GFXReconstruct - Asynchronous Pipeline Creation

Fabian Schmidt, Graphics Software Engineer, LunarG Inc.
09-24-2024

Problem Statement

GFXReconstruct is a capture and replay solution for Vulkan and DX12 applications. Recording a graphics application works by intercepting commands and storing calls sequentially in a file, including all necessary resource data. The original application might have leveraged multithreading to process its graphics calls. Common cases of this are command buffer recording or pipeline creation on multiple threads.

While GFXReconstruct keeps track of thread ids for all commands, information about the application's external synchronization, like mutexes or condition variables, is not available. Therefore, during a replay, the captured commands are read from file and performed sequentially on a single thread. While this produces a faithful reconstruction of the application's graphics API usage, it can increase the run time in the case where the original application utilized many CPU cores.

This is especially the case for pipeline creation, which involves compiling SPIR-V shader code - a very cpu intensive task. For a typical AAA title, compiling and caching many thousands of pipelines is typical.

Solution

While not completely solving the underlying problems of a true multithreaded replay, we have implemented a spot solution which uses multithreading for pipeline and shader creation.

The solution was implemented by creating a job queue backed by a threadpool. Calls to **`vkCreateGraphicsPipelines`**, **`vkCreateComputePipelines`**, and **`vkCreateShadersEXT`** are performed asynchronously in the background - while replay continues on the main-thread. Synchronization with compilation tasks is performed when a pipeline is first requested during replay.

The new behavior can be requested by using the **`--pipeline-creation-jobs`** (or **`--pcj`**) command line option to the **`gfxrecon-replay`** command.

Result

The performance increase can be substantial, but depends on the scenario. In general, shader compilation time will decrease linearly with the number of used CPU cores. Applications compiling many pipelines on a CPU with many cores will benefit most.

One AAA game developer reported the following performance increase:

Before: `gfxrecon-replay <game>.gfxr`

➔	4556.06s user	Time spent in the application (user mode)
➔	9.88s system	Time spent executing system calls (kernel mode)
➔	99% cpu	CPU utilization (1 core)
➔	**1:16:45.06 total**	**1 Hour, 16 Minutes, 45 Seconds**

After: `gfxrecon-replay <game>.gfxr --pipeline-creation-jobs 32`

➔	6532.74s user	Time spent in the application (user mode)
➔	14.97s system	Time spent executing system calls (kernel mode)
➔	3057% cpu	CPU utilization (many cores)
➔	**3:34.16 total**	**3 Minutes, 34 Seconds**

Conclusion

Anyone replaying captures of realistic engine/game scenarios can potentially achieve significant performance gains during the first replay - when shaders are compiled - by using the new command line option to the replay command which takes advantage of multiple CPU cores to compile shaders in parallel.