Compute in Vulkan

Jasper Bekkers

OTOY Inc.

October 2017

Intro

- Doing research into Real Time Path Tracing
- Used to do this in CUDA but ran into limitations
  - Single vendor
  - Gpu/Cpu synchronization overhead
  - Couldn’t use device enqueue due to runtime linking requirement
  - Limited texturing support
    - No DXT
    - No texturing from Cpu memory
  - No control over when dispatches to the Gpu happen, so lots of bubbles
- CUDA had major advantages too
  - Full and stable C++ compiler
  - Ease of use
  - Stable API and kernel binary format

What do our use cases look like

- Lots of multi-gpu with tight communication between devices
- Few but long indirect dispatches
- Huge assets so
  - Lots of migrating data, mostly in buffer objects
    - PCIe performance still not good enough
  - Lots of textures too
    - Mostly static on the Gpu
- 100% desktop / server focussed
- Not using graphics pipe at all
  - Do use swapchains & presents but that’s it
  - Write directly to swapchain from compute (except on Intel, where we can’t)

What does our API look like

- Exposed DescriptorSet as first class citizen within the API
  - Different systems create their own ‘global’ DescriptorSet
  - KernelArgs object that contains all bindings for a kernel
  - No more SetBuffer calls on the command buffer
- Exposed Heaps and Memory objects
  - Allows for more complex memory usage within the API
  - MemoryTypeIndex made things more difficult
- Exposed command buffers and synchronization
  - Semaphores and Fence’s used liberally throughout engine
  - GpuJobSystem is replacing most of the explicit synchronization

Job system

- Automatic synchronization and reordering of compute jobs and copies
  - Supports async compute
  - Supports async copies
- Simple as possible API for setting up jobs
  - Not much different than our regular API wrapped in a lambda
- Resource tracking through opaque memory handles
  - Most resources involved are transient
  - Memory can be reused after jobs end
  - No tiled resources tricks

Vulkan issues

- Bindless works but has usability issues
  - Can’t update bindings table sparsely without validation layer going complaining
  - Still no NonUniformResourceIndex equivalent
- Headless servers
- Validation layer is slow and has huge memory overhead
  - Frame that normally takes 800us now takes 22ms
- Missing equivalent to D3D12’s SetStablePowerState
  - Hacked around it by calling into D3D12

Vulkan issues

- MemoryTypeIndex not transparent enough
  - It’s not clear what buffer or texture will belong to which memory type
  - Makes it more tricky to write allocators against
- Can’t share Cpu side buffers between devices (of different vendors)
  - Nor with devices of the same vendor
  - Linear buffer only would be ok
  - VK_KHX_device_group not what we want
    - Don’t want masking in command buffer; need to have one Cpu thread per device
    - Nvidia only
  - Adds up when you need to have one staging copy per Gpu in RAM

Vulkan looking ahead

- Want our entire path tracer driven from Gpu
  - Need to do a lot of decision making now that could be kept on the Gpu with lower latency
- Would love to see a vkCmdCopy*Indirect
  - Gpu for us already decides what needs to be rendered next, but right now we need a Cpu roundtrip to get the data back in place
  - On the copy queue
  - Only need control over offsets and size
- Would love to see loops in the command buffer
  - Mantle had this as grCmdWhile / grCmdEndWhile and it’s an extremely useful building block
  - This would get rid of most of our Cpu roundtrips
- Would love to see copies between physical devices
  - Cross vendor would be ideal, but even within the same vendor would be great progress

Shading language looking ahead

- Want to start linking SPIR-V
  - Our material system would be our primary customer for this
  - Want to work closely with Pierre to make this happen
- Need to get rid of std430 and std140! Tried to add tight packing to glslang ourselves but need vendor support.
  - Have lots of code that we share with C++ so packing and alignment rules should be similar (at least to VS2017) or enforceable through #pragma's
  - Want to #include shared headers in C++ and GLSL, these mostly contain shared data structures
- Would love more explicit control over loads and stores
  - Need to use vec4, vec2 etc as hints of what stores we want on some vendors
  - I think type is the wrong place to indicate stores
  - Load/store coalescing doesn't happen at all on some vendors

Shading language looking ahead

- Would love to use workgroup sizes & count to enforce spilling and resource allocation
  - Gives easiers control over parallelism
  - CUDA has this; it would influence register allocation, shared memory size etc
    - Was amazing for performance stability (and performance tweaking)
- Would like to have dynamic indexing of uniforms in Vulkan
- Would love to be able to pass parameters into the backend compiler (-ffast-math, unrolling, register counts etc)
- Would love a mechanism for feedback from backend compiler (perf warnings, invalid SPIR-V, disassembly etc)

Shading language problems

- Would love to index into different buffers to get around the 2-4GB buffer limit size
  - Right now we need to store data together (or separate) that we don’t really want to
- Complex data structures often break the compiler / build
  - Sometimes break with dynamic loops
  - Had to manually unroll some control flow to get around issues
- Would like references to memory
  - vec3& ref = someBuffer[i].someMember[j].someOtherMember;
  - ref.x = 1;
  - // skip y
  - ref.z = 3;

Ecosystem issues

- We’ve had some difficulty getting feedback after issues have been reported to the VAP
- LunarG don’t seem to work on the most useful features first
  - Examples include 6+ month old issue on having names in the validation layer
  - Not validating extremely basic physical device limits
- Would love to contribute driver regressions / repro cases to CTS but its code-base is a huge mess
  - Ideally would want to just build my repro cases in CTS first
    - Should be small and easy to work with and upstream

Questions?

Thanks

Baldur Karlsson

Matthäus G. Chajdas

Neil Henning

Jeroen van Schijndel

Juul Joosten