Metal for Pro Apps

Eugene Zhidkov, GPU Software
Dileep Madhava, GPU Software
Brian Ross, GPU Software
What Are Pro Apps?
What Are Pro Apps?

Used by professional content creators for the media-making business

• Animated, live TV, and film
• Photography
• 3D animation and games
• Print media
• Audio production
What Are Pro Apps?

Used by professional content creators for the media-making business

• Animated, live TV, and film
• Photography
• 3D animation and games
• Print media
• Audio production

Apple and third party apps
Pro App Platforms
Pro Apps Requirements

Operate on large assets and data

Heavy compute demand

Real-time for creativity vs. offline for full fidelity
Optimizing for 8K video editing
Support for high dynamic range
Leveraging all platform resources
Efficient data transfers
Optimizing for 8K video editing
Support for high dynamic range
Leveraging all platform resources
Efficient data transfers
Before
Current 8K Proxy Workflow
Current 8K Proxy Workflow

Raw camera footage
Current 8K Proxy Workflow

Raw camera footage

Import

Transcode
Current 8K Proxy Workflow

Raw camera footage

Import

Transcode

Video editing app

Downsample subsample

Edit and color grade

8K proxy
Current 8K Proxy Workflow

1. Import Raw camera footage
2. Transcode
3. Downsample
4. Apply edits to 8K
   - Content off-line
5. Edit and color grade
6. Video editing app
Current 8K Proxy Workflow

Raw camera footage

Import

Transcode

Export

8K

4K proxy

Apply edits to 8K
Content off-line

Video editing app

Downsample subsample

Edit and color grade
Goal — Realtime Edit of 8K Content

Raw camera footage → Video editing app

Import → Export

8K
Optimizing for 8K Video Editing

Video editing pipeline
Managing large asset sizes
Maintaining a predictable frame rate
Optimizing for 8K Video Editing

Video editing pipeline
Managing large asset sizes
Maintaining a predictable frame rate
Video Editing Pipeline

Footage content storage

- Import
- Decode
- Pixel Processing
- Display
- Encode
- Export

Edit and playback in video editing app

Final content
Video Editing Pipeline

Decode → Pixel Processing → Display
~

Pixel Processing → Encode
~

Display → Pixel Processing → Decode
Video Editing Pipeline

Decode → Pixel Processing

Display

Encode
Video Editing Pipeline
Decode with VideoToolbox

Multiple decoders available via VTDecompressionSession

Afterburner card support
Video Editing Pipeline
Decode with VideoToolbox

Multiple decoders available via VTDecompressionSession

Afterburner card support
CFDictionarySetValue(decoderSpec,  
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,  
    kCFBooleanTrue);

// This is how you set-up VTDecompressionSession  
VTDecompressionSessionCreate(...,  
     videoFormatDescription, // Input format  
     decoderSpec,  
     destinationImageBufferAttributes,  
     ^(...) {...}, // didDecompress call for a single decompressed frame  
     &session);

...  

VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;  
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);

...  

VTDecompressionSessionInvalidate(session);
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFSBooleanTrue);

// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
    &session);

VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags,...);

VTDecompressionSessionInvalidate(session);
CFDictionarySetValue(decoderSpec,  
   kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,  
   kCFBooleanTrue);

// This is how you set-up VTDecompressionSession

VTDecompressionSessionCreate(...,  
   videoFormatDescription, // Input format  
   decoderSpec,  
   destinationImageBufferAttributes,  
   ^(...) {...}, // didDecompress call for a single decompressed frame  
   &session);

...

VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
...

VTDecompressionSessionInvalidate(session);
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFBooleanTrue);

// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
    &session);

...  

VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
...

VTDecompressionSessionInvalidate(session);
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFBooleanTrue);

// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
&session);
...
VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
...
VTDecompressionSessionInvalidate(session);
Video Editing Pipeline

Metal with Core Video

- GPU 1
- GPU 2
- CPU
- Afterburner

Decoders
Video Editing Pipeline
Metal with Core Video

Leverage IOSurface as a backing store

Use CVMetalTextureCache to directly access decoded frames
CVMetalTextureCacheRef sessionMetalCache;

CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(...,
    sessionMetalCache,
    pixelBuffer,
    metalFormat,
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0),
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0),
    0, &textureOut);

id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);
...

CFRelease(textureOut); // Release when Metal processing is done
...

CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
CVMetalTextureCacheRef sessionMetalCache;
CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(...,
    sessionMetalCache,
    pixelBuffer,
    metalFormat,
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0),
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0),
    0, &textureOut);

id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);
...
CFRelease(textureOut); // Release when Metal processing is done
...
CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
CVMetalTextureCacheRef sessionMetalCache;
CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(..., 
    sessionMetalCache, 
    pixelBuffer, 
    metalFormat, 
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0), 
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0), 
    0, &textureOut);

id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);

... 

CFRelease(textureOut); // Release when Metal processing is done 
...

CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
CVMetalTextureCacheRef sessionMetalCache;
CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(...,
    sessionMetalCache,
    pixelBuffer,
    metalFormat,
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0),
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0),
    0, &textureOut);

id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);
...

CFRelease(textureOut); // Release when Metal processing is done
...

CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
Video Editing Pipeline

- Decode
- Pixel Processing
- Encode
- Display
Video Editing Pipeline
Pixel processing with Metal

Implement custom shaders for filters and effects

MPS provides optimized implementations for many common filters
func myBlurTextureInPlace(inTexture: MTLTexture,.blurRadius: Float, queue: MTLCommandQueue)
{
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue) {

    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue) {
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
    inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue) {

    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
    inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue) {
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
inPlaceTexture.initialize(inTexture)
    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()}

Video Editing Pipeline

- Decode → Pixel Processing → Display
- Pixel Processing → Encode
Video Editing Pipeline

1. Decode
2. Pixel Processing
3. Encode
4. Display
Video Editing Pipeline
Encode with VideoToolbox

Similar to decode but using VTCompressionSession APIs
CVPixelBufferPool to get Metal textures
Video Editing Pipeline
Encode with VideoToolbox

Similar to decode but using VTCompressionSession APIs
CVPixelBufferPool to get Metal textures
VTCopyVideoEncoderList(...); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
...

CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(..., pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
...

CVBufferRelease(buffer); // Crucial to keep it recycled
`VTCopyVideoEncoderList(...);` // This is how you get the list of all available encoders

```cpp
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
...
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(..., pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
...
CVBufferRelease(buffer); // Crucial to keep it recycled
```
VTCopyVideoEncoderList(...); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
...
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(...) , pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
...
CVBufferRelease(buffer); // Crucial to keep it recycled
VTCopyVideoEncoderList(...); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);

... 
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(..., pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);

... 
CVBufferRelease(buffer); // Crucial to keep it recycled
VTCopyVideoEncoderList(...); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);

... 

CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(..., pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);

...

CVBufferRelease(buffer); // Crucial to keep it recycled
VTCopyVideoEncoderList(...); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
...
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(..., pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
...
CVBufferRelease(buffer); // Crucial to keep it recycled
Video Editing Pipeline

- Decode
- Pixel Processing
- Encode
- Display
Optimizing for 8K Video Editing

Video editing pipeline
Managing large asset sizes
Maintaining a predictable frame rate
16x the Memory of HD Content
16x the Memory of HD Content

UHD
3840x2160

HD
1920x1080
16x the Memory of HD Content

- 8K: 7680x4320
- UHD: 3840x2160
- HD: 1920x1080
Virtual Memory Residency
Virtual Memory Residency

[Image of a memory activity graph with various metrics and data points.]
Virtual Memory Residency
Virtual Memory Residency
Virtual Memory Residency
Managing Large Asset Sizes

Virtual memory residency

High VM page fault cost with large 8K allocations

Can impact performance at the start of video playback
Managing Large Asset Sizes
Virtual memory residency

High VM page fault cost with large 8K allocations
Can impact performance at the start of video playback
Pre-warm your buffers before playback starts
Managing Large Asset Sizes
Allocation best practices

Allocate early to minimize allocation cost mid-workflow

Reuse memory by using buffer pools

Use Metal heaps for transient allocations
Managing Transient Allocations

MTLHeap advantages

Allocating from MTLHeap is cheap
Heap is made resident as a whole
MTLHeap uses memory more efficiently
Resources may be aliased and memory may be reused
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
extecuteBlur(input, output1, blurUniforms)

// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
extecuteColorgrade(input, output2, colorgradeUniforms)

... // Tell the heap we don’t need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()

... // Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
extecuteCombinedOutput(output1, output2, intermediateBuffer, output3)
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)

// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)

...  

// Tell the heap we don’t need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()

...  

// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)

// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...

// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...

// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
execluteCombinedOutput(output1, output2, intermediateBuffer, output3)
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)

// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)

... 

// Tell the heap we don’t need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()

... 

// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)

// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...

// Tell the heap we don’t need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...

// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)

// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)

...  

// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()

...  

// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
Optimizing for 8K Video Editing

Video editing pipeline
Managing large asset sizes
Maintaining a predictable frame rate
Uneven Frame Pacing

60Hz–16.7 ms

VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL

CPU

GPU

Display

1 2 3 | 4 5 6 7

1 2 3 | 4 5 6

1 1 2 | 3 3 4 5 5 6
Uneven Frame Pacing

<table>
<thead>
<tr>
<th></th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
<th>VBL</th>
</tr>
</thead>
</table>

- 60Hz–16.7 ms

**CPU**

**GPU**

**Display**
Uneven Frame Pacing

60Hz–16.7 ms

VBL  VBL  VBL  VBL  VBL  VBL  VBL  VBL  VBL  VBL  VBL  VBL

CPU

1  2  3  blocked on drawable...

GPU

1  2

Display

1  1
Uneven Frame Pacing

60Hz–16.7 ms

- **CPU**
  - 1
  - 2
  - 3 (blocked on drawable...)
  - 4

- **GPU**
  - 1
  - 2
  - 3

- **Display**
  - 1
  - 1
  - 2
Uneven Frame Pacing

60Hz–16.7 ms

VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL

CPU

1 2 3 blocked on drawable... 4 ... 5 ...

GPU

1 2 3 4 5

Display

1 1 2 3 3
Uneven Frame Pacing

60Hz–16.7 ms

CPU

GPU

Display
Predictable Frame Pacing with CVDisplayLink

60Hz–16.7 ms

VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL VBL

CPU

GPU

Display

CVDL Interrupt
// Set-up CVDisplayLink
CVDisplayLinkRef displayLink;
CVDisplayLinkCreateWithCGDisplay(display, &displayLink);

// Callback to control the cadence
__block int lastFrameIdx = -1;
CVDisplayLinkSetOutputHandler(displayLink, ^(…, inNow, inOutput, …)
{
    int outFrameIdx = floor(inOutputSec * frameDesiredFrequency);
    if (outFrameIdx > lastFrameIdx && !presentQueue.empty())
    {
        lastFrameIdx = outFrameIdx;
        id<MTLDrawable> toShow = presentQueue.pop();
        present(toShow);
    }
    return kCVReturnSuccess;
});
// Set-up CVDisplayLink
CVDisplayLinkRef displayLink;
CVDisplayLinkCreateWithCGDisplay(display, &displayLink);

// Callback to control the cadence
__block int lastFrameIdx = -1;
CVDisplayLinkSetOutputHandler(displayLink, ^(…, inNow, inOutput, …)
{
    int outFrameIdx = floor(inOutputSec * frameDesiredFrequency);
    if (outFrameIdx > lastFrameIdx && !presentQueue.empty())
    {
        lastFrameIdx = outFrameIdx;
        id<MTLDrawable> toShow = presentQueue.pop();
        present(toShow);
    }
    return kCVReturnSuccess;
});
// Set-up CVDisplayLink
CVDisplayLinkRef displayLink;
CVDisplayLinkCreateWithCGDisplay(display, &displayLink);

// Callback to control the cadence
__block int lastFrameIdx = -1;
CVDisplayLinkSetOutputHandler(displayLink, ^(…, inNow, inOutput, …)
{
    int outFrameIdx = floor(inOutputSec * frameDesiredFrequency);
    if (outFrameIdx > lastFrameIdx && !presentQueue.empty())
    {
        lastFrameIdx = outFrameIdx;
        id<MTLDrawable> toShow = presentQueue.pop();
        present(toShow);
    }
    return kCVReturnSuccess;
});
Content Rate Not Equal to Display Rate

Content

Display

24Hz–41.7 ms

60Hz–16.7 ms
Pro Display XDR
Matching refresh and content rate

Supports multiple common video playback rates (48Hz, 50Hz)

24Hz–41.7 ms
After
“With the recent DaVinci Resolve and macOS updates, we are able to get significant performance improvements for video playback and UI interactivity. This enables workflows like 8K editing, color correction, and finishing on Mac desktops and laptops.”

Rohit Gupta, Director of DaVinci Software Engineering, Blackmagic Design
Optimizing for 8K video editing
Support for high dynamic range
Leveraging all platform resources
Efficient data transfers
Support for High Dynamic Range

Dileep Madhava, GPUSW
Common traits of HDR images
Apple’s approach to HDR
HDR rendering with Metal
Best practices
Common Traits of HDR Images
Common Traits of HDR Images

Better contrast levels
Common Traits of HDR Images

Better contrast levels

More colors
Common Traits of HDR Images

Better contrast levels
More colors
Increased brightness
Common Traits of HDR Images

Better contrast levels
More colors
Increased brightness
Display that preserves artistic intent of image
Extended Dynamic Range (EDR)

Apple’s approach to HDR

Brightness head room utilized for highlights and shadows
Extended Dynamic Range (EDR)

Apple’s approach to HDR

Brightness head room utilized for highlights and shadows
Extended Dynamic Range (EDR)

Apple’s approach to HDR

Brightness head room utilized for highlights and shadows

**Dim**
- Brightness: 200 nits
- Headroom: 800 nits

**Well-Lit**
- Brightness: 500 nits
- Headroom: 500 nits
Extended Dynamic Range (EDR)

Apple’s approach to HDR

Brightness head room utilized for highlights and shadows
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness.
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness.
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness.
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

![Graph showing Extended Dynamic Range (EDR)]
Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

Well-Lit

- HDR Range
- SDR Range
- Maximum EDR

Pixel Values: 0.0, 1.0, 2.0, 3.0, 4.0, 5.0

Nits: 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
HDR Rendering Options
HDR Rendering Options

Video playback apps

AVFoundation API

Metal (CAMetalLayer and EDR API)

Display
HDR Rendering Options

- Video playback apps
  - AVFoundation API
- Content creation apps
  - Metal (CAMetalLayer and EDR API)

Display
// Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
// Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
// Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
// Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
// Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
// Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
// Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayerPixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
/ Check for EDR support on display
NSScreen * screen = view.window.screen;
CGFloat edrSupport = screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
...
HDR Pixel Processing in Shaders

Sample
YUV
PQ
HDR Pixel Processing in Shaders

Sample YUV PQ → Convert to RGB
HDR Pixel Processing in Shaders

- Sample YUV PQ
- Convert to RGB
- Linearize PQ to RGB [0,1]
HDR Pixel Processing in Shaders

Sample YUV PQ → Convert to RGB → Linearize PQ to RGB [0,1] → Scale by maxEDR
HDR Pixel Processing in Shaders

Sample YUV PQ → Convert to RGB → Linearize PQ to RGB [0,1] → Scale by maxEDR → Editing/Grading
HDR Pixel Processing in Shaders

Sample YUV PQ → Convert to RGB → Linearize PQ to RGB [0,1] → Scale by maxEDR → Editing/Grading → Tone Map
Sample YUV PQ → Convert to RGB → Linearize PQ to RGB [0,1] → Scale by maxEDR → Editing/Grading → Tone Map

```c
CAEDRMetadata *edrMetaData = CAEDRMetadata(minLuminance, maxLuminance, opticalOutputScale);
metalLayer.edrMetadata = edrMetaData;
```
Best Practices
Best Practices

Update content when display brightness changes
Update content when display brightness changes

MTLPixelFormatRGBA16Float recommended for HDR rendering
Best Practices

Update content when display brightness changes

MTLPixelFormatRGBA16Float recommended for HDR rendering

Select color space and transfer function that matches content
Best Practices

Update content when display brightness changes

MTLPixelFormatRGBA16Float recommended for HDR rendering

Select color space and transfer function that matches content

Bypass tone mapping if contents are already tone mapped
Optimizing for 8K video editing
Support for high dynamic range
Leveraging all platform resources
Efficient data transfers
Leveraging All Platform Resources
Scale with CPU cores
Scale with GPU channels
Scale with multiple GPUs
## Scaling with CPU Cores

<table>
<thead>
<tr>
<th>Core Range</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 1–7</td>
<td>UI, Metal, Decode, ...</td>
</tr>
<tr>
<td>Core 2–14</td>
<td></td>
</tr>
<tr>
<td>Core 15–21</td>
<td></td>
</tr>
<tr>
<td>Core 22–28</td>
<td></td>
</tr>
</tbody>
</table>
Scaling with CPU Cores

CPU

- Core 1–7
  - UI Metal Decode...
- Core 2–14
- Core 15–21
- Core 22–28
Scaling with CPU Cores

CPU

Core 1–7
- UI Player
- UI Player
- UI Player
- UI Player
- UI Timeline
- UI Timecode
- UI Peakmeter

Core 2–14
- Metal Compute
- Metal Compute
- Metal Renderer
- Metal Renderer
- Metal Blit
- Metal Blit
- Metal Blit

Core 15–21
- Video Decode
- Video Decode
- Video Decode
- Video Decode
- Video Decode
- Video Decode
- Video Decode

Core 22–28
- AV Asset Reader
- AV Asset Reader
- AV Asset Reader
- AV Asset Reader
- AV Asset Reader
- AV Asset Writer
- AV Asset Writer
Single-Thread Rendering
Single-Thread Rendering

Increased latency

Thread 0
Decode | UI | Resource Updates | Filter | Effects + Blend
Multi-Threaded Rendering

MTLCommandBuffer

Thread 0: Decode, UI, Resource Updates, Filter, Effects + Blend

Thread 1:

Thread 2:

Thread 3:
Multi-Threaded Rendering

MTLCommandBuffer

Thread 0
- Decode

Thread 1
- UI
- Resource Updates

Thread 2
- Filter

Thread 3
- Effects + Blend
Multi-Threaded Rendering

MTLCommandBuffer

Thread 0: Decode
Thread 1: UI, Resource Updates
Thread 2: Filter
Thread 3: Effects + Blend
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, ... )
    commandBuffer2.commit()
}
queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, ... )
    commandBuffer1.commit()
}
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, ... )
    commandBuffer2.commit()
}

queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, ... )
    commandBuffer1.commit()
}
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, ... )
    commandBuffer2.commit()
}
queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, ... )
    commandBuffer1.commit()
}
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, ... )
    commandBuffer2.commit()
}
queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, ... )
    commandBuffer1.commit()
}
Multi-Threaded Rendering

MTLCommandBuffer

Thread 0: Decode
Thread 1: UI, Resource Updates
Thread 2: Filter
Thread 3: Effects + Blend
Multi-Threaded Rendering
Without MTLParallelRenderCommandEncoder

Thread 0
- Decode

Thread 1
- UI
- Resource Updates

Thread 2
- Filter

Thread 3
- Effects + Blend
Multi-Threaded Rendering
With MTLParallelRenderCommandEncoder

Thread 0: Decode
Thread 1: UI, Resource Updates
Thread 2: Filter
Thread 3: Effects + Blend
Thread 4: Effects + Blend
// Create parallel encoder and subordinate render command encoder objects
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!

// Encode different portions of render effects (in any order) on separate threads
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}

// Notify when encoding complete and end the parallel encoder
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
/ Create parallel encoder and subordinate render command encoder objects
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!

/ Encode different portions of render effects (in any order) on separate threads
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}

/ Notify when encoding complete and end the parallel encoder
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
Create parallel encoder and subordinate render command encoder objects

```swift
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!
```

Encode different portions of render effects (in any order) on separate threads

```swift
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}
```

Notify when encoding complete and end the parallel encoder

```swift
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
```
// Create parallel encoder and subordinate render command encoder objects
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!

// Encode different portions of render effects (in any order) on separate threads
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}

// Notify when encoding complete and end the parallel encoder
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
Scaling with GPU Channels

- Video Channel
- Blit Channel
- Compute Channel
- Render Channel
Standard Video Workload

- Decode
- Resource Update
- Filter
- FX
- Blend
Standard Video Workload
Standard Video Workload

Frame 1
Standard Video Workload

Frame 1
Standard Video Workload

Frame 1

Frame 2

Frame 3

... Frame N

Video
Blit
Compute
Render
Parallel GPU Channels with Gaps

<table>
<thead>
<tr>
<th>CPU</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
<th>Thread 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>Video</td>
<td>Blit</td>
<td>Compute</td>
<td>Render</td>
</tr>
</tbody>
</table>

60Hz–16.7 ms
Parallel GPU Channels with Gaps

60Hz–16.7 ms

Thread 1
Thread 2
Thread 3
Thread 4

CPU

GPU

Video
Blit
Compute
Render
Parallel GPU Channels with Gaps

60Hz–16.7 ms

CPU

Thread 1
Thread 2
Thread 3
Thread 4

GPU

Video
Blit
Compute
Render

Decode 1
Upload 1
Filter 1
Effects/Blend 1
Parallel GPU Channels with Gaps

60Hz–16.7 ms

Thread 1
Thread 2
Thread 3
Thread 4

CPU

Video
Blit
Compute
Render

GPU

Decode 1
Decode 2
Decode 3
Decode 4

Upload 1
Upload 2
Upload 3

Filter 1
Filter 2

Effects/Blend 1
Effects/Blend 2
Effects/Blend 3

Thread 1
Thread 2
Thread 3
Thread 4

VBL
VBL
VBL
VBL

Thread 1
Thread 2
Thread 3
Thread 4

Thread 1
Thread 2
Thread 3
Thread 4

Thread 1
Thread 2
Thread 3
Thread 4

Thread 1
Thread 2
Thread 3
Thread 4

Thread 1
Thread 2
Thread 3
Thread 4
Parallel GPU Channels with Gaps

60Hz–16.7 ms

CPU
- Thread 1
- Thread 2
- Thread 3
- Thread 4

GPU
- Video
- Blit
- Compute
- Render

Thread 1
- Decode 1
- Upload 1
- Filter 1
- Effects/Blend 1

Thread 2
- Decode 2
- Upload 2

Thread 3
- Decode 3
- Upload 3

Thread 4
- Decode 4

VBL VBL VBL VBL
Parallel GPU Channels with Gaps

60Hz–16.7 ms

CPU

- Thread 1
- Thread 2
- Thread 3
- Thread 4

GPU

- Video
- Blit
- Compute
- Render

- Decode 1
- Decode 2
- Decode 3
- Decode 4

- Upload 1
- Upload 2
- Upload 3

- Filter 1
- Effects/Blend 1

- Thread 1:
  - Decode 1
  - Upload 1

- Thread 2:
  - Decode 2
  - Filter 1

- Thread 3:
  - Decode 3
  - Effects/Blend 1

- Thread 4:
  - Decode 4
  - Upload 3
Parallel GPU Channels with Gaps

Thread 1
Thread 2
Thread 3
Thread 4

CPU

GPU

Video
Blit
Compute
Render

Decode 1
Decode 2
Decode 3
Decode 4

Upload 1
Upload 2
Upload 3

Filter 1
Filter 2
Filter 3

Effects/Blend 1
Effects/Blend 2

waitUntilCompleted returns

60Hz–16.7 ms
Parallel GPU Channels with Gaps

```javascript
commandBuffer.waitUntilCompleted()

commandBuffer.addCompletionHandler {
  // Code to execute after GPU completion
  ...
}
```
Parallel GPU Channels with Gaps

60Hz–16.7 ms

Thread 1
Thread 2
Thread 3
Thread 4

CPU

GPU

Video
Blit
Compute
Render

Decode 1
Decode 2
Decode 3
Decode 4

Upload 1
Upload 2
Upload 3

Filter 1
Filter 2

Effects/Blend 1
Parallel GPU Channels with Gaps

60Hz–16.7 ms

Thread 1 1 2 3 4 5 6 7
Thread 2 1 2 3 4 5 6 7
Thread 3 1 2 3 4 5 6 7
Thread 4 1 2 3 4 5 6 7

CPU

GPU

Video Decode 1 Decode 2 Decode 3 Decode 4
Blit Upload 1 Upload 2 Upload 3
Compute Filter 1 Filter 2 Filter 3
Render Effects/Blend 1 Effects/Blend 2 Effects/Blend 3
Parallel GPU Channels with Gaps

60Hz–16.7 ms

Thread 1
1
Thread 2
2
Thread 3
3
Thread 4
4

CPU

Video
Blit
Compute
Render

Decode 1
Upload 1
Filter 1
Effects/Blend 1
Decode 2
Upload 2
Filter 2
Effects/Blend 2
Decode 3
Upload 3
Filter 3
Effects/Blend 3
Decode 4
Upload 4
Filter 4
Effects/Blend 4
Parallel GPU Channels with Less Gaps

60Hz–16.7 ms

CPU

GPU

Video

Decode 10

Decode 11

Decode 12

Decode 13

Blit

Upload 1

Upload 2

Upload 3

Upload 4

Upload 5

Compute

Filter 1

Filter 2

Filter 3

Filter 4

Render

Effects/Blend 1

Effects/Blend 2

Effects/Blend 3

Effects/Blend 4
Parallel GPU Channels with Less Gaps

60Hz–16.7 ms

CPU

Thread 1
10

Thread 2
1

Thread 3
1

Thread 4
1

GPU

Video
Decode 10
Decode 11
Decode 12
Decode 13

Blit
Upload 1
Upload 2
Upload 3
Upload 4
Upload 5

Compute
Filter 1
Filter 2
Filter 3
Filter 4

Render
Effects/Blend 1
Effects/Blend 2
Effects/Blend 3
Effects/Blend 4

Thread 1: 10, 11, 12, 13
Thread 2: 1, 2, 3, 4
Thread 3: 1, 2, 3, 4
Thread 4: 1, 2, 3, 4

Video: Decode 10, Decode 11, Decode 12, Decode 13
Blit: Upload 1, Upload 2, Upload 3, Upload 4, Upload 5
Compute: Filter 1, Filter 2, Filter 3, Filter 4
Render: Effects/Blend 1, Effects/Blend 2, Effects/Blend 3
**Parallel GPU Channels with Minimal Gaps**

![Diagram of parallel GPU channels with minimal gaps](image)

- **CPU Threads:**
  - Thread 1: Decode 10
  - Thread 2: Decode 11
  - Thread 3: Decode 12
  - Thread 4: Decode 13

- **GPU Operations:**
  - **Video:** Decode 10, Decode 11, Decode 12, Decode 13
  - **Blit:** Upload 5, Upload 6, Upload 7, Upload 8, Upload 9
  - **Compute:** Filter 1, Filter 2, Filter 3, Filter 4, Filter 5, Filter 6, Filter 7
  - **Render:** Effects/Blend 1, Effects/Blend 2, Effects/Blend 3, Effects/Blend 4, Effects/Blend 5

- **Timeline:** 60Hz (~16.7 ms)
Scaling with Multiple GPUs

Multi GPU support in Metal

Load balancing strategies

Synchronization
Multi GPU Support in Metal

Detect multi GPU capabilities

Easy management with Metal device objects

Transfer data between GPUs

Powerful synchronization support
Metal GPU Detection
MTLDevice properties

// Query for GPU properties
let gpuLocation = device.location
let gpuLocationNumber = device.locationNumber
let gpuMaxTransferRate = device.maxTransferRate
let gpuIsLowPower = device.isLowPower
let gpuIsHeadless = device.isHeadless
let gpuIsRemovable = device.isRemovable

<table>
<thead>
<tr>
<th></th>
<th>Location</th>
<th>LocationNumber</th>
<th>isRemovable</th>
<th>isLowPower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated GPU</td>
<td>MTLDeviceLocationBuiltin</td>
<td>0</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>Discrete GPU</td>
<td>MTLDeviceLocationBuiltin</td>
<td>1</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>External GPU</td>
<td>MTLDeviceLocationExternal</td>
<td>Thunderbolt Port (1..N)</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>Mac Pro GPU</td>
<td>MTLDeviceLocationSlot</td>
<td>PCI Slot (0..N)</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Load Balancing

Many load balancing schemes
Consider your software architecture
Goal is high scaling efficiency
Load Balancing Strategies

Alternating frames

Odd frames

Frame 1, 3, 5, ...

GPU 1

Even frames

Frame 2, 4, 6, ...

GPU 2
Load Balancing Strategies
Interleaved tiling

Tile Group 1 → GPU 1
Tile Group 2 → GPU 2
Tile Group 3 → GPU 3
Tile Group 4 → GPU 4
Load Balancing Strategies
Interleaved tiling

Tile Group 1
Tile Group 2
Tile Group 3
Tile Group 4

GPU 1
GPU 2
GPU 3
GPU 4
Load Balancing Strategies

Interleaved tiling

Tile Group 1
Tile Group 2
Tile Group 3
Tile Group 4

GPU 1
GPU 2
GPU 3
GPU 4
Load Balancing Strategies

Interleaved tiling

Tile Group 1 → GPU 1
Tile Group 2 → GPU 2
Tile Group 3 → GPU 3
Tile Group 4 → GPU 4
Load Balancing Strategies
Interleaved tiling with queue
Load Balancing Strategies
Interleaved tiling with queue
Load Balancing Strategies
Interleaved tiling with queue
Load Balancing Strategies
Interleaved tiling with queue
Multi GPU Synchronization
MTLSharedEvent

Synchronize between
• Multiple GPUs
• CPU and GPU
• Processes
Multi GPU Synchronization
MTLSharedEvent
Multi GPU Synchronization

MTLSharedEvent
Multi GPU Synchronization

MTLSharedEvent

GPU 1

Frame 1  Frame 2  Motion Analysis 1  Frame 3  Frame 4  Motion Analysis 2

GPU 2
Multi GPU Synchronization

MTLSharedEvent

GPU 1

Frame 1 | Frame 2 | Motion Analysis 1 | Frame 3 | Frame 4 | Motion Analysis 2

Write: Frames 1,2 | Read: Frames 1,2

GPU 2
Multi GPU Synchronization

MTLSharedEvent

GPU 1

Frame 1 | Frame 2 | Motion Analysis 1 | Frame 3 | Frame 4 | Motion Analysis 2

Write: Frames 1,2 | Read: Frames 1,2 | Write: Frames 3,4 | Read: Frames 3,4

GPU 2


Multi GPU Synchronization
MTLSharedEvent

GPU 1
- Frame 1
- Frame 2
- Motion Analysis 1
- Frame 3
- Frame 4
- Motion Analysis 2

GPU 2
Multi GPU Synchronization
MTLSharedEvent

GPU 1
- Frame 1
- Frame 2
- Motion Analysis 1
- Frame 3
- Frame 4
- Motion Analysis 2

Signal (1)

GPU 2
Multi GPU Synchronization

MTLSharedEvent

GPU 1
- Frame 1
- Frame 2
- Motion Analysis 1
- Frame 3
- Frame 4
- Motion Analysis 2

- Signal (1)
- Wait (1)
- Signal (2)
- Wait (2)

GPU 2
Multi GPU Synchronization

MTLSharedEvent

Device A
CommandQueue A
CommandBuffer A

GPU 1
Frame 1
Frame 2
Frame 3
Frame 4
Signal = 1
Signal = 2
Signal = 3
Wait = 1
Wait = 2
Wait = 3

Device B
CommandQueue B
CommandBuffer B

GPU 2
Motion Analysis 1
Motion Analysis 2
Multi GPU Synchronization
MTLSharedEvent

Device A
CommandQueue A
CommandBuffer A

GPU 1
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Frame 7

Signal = 1
Signal = 2
Signal = 3

Wait = 1
Wait = 2
Wait = 3

Device B
CommandQueue B
CommandBuffer B

GPU 2
Motion Analysis 1
Motion Analysis 2
Motion Analysis 3
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
Metal System Trace
<table>
<thead>
<tr>
<th>GPU Device / Channel Name /...</th>
<th>Total Active</th>
<th>Total Idle</th>
<th>Min Duration</th>
<th>Avg Duration</th>
<th>Max Duration</th>
<th>Std Dev Dur...</th>
</tr>
</thead>
<tbody>
<tr>
<td>* All *</td>
<td>8.29 s</td>
<td>37.20 s</td>
<td>1 ns</td>
<td>3.27 ms</td>
<td>78.07 ms</td>
<td>11.49 ms</td>
</tr>
<tr>
<td>AMD Radeon Pro 560</td>
<td>3.54 s</td>
<td>7.03 s</td>
<td>1.76 μs</td>
<td>5.42 ms</td>
<td>78.07 ms</td>
<td>19.47 ms</td>
</tr>
<tr>
<td>sDMA1</td>
<td>5.64 ms</td>
<td>3.45 s</td>
<td>1.76 μs</td>
<td>17.45 μs</td>
<td>1.52 ms</td>
<td>144.72 μs</td>
</tr>
<tr>
<td>sDMA0</td>
<td>21.92 ms</td>
<td>3.50 s</td>
<td>15.04 μs</td>
<td>111.26 μs</td>
<td>346.08 μs</td>
<td>102.53 μs</td>
</tr>
<tr>
<td>Compute1</td>
<td>3.45 s</td>
<td>78.42 s</td>
<td>461.92 μs</td>
<td>38.29 ms</td>
<td>78.07 ms</td>
<td>38.83 ms</td>
</tr>
<tr>
<td>Blit</td>
<td>65.90 ms</td>
<td>1.68 ms</td>
<td>1.50 ms</td>
<td>1.53 ms</td>
<td>1.86 ms</td>
<td>60.84 μs</td>
</tr>
<tr>
<td>AMD Radeon Pro 580</td>
<td>1.76 s</td>
<td>8.72 s</td>
<td>1.28 μs</td>
<td>2.71 ms</td>
<td>30.21 ms</td>
<td>7.71 ms</td>
</tr>
<tr>
<td>AMD Radeon RX 570</td>
<td>1.83 s</td>
<td>8.65 s</td>
<td>1.28 μs</td>
<td>2.78 ms</td>
<td>31.87 ms</td>
<td>8.06 ms</td>
</tr>
<tr>
<td>Intel(R) HD Graphics 630</td>
<td>1.16 s</td>
<td>12.78 s</td>
<td>1 ns</td>
<td>2.01 ms</td>
<td>6.15 ms</td>
<td>1.96 ms</td>
</tr>
<tr>
<td>Channel Name</td>
<td>Creation Time</td>
<td>Duration</td>
<td>Frame</td>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute1</td>
<td>00:00.162.640</td>
<td>77.82 ms</td>
<td>Frame 1</td>
<td>Command Buffer 1:Initialize...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA1</td>
<td>00:00.162.699</td>
<td>1.92 μs</td>
<td>n/a</td>
<td>hello_world.app (1310)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA1</td>
<td>00:00.162.721</td>
<td>1.52 μs</td>
<td>n/a</td>
<td>Unknown 0x1d01ff04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA1</td>
<td>00:00.164.242</td>
<td>4.32 μs</td>
<td>n/a</td>
<td>Unknown 0x1d01ff07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA1</td>
<td>00:00.166.396</td>
<td>4.16 μs</td>
<td>n/a</td>
<td>Unknown 0x1d01ff28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA1</td>
<td>00:00.166.454</td>
<td>1.76 μs</td>
<td>n/a</td>
<td>Unknown 0x0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA0</td>
<td>00:00.166.478</td>
<td>204.96 μs</td>
<td>n/a</td>
<td>Page On 1,048,576 bytes (...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA0</td>
<td>00:00.166.684</td>
<td>19.20 μs</td>
<td>n/a</td>
<td>Page On 65,536 bytes (hell...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA1</td>
<td>00:00.197.134</td>
<td>5.28 μs</td>
<td>n/a</td>
<td>Unknown 0x1d01ff61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA1</td>
<td>00:00.197.209</td>
<td>1.76 μs</td>
<td>n/a</td>
<td>Unknown 0x0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA0</td>
<td>00:00.197.239</td>
<td>20.32 μs</td>
<td>n/a</td>
<td>Page On 65,536 bytes (hell...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sDMA0</td>
<td>00:00.197.261</td>
<td>185.92 μs</td>
<td>n/a</td>
<td>Page On 1,048,576 bytes (...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute1</td>
<td>00:00.240.462</td>
<td>463.52 μs</td>
<td>Frame 1</td>
<td>Command Buffer 5:Compact...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td>Process</td>
<td>Surface ID</td>
<td>Is Write</td>
<td>Width</td>
<td>Height</td>
<td>Pixel Format</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------</td>
<td>-------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>00:00.398.385</td>
<td>WindowServer (163)</td>
<td>1</td>
<td>Yes</td>
<td>3360</td>
<td>2100</td>
<td>r01l</td>
</tr>
<tr>
<td>00:00.400.645</td>
<td>hello_world.app (1310)</td>
<td>181</td>
<td>Yes</td>
<td>2696</td>
<td>1988</td>
<td>ARGB</td>
</tr>
<tr>
<td>00:00.415.237</td>
<td>WindowServer (163)</td>
<td>2</td>
<td>Yes</td>
<td>3360</td>
<td>2100</td>
<td>r01l</td>
</tr>
<tr>
<td>00:00.415.445</td>
<td>WindowServer (163)</td>
<td>1</td>
<td>Yes</td>
<td>3360</td>
<td>2100</td>
<td>r01l</td>
</tr>
<tr>
<td>00:00.415.445</td>
<td>WindowServer (163)</td>
<td>2</td>
<td>Yes</td>
<td>3360</td>
<td>2100</td>
<td>r01l</td>
</tr>
<tr>
<td>00:00.449.681</td>
<td>WindowServer (163)</td>
<td>181</td>
<td>No</td>
<td>2696</td>
<td>1988</td>
<td>ARGB</td>
</tr>
<tr>
<td>00:00.449.681</td>
<td>WindowServer (163)</td>
<td>60</td>
<td>Yes</td>
<td>473</td>
<td>473</td>
<td>ARGB</td>
</tr>
<tr>
<td>00:00.449.681</td>
<td>WindowServer (163)</td>
<td>12</td>
<td>Yes</td>
<td>3360</td>
<td>2100</td>
<td>r01l</td>
</tr>
<tr>
<td>00:00.449.681</td>
<td>WindowServer (163)</td>
<td>32</td>
<td>Yes</td>
<td>297</td>
<td>297</td>
<td>ARGB</td>
</tr>
<tr>
<td>00:00.449.865</td>
<td>WindowServer (163)</td>
<td>2</td>
<td>Yes</td>
<td>3360</td>
<td>2100</td>
<td>r01l</td>
</tr>
<tr>
<td>00:00.449.865</td>
<td>WindowServer (163)</td>
<td>12</td>
<td>Yes</td>
<td>3360</td>
<td>2100</td>
<td>r01l</td>
</tr>
<tr>
<td>00:00.465.730</td>
<td>WindowServer (163)</td>
<td>60</td>
<td>Yes</td>
<td>473</td>
<td>473</td>
<td>ARGB</td>
</tr>
<tr>
<td>00:00.465.730</td>
<td>WindowServer (163)</td>
<td>32</td>
<td>Yes</td>
<td>297</td>
<td>297</td>
<td>ARGB</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Event ID</td>
<td>Event Type</td>
<td>State</td>
<td>Value</td>
<td>Duration</td>
<td>Process</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>------------</td>
<td>--------</td>
<td>-------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>00:00.050.435</td>
<td>72,057,59...</td>
<td>Shared</td>
<td>Signal</td>
<td>630</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.050.435</td>
<td>1</td>
<td>Shared</td>
<td>Signal</td>
<td>630</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.050.486</td>
<td>1</td>
<td>Shared</td>
<td>Wait</td>
<td>630</td>
<td>30.04 ms</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.128.401</td>
<td>72,057,59...</td>
<td>Shared</td>
<td>Signal</td>
<td>631</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.128.401</td>
<td>1</td>
<td>Shared</td>
<td>Signal</td>
<td>631</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.128.422</td>
<td>1</td>
<td>Shared</td>
<td>Wait</td>
<td>631</td>
<td>30.22 ms</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.206.850</td>
<td>72,057,59...</td>
<td>Shared</td>
<td>Signal</td>
<td>632</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.206.850</td>
<td>1</td>
<td>Shared</td>
<td>Signal</td>
<td>632</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.206.886</td>
<td>1</td>
<td>Shared</td>
<td>Wait</td>
<td>632</td>
<td>30.24 ms</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.284.989</td>
<td>72,057,59...</td>
<td>Shared</td>
<td>Signal</td>
<td>633</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
<tr>
<td>00:00.284.989</td>
<td>1</td>
<td>Shared</td>
<td>Signal</td>
<td>633</td>
<td>n/a</td>
<td>hello_word</td>
</tr>
</tbody>
</table>
Optimizing for 8K video editing
Support for high dynamic range
Leveraging all platform resources
Efficient data transfers
Efficient Data Transfers

Bandwidth and Mac Pro configurations

Transfer strategies with Infinity Fabric Link

Unlocking challenging workflows
Baseline Transfer Rates
Baseline Transfer Rates

- Thunderbolt 3
- Dual PCIe x16
- Infinity Fabric Link

PCIe Gen3 x16
Baseline Transfer Rates

- Thunderbolt 3
- Dual PCIe x16
- Infinity Fabric Link
Mac Pro Dual GPU Configuration

Heavy bandwidth

SoC

CPU

Apple Afterburner

AMD Radeon Pro Vega II

GPU 2

AMD Radeon Pro Vega II

GPU 1

Pro Display XDR

NEW
Mac Pro Dual GPU Configuration

Heavy bandwidth

SoC

CPU

PCIe x16

Apple Afterburner

AMD Radeon Pro Vega II

GPU 2

Pro Display XDR

PCIe x16

GPU 1

PCIe x16

AMD Radeon Pro Vega II

NEW

NEW
Mac Pro Dual GPU Configuration

Heavy bandwidth

SoC

CPU

PCle x16

Apple Afterburner

AMD Radeon Pro Vega II

GPU 2

PCle x16

AMD Radeon Pro Vega II

GPU 1

PCle x16

Pro Display XDR
Mac Pro Dual GPU Configuration

Heavy bandwidth

- SoC
- CPU
- PCIe x16
- Apple Afterburner
- AMD Radeon Pro Vega II
- Infinity Fabric Link
- GPU 2
- PCIe x16
- GPU 1
- PCIe x16
- AMD Radeon Pro Vega II
- Pro Display XDR
Mac Pro Quad GPU Configuration

Heavy compute

SoC

CPU

AMD Radeon Pro Vega II Duo

GPU 1

GPU 2

AMD Radeon Pro Vega II Duo

GPU 3

GPU 4

Pro Display XDR

NEW
Mac Pro Quad GPU Configuration

Heavy compute

SoC

CPU

PCIe x16

AMD Radeon Pro Vega II Duo

GPU 1

GPU 2

PCIe x16

AMD Radeon Pro Vega II Duo

GPU 3

GPU 4

Pro Display XDR

NEW
Mac Pro Quad GPU Configuration
Heavy compute

SoC

CPU

PCIe x16

AMD Radeon Pro Vega II Duo

Infinity Fabric Link

GPU 1

GPU 2

PCIe x16

AMD Radeon Pro Vega II Duo

Infinity Fabric Link

GPU 3

GPU 4

Pro Display XDR
Transfer Strategies with Infinity Fabric

Many transfer schemes

Consider your software architecture

Goal is high bandwidth efficiency
Infinity Fabric Transfer Strategies
Transfer entire frames

Odd frames
Frame 1, 3, 5, ...
Aux GPU

Even frames
Frame 2, 4, 6, ...
Display GPU

Infinity Fabric Link

Display GPU
Display
Infinity Fabric Transfer Strategies
Transfer entire frames

Odd frames

Frame 1, 3, 5, ...

Aux GPU

Infinity Fabric Link

Even frames

Frame 2, 4, 6, ...

Display GPU

Display
Infinity Fabric Transfer Strategies

Transfer tiles

Tiles → Aux GPU

Infinity Fabric Link

Tiles → Display GPU

Display GPU → Display
Infinity Fabric Transfer Strategies

Transfer tiles

Tiles -> Aux GPU

Infinity Fabric Link

Tiles -> Display GPU

Display GPU -> Display
Final Cut Pro X

Efficient scaling on CPU and GPU
Making full use of Metal Peer Group API
More streams of 8K ProRes video
Single GPU PCIe Transfer

Preroll → 30fps–33.3 ms

Frame 1  Frame 2  Frame 3  Frame 4

CPU
PCIe
GPU
Display
Single GPU PCIe Transfer

<table>
<thead>
<tr>
<th>Preroll</th>
<th>30fps–33.3 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame 1</td>
<td>Frame 2</td>
</tr>
<tr>
<td>Frame 3</td>
<td>Frame 4</td>
</tr>
</tbody>
</table>

- **CPU**
- **PCIe**
- **GPU**
- **Display**
Single GPU PCIe Transfer

Preroll → 30fps–33.3 ms

Frame 1  Frame 2  Frame 3  Frame 4

CPU

PCIe

GPU

Display
Single GPU PCIe Transfer

- Preroll
- 30fps–33.3 ms

Frame 1
CPU
1

PCIe
1 1 1

GPU

Display

Frame 2
Frame 3
Frame 4
Dual GPU PCIe Transfer

Preroll → 30fps–33.3 ms

<table>
<thead>
<tr>
<th>CPU</th>
<th>PCIe 1</th>
<th>GPU 1 (aux)</th>
<th>PCIe 2</th>
<th>GPU 2 (display)</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame 1</td>
<td>Frame 2</td>
<td>Frame 3</td>
<td>Frame 4</td>
<td>Frame 5</td>
<td></td>
</tr>
</tbody>
</table>

30fps–33.3 ms
# Dual GPU PCIe Transfer

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
<th>Frame 4</th>
<th>Frame 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCIe 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU 1 (aux)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCIe 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU 2 (display)</td>
<td>Process 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Preroll: 30fps–33.3 ms
Dual GPU PCIe Transfer

- **Preroll**: 30fps–33.3 ms

<table>
<thead>
<tr>
<th>Frame</th>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>GPU 2</strong></td>
<td><strong>GPU 1</strong></td>
</tr>
<tr>
<td>2</td>
<td><strong>Display</strong></td>
<td><strong>PCIe 2</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>PCIe 1</strong></td>
<td><strong>CPU</strong></td>
</tr>
<tr>
<td>4</td>
<td><strong>CPU</strong></td>
<td><strong>GPU 2</strong></td>
</tr>
<tr>
<td>5</td>
<td><strong>GPU 1</strong></td>
<td><strong>GPU 1</strong></td>
</tr>
</tbody>
</table>

- **Display**: 1
- **Process 1**: 2
- **Process 2**: 2

- **GPU 2** (display)
- **GPU 1** (aux)
- **PCIe 2**
- **PCIe 1**
- **CPU**
Dual GPU PCIe Transfer

- **CPU**: 1, 2, 3
- **PCIe 1**: 2, 2, 2
- **GPU 1 (aux)**: Process 2
- **PCIe 2**: 1, 1, 1
- **GPU 2 (display)**: Process 1, Process 3
- **Display**: 1, 2, 3, 4

**Frameewater**: 30fps - 33.3 ms

- **Frame 1**: 2, 4, 4, 4
- **Frame 2**: 4
- **Frame 3**: 4
- **Frame 4**: 4
- **Frame 5**: 4
Dual GPU PCIe Transfer

Frame 1
- CPU: 1, 3, 4
- PCIe 1: 2, 2, 2
- GPU 1 (aux): Process 2
- PCIe 2: 1, 1, 1
- GPU 2 (display): Process 1
- Display: 1

Frame 2
- CPU: 1, 3, 4
- PCIe 1: 2, 4, 4
- GPU 1 (aux): Process 4
- PCIe 2: 3, 3, 3
- GPU 2 (display): Process 3
- Display: 2

Frame 3
- CPU: 1, 3, 4
- PCIe 1: 2
- GPU 1 (aux): Process 4
- PCIe 2: 2
- GPU 2 (display): Process 3
- Display: 3

Frame 4
- CPU: 1, 3, 4
- PCIe 1: 4
- GPU 1 (aux): Process 4
- PCIe 2: 4
- GPU 2 (display): Process 3
- Display: 4

Frame 5
- CPU: 1, 3, 4
- PCIe 1: 4
- GPU 1 (aux): Process 4
- PCIe 2: 4
- GPU 2 (display): Process 3
- Display: 4

Preroll: 30fps–33.3 ms

Processes:
- Process 1
- Process 2
- Process 3
- Process 4

Note: (aux) indicates auxiliary GPU (used for general processing), (display) indicates the display GPU (used for rendering)
Dual GPU PCIe Transfer

Preroll: 30fps–33.3 ms

Frame 1
- GPU 1 (aux)
  - Process 2

Frame 2
- GPU 2 (display)
  - Process 1
  - Process 3

Frame 3
- PCIe 2
  - 1 1 1
  - 3 3 3
- GPU 2 (display)
  - 3

Frame 4
- PCIe 1
  - 2 2 2
  - 4 4 4
- GPU 1 (aux)
  - Process 4

Frame 5
- CPU
  - 1 3
Dual GPU PCIe Transfer
Infinity Fabric Link

Preroll 30fps–33.3 ms

Frame 1 Frame 2 Frame 3 Frame 4 Frame 5

CPU 1 2 4

PCIe 1 2 2 2

GPU 1 (aux) Process 2

Infinity Fabric

PCIe 2 1 1 1 3 3 3

GPU 2 (display) Process 1 Process 3

Display 1 3

Process 4

( aux )

(display)
Dual GPU PCIe Transfer

Infinity Fabric Link

Preroll ➔ 30fps–33.3 ms

Frame 1 ➔ Frame 2 ➔ Frame 3 ➔ Frame 4 ➔ Frame 5

CPU

PCIe 1

GPU 1 (aux)

Infinity Fabric

PCIe 2

GPU 2 (display)

Display

1. Process 1
2. Process 2
3. Process 3
4. Process 4

Display 1 ➔ Display 2 ➔ Display 3 ➔ Display 4

Frame 1 ➔ Frame 2 ➔ Frame 3 ➔ Frame 4 ➔ Frame 5

1. Preroll
2. 30fps–33.3 ms
Dual GPU PCIe Transfer

Infinity Fabric Link

30fps–33.3 ms

Frame 1: Process 2

Frame 2: Process 4

Frame 3: Infinity Fabric link

Frame 4: GPU 1 (aux)

Frame 5: GPU 2 (display)
Detecting Infinity Fabric Configurations

// Query for Infinity Fabric connections
let gpuPeerGroupID = device.peerGroupID
let gpuPeerIndex = device.peerIndex
let gpuPeerCount = device.peerCount
let gpuLocationNumber = device.locationNumber

<table>
<thead>
<tr>
<th></th>
<th>peerGroupID</th>
<th>peerIndex</th>
<th>peerCount</th>
<th>locationNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked GPUs, dual PCIe AMD Radeon Pro Vega II</td>
<td>Equal</td>
<td>(0..peerCount-1)</td>
<td>2</td>
<td>Different</td>
</tr>
<tr>
<td>Linked GPUs, shared PCIe AMD Radeon Pro Vega II Duo</td>
<td>Equal</td>
<td>(0..peerCount-1)</td>
<td>2</td>
<td>Equal</td>
</tr>
</tbody>
</table>
// Create shared event and command queues for auxiliary and display connected GPUs
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

// Encode rendering of video frame on auxiliary device
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

// Encode blit from auxiliary device to display device
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, ...)
blitCommandEncoder.endEncoding()
Create shared event and command queues for auxiliary and display connected GPUs

```swift
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!
```

Encode rendering of video frame on auxiliary device

```swift
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)
```

Encode blit from auxiliary device to display device

```swift
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, ...)
blitCommandEncoder.endEncoding()
```
Create shared event and command queues for auxiliary and display connected GPUs

```swift
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!
```

Encode rendering of video frame on auxiliary device

```swift
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)
```

Encode blit from auxiliary device to display device

```swift
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, …)
blitCommandEncoder.endEncoding()
```
// Create shared event and command queues for auxiliary and display connected GPUs
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

// Encode rendering of video frame on auxiliary device
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

// Encode blit from auxiliary device to display device
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, …)
blitCommandEncoder.endEncoding()
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, ...)
blitCommandEncoder.endEncoding()
Metal Peer Group API

Transfer between GPUs at high speed
Take advantage of parallel channel
Reduce PCIe traffic
Enable challenging workflows
18 Core CPU
4 External GPUs
4 External GPUs
Metal for Pro Apps

Takeaways
Optimize your apps for 8K content
Metal for Pro Apps
Takeaways

Optimize your apps for 8K content

Support HDR TVs and displays
Metal for Pro Apps

Takeaways

Optimize your apps for 8K content
Support HDR TVs and displays
Scale performance based on platform resources
Metal for Pro Apps

Takeaways

Optimize your apps for 8K content

Support HDR TVs and displays

Scale performance based on platform resources

Leverage Infinity Fabric Link with Metal Peer Group API
## More Information

developer.apple.com/wwdc19/608

<table>
<thead>
<tr>
<th>Lab</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metal for Pro Apps Lab</td>
<td>Thursday, 9:00</td>
</tr>
<tr>
<td>Metal Lab</td>
<td>Friday, 9:00</td>
</tr>
<tr>
<td>Metal for Machine Learning and Ray Tracing Lab</td>
<td>Friday, 12:00</td>
</tr>
</tbody>
</table>