GPU SUBMISSION STRATEGIES

MATTIAS LILJESON
GPU DEVELOPER TECHNOLOGY ENGINEER, AMD
WHOAMI

I work as a DevTech at AMD!

My role can be summarized as:

*I help game developers make the most out of our hardware*

Fun fact:

*I always have a Justin Bieber poster in my office!*
GPU WORK SUBMISSION

Some background
FROM A HIGH-LEVEL VIEW

- Draw...
- Draw...
- Draw...
NAÏVE APPROACH

Draw  GPU Draw

Draw  GPU Draw

Time
NAÏVE APPROACH

*Also the driver stack, we’ll talk about that a bit later*
HOW WORK IS SUBMITTED TO THE GPU

- Command list:
  - Draw...
  - Draw...
  - Draw...

[Diagram showing command list (CL) with arrows to another command list (CL) with more draw commands]
IF DRAWING WAS SYNCHRONOUS

- Record CLs
- latency
- GPU render
- latency
- Record CLs
- latency
- GPU render
- Frame n
- Frame n+1

* PCIe bus adds latency*

* Also the driver stack, we’ll talk about that a bit later*
SYNCHRONOUS VS ASYNCHRONOUS

Task 1
Task 2
Task 3

Time

Task 1
Task 2
Task 3

Time
WITH AN ASYNC APPROACH

Rendering while writing to the same CLs?
- Not possible!
  - Solved with multiple buffering

Start rendering before previous frame has finished?
- Not possible!
  - Solved with semaphores/fences
WHAT ABOUT MULTIPLE SUBMISSIONS?

In most games we are not doing a single submit per frame though

• Let’s look at different submission patterns
• How they will be executed on the HW
• We will use a real-world example
• And some suitable tools
TODAY'S TOOLS

GPUView

- Tool developed by Microsoft
- Holistic CPU and GPU overview
- Can be used to answer questions like:
  - Are we CPU bound or GPU bound?
  - Or both, during different parts of the frame?
  - Are stutters due to memory allocations in-between frames or something else?
TODAY'S TOOLS

Radeon™ GPU Profiler (RGP)
- A profiling tool for AMD graphic cards
- In depth analysis of a GPU frame
- Graphics and async compute usage
- Event timing
- Barriers
- Instruction timing
- …
THE EXAMPLE APPLICATION

- $64 \times 32 = 2048$ simple meshes
- Each mesh = 1089 tris
- Simple textured shading
This application is designed to show limits

It’s not an example of good renderer design
THE BASE CASE

1 Command list for everything in the whole frame

- Barrier: framebuffer from **present** to **render target**
- Clear render target (back buffer) and depth buffer
- Draw 2048 boomboxes
- Barrier: framebuffer from **render target** to **present**
### THE BASE CASE

- **Operating System (OS)**: Microsoft Windows 10 Pro 19043
- **Processor**: AMD Ryzen 7 5800X
- **Motherboard**: TUF GAMING X570-PLUS
- **RAM**: 32.0 GB DDR4
- **GPU**: AMD Radeon RX 6800 XT
- **Driver**: AMD Software Adrenalin Edition Version 22.6.1

---

![Diagram showing GPU command lists](image)

- The app's color (blue)
- The app's threads
- The apps command lists on the GPU (blue)
- Calls to present
- Command list on the GPU as seen by the GPU (blue)
THE BASE CASE

- OS: Microsoft Windows 10 Pro 19043
- Processor: AMD Ryzen 7 5800X
- Motherboard: TUF GAMING X570-PLUS
- RAM: 32.0 GB DDR4
- GPU: AMD Radeon RX 6800 XT
- Driver: AMD Software Adrenalin Edition Version 22.6.1

This graph shows enqueued work over time. The blocks at the bottom are tasks being worked on by the GPU. Enqueued work is stacked on top of that. You can also see fences, signals and other synchronization going on here in the frame (pink). Command list on the GPU. The same command list as seen by the app.
THE BASE CASE

0.35 ms, present-to-present

“Holes”. The GPU is idle parts of the frame

The CPU is constantly busy, no “holes”

<table>
<thead>
<tr>
<th>OS</th>
<th>Microsoft Windows 10 Pro 19043</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>AMD Ryzen 7 5800X</td>
</tr>
<tr>
<td>Motherboard</td>
<td>TUF GAMING X570-PLUS</td>
</tr>
<tr>
<td>RAM</td>
<td>32.0 GB DDR4</td>
</tr>
<tr>
<td>GPU</td>
<td>AMD Radeon RX 6800 XT</td>
</tr>
<tr>
<td>Driver</td>
<td>AMD Software Adrenalin Edition Version 22.6.1</td>
</tr>
</tbody>
</table>
THE BASE CASE

Frame nr

<table>
<thead>
<tr>
<th>Frame</th>
<th>Start Time</th>
<th>End Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame 510</td>
<td>2,242 ms</td>
<td>3,080 ms</td>
<td>838 ms</td>
</tr>
<tr>
<td>Frame 511</td>
<td>3,080 ms</td>
<td>3,107 ms</td>
<td>27 ms</td>
</tr>
<tr>
<td>Frame 512</td>
<td>3,107 ms</td>
<td>3,750 ms</td>
<td>643 ms</td>
</tr>
<tr>
<td>Frame 513</td>
<td>4,415 ms</td>
<td>4,430 ms</td>
<td>15 ms</td>
</tr>
<tr>
<td>Frame 514</td>
<td>4,430 ms</td>
<td>5,040 ms</td>
<td>610 ms</td>
</tr>
<tr>
<td>Frame 515</td>
<td>5,040 ms</td>
<td>5,060 ms</td>
<td>20 ms</td>
</tr>
<tr>
<td>Frame 516</td>
<td>5,060 ms</td>
<td>6,000 ms</td>
<td>940 ms</td>
</tr>
</tbody>
</table>

System Configuration:
- OS: Microsoft Windows 10 Pro 19043
- Processor: AMD Ryzen 7 5800X
- Motherboard: TUF GAMING X570-PLUS
- RAM: 32.0 GB DDR4
- GPU: AMD Radeon RX 6800 XT
- Driver: AMD Software Adrenalin Edition Version 22.6.1
THE BASE CASE

Frame time: 0.84ms (1190 fps)

1st submit, “our” submit
2nd submit, inserted by driver
THE BASE CASE

- **Os**: Microsoft Windows 10 Pro 19043
- **Processor**: AMD Ryzen 7 5800X
- **Motherboard**: TUF GAMING X570-PLUS
- **RAM**: 32.0 GB DDR4
- **GPU**: AMD Radeon RX 6800 XT
- **Driver**: AMD Software Adrenalin Edition Version 22.6.1
THE BASE CASE

OS: Microsoft Windows 10 Pro 19043
Processor: AMD Ryzen 7 5800X
Motherboard: TUF GAMING X570-PLUS
RAM: 32.0 GB DDR4
GPU: AMD Radeon RX 6800 XT
Driver: AMD Software Adrenalin Edition Version 22.6.1

- Heavy VS workload
- 50% achieved occupancy
- Many tasks in parallel

~0.140ms
WITH SMALL BATCHES

Command lists with a few draws

1: Barrier: framebuffer from present to render target
   Clear render target (back buffer) and depth buffer

2 – 257: Draw 2048 boomboxes in batches of 8. (8 meshes per CL)

258: Barrier: framebuffer from render target to present

All CLs submitted one-by-one
WITH SMALL BATCHES

6.45 ms, present-to-present

Lots of synchronization (pink)

The CPU is idle part of the frame

Many small command lists (blue)
WITH SMALL BATCHES

Small holes

Many small CLs

Low achieved occupancy

Larger gaps
WITH SMALL BATCHES

Many small CLs

Holes between CLs
WITH MULTI THREADING

Command lists with a few draws

1: Barrier: framebuffer from present to render target
   Clear render target (back buffer) and depth buffer

2 – 257: Draw 2048 boomboxes in batches of 8. (8 meshes per CL)
   As 256 tasks from a thread pool

258: Barrier: framebuffer from render target to present

All CLs submitted one-by-one from their respective thread
WITH MULTI THREADING

8.20 ms, present-to-present

Many small command lists

The threads are often idle
WITH MULTI THREADING

Looks pretty much like the last case
THAT DOESN’T LOOK TOO GOOD
SO WHY DO WE SEE WHAT WE SEE?

• Why does performance tank so badly with many submits?
• Why doesn't multithreading help?

Let’s look at the driver stack!

• In this case, let’s use our open-source vulkan driver as an example
• Since it’s open source, we can freely discuss its details
• The source code is available online here: https://github.com/GPUOpen-Drivers/AMDVLK
THE DRIVER STACK

- This is what our Open Source driver look like
  - The application is on top and calls the vulkan runtime, libvulkan
  - The runtime talks to the upmost part of the driver, amdvlk64
  - The red upper part, amdvlk64 talks to libdrm_amdgpu which communicates with the lower orange part amdgpu
  - amdgpu talks to the graphics card

Drm in this case stands for Direct Rendering Manager
- Basically an API for modern GPU drivers in linux
THE DRIVER STACK

• The interesting part of this picture is this line here:
  • It divides the driver into user and kernel space parts

• To protect the system and the user, only the operating system kernel can access the hardware
  • The CPU can therefore execute in **user space** and **kernel space**
  • Switching between the two takes time

Important note:

*Not all calls to the user mode part of the driver end up causing a mode switch to kernel mode*
THE DRIVER STACK

- The driver also needs to support multiple applications using the graphics card
- The kernel mode part therefore supports scheduling GPU work for multiple applications

![Diagram](image)

- Vulkan application
- libvulkan
- amdvlk64
  - XGL
  - PAL
- libdrm_amdgpu
- Another application using the GPU
  - (Rendering, video decode)

User space
Kernel space

THE DRIVER STACK

- The driver also needs to support multiple applications using the graphics card
- The kernel mode part therefore supports scheduling GPU work for multiple applications

![Diagram](image)

- Vulkan application
- libvulkan
- amdvlk64
  - XGL
  - PAL
- libdrm_amdgpu
- Another application using the GPU
  - (Rendering, video decode)

User space
Kernel space
THE DRIVER STACK, ON WINDOWS

- There’s a similar setup on windows
- The driver is split up in user mode and kernel mode parts
  - The model is called: Windows Display Driver Model
- There is a compositor that handles multiple applications rendering at the same time
  - It is called: Desktop Window Manager
CPU ➔ GPU COMMUNICATION

- The CPU communicates with the GPU over a ring buffer
- The ring buffer has two pointers:
  - The CPU writes data and moves the write pointer forward
  - The GPU reads data and moves the read pointer forward
The CPU communicates with the GPU over a ring buffer.

The ring buffer has two pointers:
- The CPU writes data and moves the write pointer forward.
- The GPU reads data and moves the read pointer forward.
CPU → GPU COMMUNICATION

- The ring buffer contains PM4 packets which contain:
  - State settings
  - Draws
  - And other things telling the GPU what to do
- PM4 packets travel over the PCIe bus
  - The Command Processor on the GPU receives these packets and configures the GPU accordingly
WHAT CAN WE DO WITH THIS KNOWLEDGE?

• We learnt that the driver and runtime is a few layers thick

• Some calls will end up in the kernel
  • Causing a mode switch
  • Not cheap
  • We should try to limit the amount of mode switches

• We also saw how packets with instructions are sent to the GPU
  • By writing to the ring buffer and moving pointers forward
  • That communication over the PCIe bus is not free either
  • We also mentioned the latency issues previously 😞
WITH MULTI THREADING, BUT WITH A SINGLE SUBMIT

Command lists with a few draws

1: Barrier: framebuffer from present to render target
   Clear render target (back buffer) and depth buffer

2 – 257: Draw 2048 boomboxes in batches of 8. (8 meshes per CL)
   As 256 tasks from a thread pool

258: Barrier: framebuffer from render target to present

All CLs submitted in one go from the main thread
WITH MULTI THREADING, BUT WITH A SINGLE SUBMIT
WITH MULTI THREADING, BUT WITH A SINGLE SUBMIT

Small/few holes GPU Hardware Queue

0.77 ms, present-to-present

No holes in the app’s GPU Queue, Multiple frames queued

Small Command lists have been merged by the driver

Main thread seldom idle
WITH MULTI THREADING, BUT WITH A SINGLE SUBMIT

Clustered, as seen in GPUView

Low “height”
BETTER, BUT STILL HORRENDOUS
**WITH SANE MULTI THREADING**

Command lists with a few draws

1: Barrier: framebuffer from **present** to **render target**  
Clear render target (back buffer) and depth buffer

2 – 10: Draw 2048 boomboxes in batches of 256. (256 meshes per CL)  
As 8 tasks from a thread pool

11: Barrier: framebuffer from **render target** to **present**

All CLs submitted in one go from the **main thread**
WITH SANE MULTI THREADING

Holes in the GPU Hardware Queue are back

0.23 ms, present-to-present

Command lists have been merged by the driver

No holes in the app’s GPU Queue

Main thread seldom idle

Holes seems to happen during submit

<table>
<thead>
<tr>
<th>OS</th>
<th>Microsoft Windows 10 Pro 19043</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>AMD Ryzen 7 5800X</td>
</tr>
<tr>
<td>Motherboard</td>
<td>TUF GAMING X570-PLUS</td>
</tr>
<tr>
<td>RAM</td>
<td>32.0 GB DDR4</td>
</tr>
<tr>
<td>GPU</td>
<td>AMD Radeon RX 6800 XT</td>
</tr>
<tr>
<td>Driver</td>
<td>AMD Software Adrenalin Edition 22.6.1</td>
</tr>
</tbody>
</table>
WITH SANE MULTI THREADING

- Occupancy up again
- Still visible dips in the occupancy view
- Parallelism again
WITH SUBMIT IN PARALLEL WITH CL RECORDING

What we’ve been doing so far

With async submit

![Diagram showing GPU submission strategies]

Time
WITH SUBMIT IN PARALLEL WITH CL RECORDING

Holes in the GPU Hardware Queue are gone!

0.18 ms, present-to-present

A few frames have been stacked up

Main thread seldom idle

---

**Operating System**: Microsoft Windows 10 Pro 19043

**Processor**: AMD Ryzen 7 5800X

**Motherboard**: TUF GAMING X570-PLUS

**RAM**: 32.0 GB DDR4

**GPU**: AMD Radeon RX 6800 XT

**Driver**: AMD Software Adrenalin Edition Version 22.6.1
WITH SUBMIT IN PARALLEL WITH CL RECORDING
SOME CONCLUSIONS

• Submission strategy can affect performance

• Submit size matters: Avoid small CLs
  • That will keep the GPU busy

• Batch CLs up for submission
  • Not one-by-one

• Take advantage of multithreaded command list recording
  • When you see a benefit from it

• Kick off next frame CL recording early

• If unsure of good API usage, take a look at our performance guides on GPUOpen!
  • https://gpuopen.com/performance/
  • https://gpuopen.com/ryzen-performance/
THANKS

• Lou Kramer
• Gareth Thomas
• Dominik Baumeister
• Matthäus Chajdas
QUESTIONS?

mattias.liljeson@amd.com
DISCLAIMER & ATTRIBUTES

Disclaimer:

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. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2022 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Epyc, Infinity Cache, Radeon, Ryzen and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.