D3D12 Command Throttle Policy Extension Q/A

Adam Lake, Jiawei Shao

3/18/22

Contents

[Audience 2](#_Toc130027834)

[What does this extension do? 2](#_Toc130027835)

[What this extension does not do? 2](#_Toc130027836)

[Won’t this drain my battery faster? 2](#_Toc130027837)

[Should I use this for all of my compute work? 2](#_Toc130027838)

[Who would want to use this extension? 2](#_Toc130027839)

[What is the support required by release of this extension? 2](#_Toc130027840)

[Is this on both integrated and discrete? 3](#_Toc130027841)

[How does this interact with asynchronous compute? 3](#_Toc130027842)

[How to use the API 3](#_Toc130027843)

[Sample Application 4](#_Toc130027844)

[Creating the Compute Command Queue 4](#_Toc130027845)

[Building and Running the sample 4](#_Toc130027846)

[Acknowledgements 5](#_Toc130027847)

# Audience

Anyone interested in finding out more about the DX12 Command Throttle Policy Extension capability.

# What does this extension do?

This extension is a *hint* to the driver that allows work scheduled by an application submitted in a DX12 command queue to be scheduled at a higher *maximum* clock frequency. This maximum clock frequency hint is a system dependent time on the order of a few seconds. After the work is completed, the driver returns to the default DX12 *dynamic* power setting. This hint can be ignored by the driver depending on the circumstances, for example when running on battery or a low power scenario.

# What this extension does not do?

The extension does not permanently change the power in the platform. It is not even a new feature. Other APIs and workloads in the platform have this scheduling model by default for work scheduled in different command queues. This just empowers a DX12 application with the same capability in constrained scenarios where the application vendor has tested to ensure they see a measurable benefit.

# Won’t this drain my battery faster?

Engaging this extension when under battery power would likely have a small impact on overall reduction in battery life. However, the application can decide to not engage the extension when in a battery powered scenario.

# Should I use this for everything?

No. One possible use case is when you have a known, long running, and dedicated GPU task and the CPU is waiting on the result. Peak frequency on the GPU can mean tradeoffs in CPU performance. Something as complex as a game engine with many fine-grained tasks to execute per frame is concurrently executing work on the CPU and GPU and the system is optimally managing the power balance between them. Therefore, we feel the common use case is best left to the platform provider. Additionally, because the time quanta that peak frequency is maintained is at the granularity of multiple seconds, it is not suitable for fine grained task scheduling.

# Who would want to use this extension?

An application that wants to ensure that the work scheduled in a compute queue gets a small burst of performance. One example might be a machine learning package that uses the GPU to do a large amount of compute **that the CPU is waiting on** to compute a result.

## What is the support required by release of this extension?

This extension will be supported as part of our driver release. It is required that you use the extension framework linked into your application as a static library. For more details about the extensions in general see the sample located in the samples directory: <https://github.com/GameTechDev/D3DExtensions_public>. For specifics about this extension read the section ‘How to use the API’ in this document.

## Is this on both integrated and discrete?

This feature is implemented for integrated and discrete with the same behavior on both platforms. The specific time quanta may change and it is under the platforms discretion to honor the hint or not.

# How does this interact with asynchronous compute?

The high frequency burst applies to any work active when the hardware receives the signal. Thus, all the work will run until the time interval expires. The time interval is system dependent but expected to be on the order of a few seconds.

# How to use the API

The programmer creates a command queue with the Intel Extension Framework and passes in a flag that requests a command queue with this hint set. From then on, work scheduled on this command queue will set the flag in the KMD, execute the work, and lower the clock frequency after the time quanta has expired. We introduce a new data structure, INTC\_D3D12\_COMMAND\_QUEUE\_THROTTLE\_POLICY, and pass this into the INTC\_DX12\_CreateCommandQueue() API. These new structures are shown here from the extension header file:

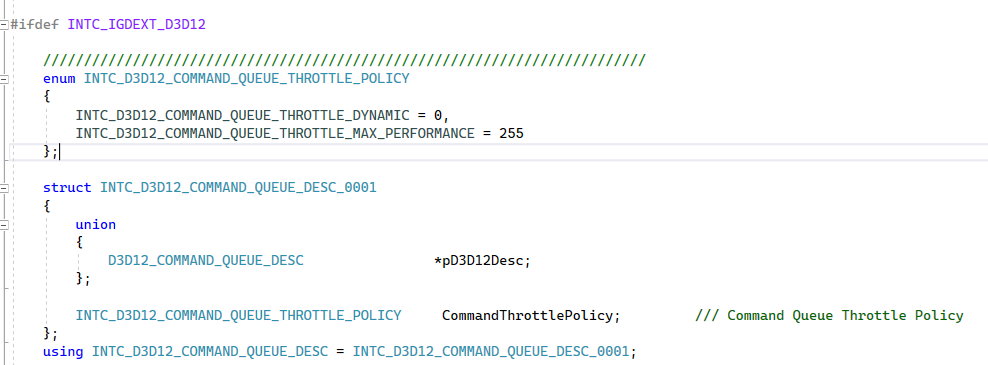


Figure . New data structure required to support this extension.

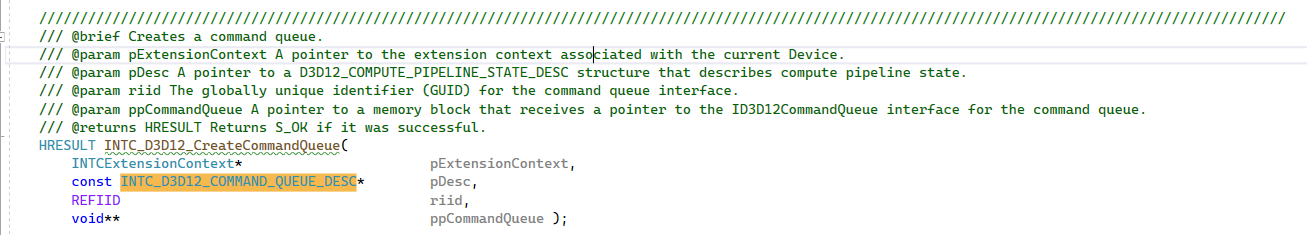


Figure . The extension API to create the command queue that takes as input the INTC\_D3D12\_CREATE\_COMMAND\_QUEUE\_DESC from above.

# Sample Application

To demonstrate the use of this extension we have developed a simple DirectX12 sample that performs a matrix multiplication. The user can pass a flag in to disable the extension and verify that it either engaged or did not based on the time required to compute the result.

## Creating the Command Queue

Setup and teardown the extension framework is covered in a separate document and most of this work is in the function InitIntelExtensions(). After the framework has been initialized the only additional requirement that is different than creating a normal command queue is to add the INTC\_D3D12\_COMMAND\_QUEUE\_DESC and specify the INTC\_D3D12\_COMMAND\_QUEUE\_THROTTLE\_MAX\_PERFORMANCE flag then call INTC\_D3D12\_CreateCommandQueue() as shown here:



Figure 3. Code sequence to create the command queue with the command throttle policy extension hint. After the queue is created, use this queue as you would any other queue.

## Building and running the sample

To build the sample application, open it in Microsoft\* Visual Studio. This was tested in Visual Studio 2022 on an Intel® A770 Discrete GPU. After building the application, you can run it from the command line or directly from the IDE. The following flags are supported:

|  |  |
| --- | --- |
| **Flag** | **Behavior** |
| --disable-command-throttle-policy-extension | Disables the throttling behavior. Useful to verify that the default behavior of the application which uses the extension either worked or not. |
| --check-gpu-result | Do matrix multiplication on the CPU. Useful to verify the matrix multiplication produced a ‘correct’ result. |
| --h | Prints command line parameters |

To verify that the extension has been engaged, run the application without any flags then run it again and pass the –disable-command-throttle-policy-extension flag. The time printed to the screen will allow you to determine if the extension was engaged or not.

# Acknowledgements

Thanks to Marc Beuchat, Jiawei Shao, and Adam Kunka for investigations, analysis, and implementation of this extension.