**OPENDECODE**

**Step 1: Initialization**

- Use the OpenDecode functions to get the *Device Info. and Capabilities*:
  - **OVDecodeGetDeviceInfo**
    - Obtains the information about the device (including number of devices, device ID, Cap size of each device)
  - **OVDecodeGetDeviceCaps**
    - Obtains the supported output format and the compression profile.
      The application then can verify that these values support the requested decode

```c
OVDecodeGetDeviceInfo(&numDevices, deviceInfo);
for(unsigned int i = 0; i < numDevices; i++){
    ovdecode_cap *caps = new ovdecode_cap[deviceInfo[i].decode_cap_size];
    status = OVDecodeGetDeviceCap(deviceInfo[i].device_id,deviceInfo[i].decode_cap_size,caps);
    for(unsigned int j = 0; j < deviceInfo[i].decode_cap_size; j++){
        if(caps[j].output_format == OVD_NV12_INTERLEAVED_AMD &&
           caps[j].profile == OVD_H264_HIGH_41){ ovDeviceID = deviceInfo[i].device_id; break;}
    }
}
```
OPENDECODE

Step 2: Creating the Context

- Create a context for the decode by using `clCreateContext`:
  - Initializes the OpenDecode function pointers
  - Creates callbacks functions
  - Allocates buffers

```c
// Use clGetPlatformID() to obtain a platform_id provided by “Advanced Micro Devices, Inc.”
 intptr_t properties[] = {CL_CONTEXT_PLATFORM, (cl_context_properties)platform_id,
                          CL_WGL_HDC_KHR,      (intptr_t)hdc,
                          CL_GL_CONTEXT_KHR,   (intptr_t)hrc, 0};

ovdContext = clCreateContext( properties,
                               1,
                               &clDeviceID,
                               0,
                               0,
                               &err);
```
OPENDECODE

Step 3: Creating the Session

- Create a decode session using `OVDecodeCreateSession` based on input/output format and video resolution:
  - Call UVDLib to creates a decode driver session (initializes parameters based on input data)
  - Performs the internal resource allocation (creates comand buffers and allocates surface memory)
  - Initializes UVD configuration, register UVD client and send Start Decoder Session message to firmware.

```c
ovdecode_profile profile = OVD_H264_HIGH_41;
ovdecode_format oFormat = OVD_NV12_INTERLEAVED_AMD;
oWidth = video_width;
oHeight = video_height;
session = OVDecodeCreateSession( ovdContext,
oDeviceID,
profile,
oFormat,
oWidth,
oHeight);
```
OPENDECODE

Step 4: Decoding

- Decode execution goes through OpenCL and starts the UVD decode function

```c
output surface = clCreateBuffer((cl_context)ovdContext, CL_MEM_READ_WRITE, host_ptr_size,
                                NULL, &err);

OVresult res   = OVDecodePicture(session,
                                &picture_parameter,//profile, level, resolution
                                &pic_parameter_2, pic_parameter_2_size,// codec dependent
                                &bitstream_data, bitstream_data_read_size,
                                slice_data_control, slice_data_control_size,//multi-slice
                                output_surface,
                                num_event_in_wait_list, NULL,
                                &eventRunVideoProgram, 0);

clWaitForEvents(1, (cl_event *)&(eventRunVideoProgram));
```

- **OVDecodePicture Latency:** ~5ms for 720x480; ~12ms for 1920x1080
Step 5: Process Decode Output

- The decompressed video data can be read to CPU or post-processed by using the GPU compute shader units, then displayed by OpenGL rendering pipeline
  
  - Read nv12 data from video buffer to host buffer
    
    ```c
    clEnqueueReadBuffer(cl_cmd_queue,
                        nv12_buffer, CL_TRUE, 0,
                        oWidth*oHeight*1.5, g_decoded_frame, 0, NULL, 0);
    ```
  
  - Post-process
    
    ```c
    clEnqueueNDRangeKernel(cl_cmd_queue,
                           nv12_to_rgb_kernel, 2, 0, // YUV to RGB conversion kernel
globalThreads,
localThreads,0, 0, 0);
    ```
**OPENDECODE**

**Step 6: Destroying Session and Context**

- The final step is to release the resources and close the decode session
  - `OVDecodeDestroySession`
    - Frees allocation of resources needed for the decode session and set the UVD clock to idle state
  - `clReleaseContext`
    - Frees driver session and all internal resources

```c
clReleaseMemObject((cl_mem)output_surface);
OVDecodeDestroySession(session);
clReleaseContext((cl_context)ovdContext);
```
VIDEO DEBLURRING ALGORITHMS

- Video/Image enhancement by using Deblurring Algorithms
  - Aims to remove undesired artifacts in images, including blurring introduced by an optical system, a motion blur, and a noise from electronic sources
  - Removing blur leads to deconvolution techniques
  - The complex deconvolution processing methods can be very time consuming and their computation can be challenging even for recent computer hardware
  - This presentation demonstrates a detailed performance evaluation of deconvolution algorithms with respect to their OpenCL acceleration
**VIDEO DEBLURRING ALGORITHMS**

  - A simple frequency domain deconvolution method
    - Assume a shift-invariant blur model with independent additive noise, which is given by \( y(m,n) = h(m,n)^*x(m,n) + u(m,n) \)
    - For known blurring and noise models, Wiener Filter only requires 2 FFTs for convolving each color channel with the inverse filter.
    - However, its quality is usually much worse than sophisticated deconvolution methods

\[
\hat{X}(k, l) = G(k, l)Y(k, l)
\]

We can choose \( G(k, l) \) so that we minimize

\[
E[|X(k, l) - G(k, l)Y(k, l)|^2]
\]

\[
G(k, l) = \frac{H(k, l)}{|H(k, l)|^2 + S_u(k, l)/S_x(k, l)}
\]

where \( S_x(k, l) \) = signal power spectrum and \( S_u(k, l) \) = noise power spectrum
VIDEO DEBLURRING ALGORITHMS

- Krishnan and Fergus’s algorithm
  - Recently Dilip Krishnan and Rob Fergus have provided a fast deconvolution algorithm that yields high quality results
  - This method uses a Hyper-Laplacian image prior to regularizing the deconvolution problem
  - The resulting optimization problem is solved by using alternating minimization in conjunction with a half-quadratic penalty
  - While one sub-problem is to calculate the deblurred image by FFT, the other sub-problem is to find an optimal auxiliary variable $W$
  - This method has been shown to be efficiently accelerated by GPU programming
VIDEO DEBLURRING ALGORITHMS

- The complete deconvolution algorithm using Hyper-Laplacian Priors
  - refer to “Fast Image Deconvolution using Hyper-Laplacian Priors” by Dilip Krishnan and Rob Fergus
  

Algorithm 1 Fast image deconvolution using hyper-Laplacian priors

```
Require: Blurred image y, kernel k, regularization weight λ, exponent α (≠0)
Require: β regime parameters: β₀, βᵢⁿᶜ, βₘᵃₓ
Require: Number of inner iterations T.
1: β = β₀, x = y
2: Precompute constant terms in Eqn. 4.
3: while β < βₘᵃₓ do
4:     iter = 0
5:     for i = 1 to T do
6:         Given x, solve Eqn. 5 for all pixels using a LUT to give w
7:         Given w, solve Eqn. 4 to give x
8:     end for
9:     β = βᵢⁿᶜ · β
10: end while
11: return Deconvolved image x
```
VIDEO DEBLURRING ALGORITHMS

- **W-sub problem**
  - For a fixed value of $\alpha$, $w^*$ in (5) only depends on two variables, $\beta$ and $v$, hence can easily be tabulated to form a lookup table

  \[
  w^* = \arg \min_w |w|^\alpha + \frac{\beta}{2}(w-v)^2
  \]  

  \[
  \text{(5)}
  \]

- **X-sub problem**

  \[
  x = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(F^1)\ast \mathcal{F}(w^1) + \mathcal{F}(F^2)\ast \mathcal{F}(w^2) + (\lambda/\beta)\mathcal{F}(K)\ast \mathcal{F}(y)}{\mathcal{F}(F^1)\ast \mathcal{F}(F^1) + \mathcal{F}(F^2)\ast \mathcal{F}(F^2) + (\lambda/\beta)\mathcal{F}(K)\ast \mathcal{F}(K)}\right)
  \]  

  \[
  \text{(4)}
  \]

- **Computational requirement:**
  - Krishnan and Fergus’s method requires 3 FFTs for each iteration and a total of 18 FFTs for the entire deconvolution (for $T=1$ and 6 values of $\beta$) for each color channel.
ACCELERATION BY CLAMDFFT

- clAmdFft is an OpenCL library implementation of discrete Fast Fourier Transforms
- clAmdFft 1.2 features:
  - Provides a fast and accurate platform for calculating discrete FFTs
  - Single precision floating point formats
  - Works on CPU, GPU, or APU
  - 1D, 2D, and 3D transforms with batch support
  - Planar and interleaved data formats
  - Power-of-2 dimensions (other radices in progress)
ACCELERATION BY CLAMDFFT

- Client program provided
  - Initialization (create OpenCL context and set up devices)
    
    ```c
    clAmdFftSetup( setupData.get( ));
    ```

  - Data Initialization
    
    ```c
    clAmdFftCreateDefaultPlan(&plHandle, context, dim, clLengths)
    clAmdFftSetLayout(plHandle, inLayout, outLayout ); //set 2D, I/O buffers, Planar complex
    ```

  - Bake Plan
    
    ```c
    clAmdFftBakePlan( plHandle, 1, &queue, NULL, NULL )
    ```

  - Set Up Temp Buffer as needed
    
    ```c
    clAmdFftGetTmpBufSize(plHandle, &buffer size )
    clCreateBuffer ( context, CL_MEM_READ_WRITE , buffer size, 0, &medstatus);
    ```
ACCELERATION BY CLAMDFFT, CONT’D

– Execution

```
clAmdFftEnqueueTransform( plHandle, CLFFT_FORWARD, 1, &queue, 0, NULL, &outEvent,
                          &clMemBuffersIn[ 0 ], BuffersOut, clMedBuffer )

c1Finish(queue)
```

– Read Output Buffer

```
c1EnqueueReadBuffer(queue,clMemBuffersIn[0],CL_TRUE,0,buffSizeBytesIn,&output[0],0,NULL,NULL)
```

– Destroy and Clean-up

```
c1AmdFftDestroyPlan( &plHandle )

```
```
c1AmdFftTeardown( )
```

EXPERIMENTAL RESULTS

- Run times on an AMD Phenom™ II X4 Processor (2.6 GHz, 3.25GB RAM) with 32-bit Windows® 7 system and AMD Radeon™ HD 6950 graphics card:
  - 1920x1080 H.264 decode and 1024x1024 FFT for Wiener Filter
    - 26.5ms per frame
  - 1920x1080 H.264 decode and 1024x1024 FFT for Krishnan and Fergus’s method
    - 106ms per frame
  - 720x480 H.264 decode and 512x512 FFT for Wiener Filter
    - 8.4ms per frame
  - 720x480 H.264 decode and 512x512 FFT for Krishnan and Fergus’s method
    - 28ms per frame
CONCLUSION AND REMARKS

- We have presented the use of OpenCL interoperability, OpenDecode and clAmdFft for performing hardware decode and video quality enhancement by GPU.

- Runtime performance indicates that we can achieve real-time H.264 decoding and deblurring for HD video by using Wiener filter and for 512x512 video by using Krishnan and Fergus’s method for high quality.

- Optimization opportunities:
  - The hardware decoding latency can be hidden by interleaving OpenDecode with post-processing.
  - The efficiency of clAmdFft can be improved by utilizing transforms in batches, which can minimize the penalty of transfer overhead.

- This presentation is based on the OpenDecode API Tutorial and the OpenCL Fast Fourier Transforms document.
  - clAmdFft 1.2 binary packages are available for download at http://developer.amd.com/gpu/appmathlibs/Pages/default.aspx.
Disclaimer & Attribution
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREOF, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, AMD Phenom, AMD Radeon, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

OpenCL is a trademark of Apple Inc. used with permission by Khronos.
Windows is a registered trademark of Microsoft Corporation.

© 2011 Advanced Micro Devices, Inc. All rights reserved.