CUDA ON MOBILE
Yogesh Kini, GTC 2016
ABSTRACT

Typical pipeline
CUDA Interop APIs
Unified Memory on Tegra
TYPICAL USE CASES

Automobiles: Autonomous Cars

Mobile Devices: Consoles, Tablets

Embedded: Drones, Robots, Smart-Surveillance
TYPICAL PIPELINE

Camera → ISP/DSP → NVIDIA CUDA → Graphics
Camera → ISP/DSP → Graphics

Sensor → ISP/DSP → NVIDIA CUDA → Actuators
Sensor → ISP/DSP → Actuators

CAPTURE
PROCESS
DISPLAY

cuDNN
Caffe
CUDA

4/1/2016
CUDA OPENGL(ES) INTEROP
CUDA-OPENGL(ES)

- Provide access to OpenGL-ES resources in CUDA
- **Support for EGL**
- Supported on Android, L4T, Vibrante-Linux, QNX
- Implicit synchronization support
- Useful for graphics applications and games
EGL IMAGE INTEROP
EGL IMAGE

Source for EGL image

- GStreamer
- OpenGL ES
- OpenMAX
- Android - GraphicBuffer

Khronos EGL_image_base:

https://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_image_base.txt
EGL IMAGE

Begin resource usage in Other API
Other API code
End resource usage in Other API

Begin resource Usage in CUDA
CUDA code
End resource Usage in CUDA

synchronize

cuGraphicsEGLRegisterImage()
cuGraphicsResourceGetMappedPointer()
cuGraphicsResourceGetMappedArray()
EGL STREAMS INTEROP
EGL STREAMS

- Producer-Consumer architecture
- EGL streams spec: [https://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream.txt](https://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream.txt)
- Implicit Synchronization
- Cross Process support
- Supports YUV formats
EGL STREAMS

CUDA Producer

Frame

1

cuEGLStreamProducerPresentFrame(frame)

cuEGLStreamProducerReturnFrame(frame)

EGL Stream

Frame

2

cuEGLStreamConsumerAcquireFrame(frame)

Use in CUDA

Frame

3

cuEGLStreamConsumerReleaseFrame(frame)

CUDA Consumer
INTEROP SUMMARY

**EGL STREAMS**
- Producer-Consumer
- Implicit-Synchronization
- Cross-Process support
- YUV Planar Image support

**EGL IMAGE**
- Easy setup
- Works with several EGL client API
- YUV Planar Image support

**CUDA-OPENGL**
- EGL support
- OpenGL-ES support
- Portable across Tegra and discrete GPU
CUDA UNIFIED MEMORY ON TEGRA

- Helps take advantage of unified DRAM on Tegra
- Easier to program, Unified allocator: cudaMallocManaged
- Programming model enforced through memory access protection
- Memcpy not needed, migration managed by CUDA driver
- Saves memory consumption and power
- Attach API will help achieve optimal performance
# CUDA MEMORY TYPES

<table>
<thead>
<tr>
<th>Allocate</th>
<th>CPU use</th>
<th>Migrate</th>
<th>CUDA kernel</th>
<th>Migrate</th>
<th>CPU use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traditional</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>malloc()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cudaMalloc()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Zero Copy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cudaMallocHost()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unified Memory</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cudaMallocManaged()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CPU use**
- cudaMalloc()
- cudaMemcpyHtoD()
- cudaMemcpyDtoH()
- cudaMemAttach[Optional]

**Zero Copy**
- cudaMemcpyHtoD()
- cudaMemcpyDtoH()
- cudaMemAttach[Optional]

**Unified Memory**
- cudaMemcpyHtoD()
- cudaMemcpyDtoH()
- cudaMemAttach[Optional]
CUDA MEMORY TYPES

<table>
<thead>
<tr>
<th>Traditional</th>
<th>Zero Copy</th>
<th>Managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Easy portability from existing desktop programs</td>
<td>• Cache is bypassed by both GPU and CPU while accessing these allocations</td>
<td>• Memory access by CPU and GPU is through cache.</td>
</tr>
<tr>
<td>• Faster for some small allocations</td>
<td>• Suitable when memory access is not affected by caching</td>
<td>• Faster for larger allocations</td>
</tr>
<tr>
<td>• Suitable for GPU intermediate buffers, tables, etc</td>
<td></td>
<td>• Suitable when memory used on both host and GPU</td>
</tr>
</tbody>
</table>

Time taken (ms) by the Matrix Multiply CUDA kernel with different allocation types:

<table>
<thead>
<tr>
<th></th>
<th>TRADITIONAL</th>
<th>ZERO COPY</th>
<th>MANAGED MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>16KB</td>
<td>0.617</td>
<td>0.544</td>
<td>0.644</td>
</tr>
<tr>
<td>1MB</td>
<td>9.723</td>
<td>11.119</td>
<td>7.093</td>
</tr>
<tr>
<td>4MB</td>
<td>59.37618</td>
<td>62.232</td>
<td>46.42551</td>
</tr>
<tr>
<td>16MB</td>
<td>377.9244</td>
<td>403.2382</td>
<td>344.926</td>
</tr>
</tbody>
</table>
THANK YOU

JOIN THE CONVERSATION
#GTC16  

JOIN THE NVIDIA DEVELOPER PROGRAM AT developer.nvidia.com/join