Vulkan synchronization for WebGPU

Albin Bernhardsson, Arm
WebGPU

+ Successor to WebGL
+ API for accessing the GPU ... on the web!
Differences to WebGL

- No global state
- Command buffers
- Pipelines
- Bind groups
- Render/compute passes
- Compute shaders
Differences to Vulkan

- Some features/extensions not available (yet)
  - VRS
  - Raytracing
  - Bindless
  - Geometry shaders
  - Etc.

- No explicit memory management
- No explicit synchronization
No explicit synchronization?

- So the browser is responsible
  - Rather, the WebGPU implementation in your browser
- In Chromium, that’s Dawn
  - https://dawn.googlesource.com/dawn
- Implements WebGPU on top of Vulkan
  - (Also DirectX 12, Metal and GLES)
Synchronization in Dawn

User submits:
• GPUCommandBuffer
• Without synchronization

We need:
• VkCommandBuffer
• With pipeline barriers
When recording `GPUCommandEncoder`

```plaintext
pass = cmd.beginRenderPass(mainPass)
pass.setPipeline(meshPipeline)
pass.setBindGroup(cameraMatricesBindGroup)
pass.setBindGroup(textureBindGroup)

// Draw table
pass.setVertexBuffer(tableVertices)
pass.setBindGroup(tableBindGroup)
pass.draw(123)

// Draw chair
pass.setVertexBuffer(chairVertices)
pass.setBindGroup(chairBindGroup)
pass.draw(234)
```
When submitting GPUCommandBuffer

```cpp
void CommandBuffer::BeginRenderPass (syncScope) {
    for (auto t : syncScope.textures) {
        t.texture->RecordPipelineBarrier(
            t.usage);
    }

    for (auto b : syncScope.buffers) {
        b.buffer->RecordPipelineBarrier(
            b.usage);
    }

    vkCmdBeginRenderPass(...);
}
```

### Sync scope

<table>
<thead>
<tr>
<th>Texture</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framebuffer</td>
<td>ColorAttachment</td>
</tr>
<tr>
<td>DepthBuffer</td>
<td>DepthAttachment</td>
</tr>
<tr>
<td>BurntWoodTexture</td>
<td>TextureBinding</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CameraMatrices</td>
<td>UniformBuffer</td>
</tr>
<tr>
<td>TableVertices</td>
<td>VertexBuffer</td>
</tr>
<tr>
<td>TableUniforms</td>
<td>UniformBuffer</td>
</tr>
<tr>
<td>ChairVertices</td>
<td>VertexBuffer</td>
</tr>
<tr>
<td>ChairUniforms</td>
<td>UniformBuffer</td>
</tr>
</tbody>
</table>
void Texture::RecordPipelineBarrier(
    TextureUsage usage)
{
    VkImageMemoryBarrier barrier = {};  
    [...]
    barrier.accessMask = GetAccessMask(usage);
    barrier.imageLayout = GetImageLayout(usage);

    VkPipelineStageFlags stages = 
        GetPipelineStage(usage);

    vkCmdPipelineBarrier(...);
}
Recording a barrier

 [...]  
 barrier.srcAccessMask = GetAccessMask(mLastUsage);  
 barrier.oldImageLayout = GetImageLayout(mLastUsage);  
 VkPipelineStageFlags srcStage = GetPipelineStage(mLastUsage);  
 mLastUsage = usage;  
 [...]
Recording a barrier

- GetPipelineStage(usage)
  - The most common way to do Vulkan synchronization?

- Used in:
  - Dawn
  - Other open source WebGPU implementation
  - Vulkan applications
  - Commercial game engines

- Let’s hope there are no issues with it...
The issue with GetPipelineStage(usage)

```cpp
void Texture::RecordPipelineBarrier(
    TextureUsage usage)
{
    VkImageMemoryBarrier barrier = {};
    [...] 
    barrier.accessMask = GetAccessMask(usage);
    barrier.imageLayout = GetImageLayout(usage);

    VkPipelineStageFlags stage = 
        GetPipelineStage(usage);

    vkCmdPipelineBarrier(...);
}
```

Example

```
TextureBinding

VK_ACCESS_SHADER_READ_BIT
VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL

VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
| VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
| VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
| VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
| VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
| VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
| VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT
| VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT
| VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR
```
A simple frame

Vertex

Fragment
... on a tiled GPU
... on a tiled GPU
... on a tiled GPU
Hold it!

It looks like you have a dependency. Would you like a barrier?

Yes

srcStages = LATE_FRAGMENT_TESTS
dstStages = FRAGMENT_SHADER | VERTEX_SHADER | COMPUTE_SHADER
Hold it!

**srcStages** = LATE_FRAGMENT_TESTS
**dstStages** = FRAGMENT_SHADER | VERTEX_SHADER | COMPUTE_SHADER
The problem

- `GetPipelineStage(usage)`
  - Can’t tell if TextureBinding is Vertex, Fragment or Compute

- `GetPipelineStage(usage, shaderStages)`
  - Trivial to determine
## Tracking shader stages

Sync scope

<table>
<thead>
<tr>
<th>Texture</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framebuffer</td>
<td>ColorAttachment</td>
</tr>
<tr>
<td>DepthBuffer</td>
<td>DepthAttachment</td>
</tr>
<tr>
<td>BurntWoodTexture</td>
<td>TextureBinding</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CameraMatrices</td>
<td>UniformBuffer</td>
</tr>
<tr>
<td>TableVertices</td>
<td>VertexBuffer</td>
</tr>
<tr>
<td>TableUniforms</td>
<td>UniformBuffer</td>
</tr>
<tr>
<td>ChairVertices</td>
<td>VertexBuffer</td>
</tr>
<tr>
<td>ChairUniforms</td>
<td>UniformBuffer</td>
</tr>
</tbody>
</table>
## Tracking shader stages

### Sync scope

<table>
<thead>
<tr>
<th>Texture</th>
<th>Usage</th>
<th>Shader stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framebuffer</td>
<td>ColorAttachment</td>
<td>None</td>
</tr>
<tr>
<td>DepthBuffer</td>
<td>DepthAttachment</td>
<td>None</td>
</tr>
<tr>
<td>BurntWoodTexture</td>
<td>TextureBinding</td>
<td>Fragment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Usage</th>
<th>Shader stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>CameraMatrices</td>
<td>UniformBuffer</td>
<td>Vertex</td>
</tr>
<tr>
<td>TableVertices</td>
<td>VertexBuffer</td>
<td>None</td>
</tr>
<tr>
<td>TableUniforms</td>
<td>UniformBuffer</td>
<td>Vertex</td>
</tr>
<tr>
<td>ChairVertices</td>
<td>VertexBuffer</td>
<td>None</td>
</tr>
<tr>
<td>ChairUniforms</td>
<td>UniformBuffer</td>
<td>Vertex</td>
</tr>
</tbody>
</table>
How to find the shader stages?

- Explicit tagging? Shader analysis?
- User already told us
- `BindGroupLayoutEntry.visibility`
  - Mask of all shader stages the resource is used in.
Recording a barrier

- From
  - `GetPipelineStage(usage)`

- To
  - `GetPipelineStage(usage, shaderStages)`
Results

srcStages = LATE_FRAGMENTS_TESTS
dstStages = FRAGMENT_SHADER | VERTEX_SHADER | COMPUTE_SHADER
Results

```
srcStages = LATE_FRAGMENT_TESTS
dstStages = FRAGMENT_SHADER
```
## Results

- Measured on an Immortalis-G715 device

![Graph showing frame times for different functions](image-url)

- `GetPipelineStages(usage)`: 100%
- `GetPipelineStages(usage, shaderStages)`: 85%
When submitting GPUCommandBuffer

```cpp
void CommandBuffer::BeginRenderPass (syncScope) {
    for (auto t : syncScope.textures) {
        t.texture->RecordPipelineBarrier(
            t.usage);
    }

    for (auto b : syncScope.buffers) {
        b.buffer->RecordPipelineBarrier(
            b.usage);
    }

    vkCmdBeginRenderPass(...);
}
```

Sync scope

<table>
<thead>
<tr>
<th>Texture</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framebuffer</td>
<td>ColorAttachment</td>
</tr>
<tr>
<td>DepthBuffer</td>
<td>DepthAttachment</td>
</tr>
<tr>
<td>BurntWoodTexture</td>
<td>TextureBinding</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CameraMatrices</td>
<td>UniformBuffer</td>
</tr>
<tr>
<td>TableVertices</td>
<td>VertexBuffer</td>
</tr>
<tr>
<td>TableUniforms</td>
<td>UniformBuffer</td>
</tr>
<tr>
<td>ChairVertices</td>
<td>VertexBuffer</td>
</tr>
<tr>
<td>ChairUniforms</td>
<td>UniformBuffer</td>
</tr>
</tbody>
</table>
Barrier merging

- One vkCmdPipelineBarrier per resource is a lot...
- vkCmdPipelineBarrier can contain many memory barriers
- Let’s merge them into one!
Barrier merging example

- First barrier: TRANSFER → VERTEX
- Second barrier: FRAGMENT → FRAGMENT
- Merged barrier: TRANSFER | FRAGMENT → VERTEX | FRAGMENT
Better barrier merging

- Two `vkCmdPipelineBarrier`'s
  - One for everything with VERTEX in its `dstStageMask`
  - One for everything else

- (Or synchronization2)
Conclusions

+ If you’re doing:
  • GetPipelineStage(usage)

+ Consider:
  • GetPipelineStage(usage, shaderStages)

+ And save 15%
Resources

- Synchronization validation layers
  • See talk: Using Vulkan Synchronization Validation Effectively

- Vulkan Samples
  • Pipeline barriers sample

- Full details on the Dawn commit
  • https://dawn-review.googlesource.com/c/dawn/+/151340
Build the future on Arm

- Join our developer community
- Connect with like-minded developers
- Get fresh insights from Arm experts
- **Sign up:** arm.com/developerprogram

Find out more

- **Talk to us via Discord** ArmSoftwareDev
- **Docs & Resources:** arm.com/vulkan
- **Forums and blogs:** community.arm.com

© 2024 Arm