Performance State Tracking 37

Aleksandar Dimitrijević

37.1 Introduction

Reducing power consumption and dissipation is one of the predominant goals of all modern integrated circuit designs. Besides the design-time optimizations, all CPU/GPU vendors implement various real-time methods to reduce power consumption while preserving acceptable performance. One of the consequences of power management is a dynamic change in working frequencies and, hence, the overall performance capabilities of the system. Modern GPUs, for both desktop and mobile platforms, can be very aggressive in changing working frequencies according to the current load.

Consider a simple case of rendering a triangle on a system with an NVIDIA GeForce GTX 470 graphics card. NVIDIA drivers raise the frequencies to the highest level instantly if they detect a 3D application. Even a creation of the OpenGL rendering context is enough to make the GPU enter the highest performance state. The moment the application starts, the GPU frequency is 607.5 MHz, while the memory IO bus frequency is 1674 MHz. A frame rendering time is less than 0.16 ms for the full HD MSAA 8x screen and the GPU utilization is about 0%. After a dozen seconds, since the utilization is extremely low, the GPU enters a lower performance state. The frame rendering time is changed to about 0.24 ms. Since the GPU remains at low utilization, the performance is further reduced. After changing four performance levels, the GPU finally enters the lowest performance state with the GPU frequency at 50.5 MHz, and memory IO bus frequency at 101 MHz. The rendering capabilities are reduced by an order of magnitude, while the frame rendering time rises up to 1.87 ms. If we do not track the performance state, we are not able to

© 2012 by Taylor & Francis Group, LLC

interpret measured results correctly. Furthermore, for less demanding applications, it is possible to get shorter execution time on some older and less powerful graphics cards because their lower performance states may involve much higher frequencies.

37.2 Power Consumption Policies

For many years, graphics card vendors have been developing a highly advanced form of dynamic power management (DPM). DPM estimates the relative workload and aggressively conserves power when the workload is low. Power consumption is controlled by changing voltage levels, GPU frequencies, and memory-clock frequencies. A set of values that define the current power consumption and performance capabilities of the graphics card is known as a *performance state* (P-state).

NVIDIA defines sixteen P-states, where P0 is the highest P-state, and P15 is the idle state. Not all P-states are present on a given system. The state P0 is activated whenever a 3D application is detected. If the utilization is below some threshold for a certain period of time, the P-state is changed to a lower level.

AMD defines three P-states, where P0 is the lowest, and P2 is the highest performance state. P0 is the starting state, and it is changed only by demanding applications. The latest AMD technology, known as PowerTune [AMD 10], defines a whole range of working frequencies in the highest P-state. When the GPU reaches the thermal design power (TDP) limits, the GPU frequency is gradually decreased while maintaining the high power state. This enables much better performance for demanding applications, while preserving acceptable power level.

Having in mind such advanced power-management scenarios, a fair comparison of different rendering algorithms cannot be done on a frame-rate basis only. If the same or an even lower frame-rate is achieved in the lower P-state, it certainly qualifies the algorithm as more efficient, or at least less demanding. That is why P-state tracking is an important part of profiling software. So far, OpenGL doesn't have a capability to track P-states; thus, we will take a look at how it can be implemented using vendor-specific APIs: NVAPI for NVIDIA and ADL for AMD hardware.

37.3 P-State Tracking Using NVAPI

NVAPI is NVIDIA's core API that allows direct access to NVIDIA drivers on all Microsoft Windows platforms [NVIDIA 11c], and it is shipped as a DLL with the drivers.[1] NVAPI has to be statically linked to an application; hence, a software development kit (SDK) has been released with appropriate static library and

[1]The official documentation states that NVAPI is supported by drivers since Release 81.20 (R81.20), but there are problems in accessing most of its functionality through the SDK in pre-R195 drivers. The first NVAPI SDK was released with R195 in October 2009. Since R256 drivers, all settings have become wide open to change with NVAPI.

© 2012 by Taylor & Francis Group, LLC

header files.[2] It is both forward and backward compatible. Calling a function that is not implemented in the current version of the DLL only returns a `NVAPI_NO_IMPLEMENTATION` error and does not result in an application crash [NVIDIA 11a].

When reading or changing some driver parameters, communication with the drivers is session-based [NVIDIA 11a]. An application has to create a session, load system settings into the session, get a profile, and finally read or write some setting to the profile. All driver settings are organized into profiles. Some profiles are shipped with the driver as Predefined Profiles, while others are created by the user as User Profiles. A profile can have an arbitrary number of applications associated to it, while each application can be associated with only a single profile. A profile associated with applications is called an Application Profile. If the application is not associated with a certain Application Profile, or the specific setting is not defined in the associated profile, the Current Global Profile is used. All profiles that are not associated with applications are called Global Profiles. Only one of the Global Profiles can be selected as the Current Global Profile. If the setting is not found in the Current Global Profile, the Base Profile is used. The Base Profile is system wide, and its settings are automatically applied as defaults.

The driver's settings are loaded and applied at the moment when the driver DLL is initialized. If the settings are changed during the application execution, the application has to be restarted in order to take advantage of new settings. Considering OpenGL, NVAPI offers the ability to configure OpenGL Expert Mode, its feedback and reporting mechanism.

P-states tracking does not require reading or changing any driver settings, which eliminates the need for sessions and profile manipulation. The only required step is to initialize NVAPI through `the NvAPI_Initialize` function call.

37.3.1 GPU Utilization

The GPU utilization directly affects the current P-state. High utilization activates a higher P-state if the GPU is not in the highest state already. The function that can be used to retrieve utilization is `NvAPI_GPU_GetDynamicPstatesInfoEx`. Unlike its name would suggest, the function does not tell which P-state the GPU is currently in, but rather information about the utilization in the current state. The utilization is defined over three GPU domains:

- graphic engine (GPU),
- framebuffer (FB), and
- video engine (VID).

The official documentation [NVIDIA 11c] states that P-state thresholds can also be retrieved, although this functionality is not yet exposed through the SDK.

[2] http://developer.nvidia.com/nvapi

© 2012 by Taylor & Francis Group, LLC

```
#define UTIL_DOMAIN_GPU 0
#define UTIL_DOMAIN_FB 1
#define UTIL_DOMAIN_VID 2

NvPhysicalGpuHandle m_hPhysicalGPU[NVAPI_MAX_PHYSICAL_GPUS];
NvU32 m_gpuCount;
NV_GPU_DYNAMIC_PSTATES_INFO_EX m_DynamicPStateInfo;
NvAPI_EnumPhysicalGPUs(m_hPhysicalGPU, &m_gpuCount);
m_DynamicPStateInfo.version = NV_GPU_DYNAMIC_PSTATES_INFO_EX_VER;
NvU32 utilGPU, utilFB, utilVID;

NvAPI_Status status = NvAPI_GPU_GetDynamicPstatesInfoEx(m_hPhysicalGPU[0], &↩
    m_DynamicPStateInfo);

if(status == NVAPI_OK ){
    utilGPU = m_DynamicPStateInfo.utilization[UTIL_DOMAIN_GPU].percentage;
    utilFB  = m_DynamicPStateInfo.utilization[UTIL_DOMAIN_FB].percentage;
    utilVID = m_DynamicPStateInfo.utilization[UTIL_DOMAIN_VID].percentage;
}
```

Listing 37.1. Reading GPU utilization using NVAPI.

Since the system can have multiple GPUs, `NvAPI_GPU_GetDynamicPstatesInfoEx` requires a handle to a physical GPU for which utilization should be retrieved. The total number of physical GPUs and the handles to them can be retrieved with `NvAPI_EnumPhysicalGPUs`. Listing 37.1 illustrates reading GPU utilization using NVAPI. The utilization is read for the default graphics adapter (`m_hPhysicalGPU[0]`).

All NVAPI functions require setting a proper value for the `version` field of the structures passed to them as parameters. For the structures defined in the SDK, the proper values of the *version* fields are defined as `<structure_name>_VER`.

The retrieved utilization value is not a single clock-interval sample, but an averaged value over the last second. If the GPU utilization drops below a certain threshold, depending on the mutual relationship of two adjacent P-states, the next lower state is activated. Experimental results show that the lower threshold varies from 5% to 30%. If GPU utilization crosses the upper threshold (usually about 60%), the next higher level is activated. The threshold values may vary depending on the hardware, P-state settings, and the driver's policy. The transition to a lower P-state requires having a low utilization for about 15 s for pre-R280 drivers on Windows, while the transition to a higher P-state is instantaneous. Starting from R280, NVIDIA decreases the amount of time required at low GPU utilization before transitioning to a lower P-state [NVIDIA 11c].

37.3.2 Reading P-States

All available P-states of a physical GPU, with accompanying parameters, can be retrieved using the `NvAPI_GPU_GetPstatesInfoEx` function, which has three parameters:

- a handle to a physical GPU,

© 2012 by Taylor & Francis Group, LLC

- a pointer to an NV_GPU_PERF_PSTATES_INFO structure, and
- input flags.

Input flags are allocated to define various options, but currently, only bit 0 is used to select whether the returned values are the defaults, or the current settings. The current settings might differ from the defaults if the GPU is overclocked.

If a call to NvAPI_GPU_GetPstatesInfoEx succeeds, an NV_GPU_PERF_PSTATES_INFO structure is filled with parameters regarding P-states. In the version of NVAPI shipped with R290 drivers, NV_GPU_PERF_PSTATES_INFO contains the following fields:

- version. Version of NV_GPU_PERF_PSTATES_INFO; it should be set to NV_GPU_PERF_PSTATES_INFO_VER before calling the function.
- flags. Reserved for future use.
- numPstates. Number of available P-states.
- numClocks. Number of domains for which clocks are defined; there are currently three public clock domains: NVAPI_GPU_PUBLIC_CLOCK_GRAPHICS, NVAPI_GPU_PUBLIC_CLOCK_MEMORY, and NVAPI_GPU_PUBLIC_CLOCK_PROCESSOR.
- numVoltages. Number of domains for which voltages are defined; currently, only one domain is presented: NVAPI_GPU_PERF_VOLTAGE_INFO_DOMAIN_CORE.
- pstates[16]. Parameters of each P-state:
  - pstateId. ID of the P-state (0...15),
  - flags:
    - bit 0. PCI-e limit (version 1 or 2),
    - bit 1. P-state is overclocked,
    - bit 2. P-state can be overclocked,
    - bits 3–31. Reserved for future use.
- clocks[32]:
  - domainId. Domain for which the particular clock is defined: NVAPI_GPU_PUBLIC_CLOCK_GRAPHICS, NVAPI_GPU_PUBLIC_CLOCK_MEMORY, or NVAPI_GPU_PUBLIC_CLOCK_PROCESSOR,
  - flags:
    - bit 0. Clock domain can be overclocked,
    - bits 1–31. Reserved for future use,

© 2012 by Taylor & Francis Group, LLC

- freq. Clock frequency in kHz.

- voltages[16]:
  - domainId. Domain for which the voltage is defined (NVAPI_GPU_PERF_VOLTAGE_INFO_DOMAIN_CORE),
  - flags. Reserved for future use,
  - mvolt. Voltage in mV.

The current P-state ID of a physical GPU is retrieved using NvAPI_GPU_GetCurrentPstate. Unlike NvAPI_GPU_GetPstatesInfoEx, which should be called only once, usually during the application initialization; NvAPI_GPU_GetCurrentPstate must be called frequently, at least once per frame.

Unfortunately, the NVAPI SDK does not expose all the functionality of NVAPI. Most of the functionality is available only in the NDA version of the SDK. Furthermore, the official documentation refers to some functions and structures that are not included even in NDA version. We hope that more functions will be published in the future.

37.4 P-State Tracking Using ADL

AMD Display Library (ADL) is an API that allows access to various graphics driver settings. It is a wrapper around a private API for both Windows and Linux. ADL binaries are delivered as a part of the Catalyst display driver package for Windows or Linux, while the SDK (documentation, definitions, and sample code) can be downloaded from the website.[3]

P-state management is exposed openly through the ADL OverDrive Version 5 (OD5) API. OD5 provides access to the engine clock, memory clock, and core voltage level. Each state component can be read and also changed within some predefined range. The range is defined inside the BIOS of the graphics card and prevents hardware malfunctioning.

P-states have to be enumerated in an ascending order, with each component having the same or a greater value compared to the previous state. If the rule is violated, the state will not be set. Custom settings are not preserved after a system restart, and should be maintained by the application. Since the standard P-state settings are determined through a comprehensive qualification process, it is not recommended to change them. We will confine our interaction with OD5 only to P-state tracking. Listing 37.2 illustrates how the current P-state setting can be read.

All relevant parameters of the current P-state can be retrieved using the ADL_Overdrive5_CurrentActivity_Get function call. The values are stored in an

[3] http://developer.amd.com/sdks/ADLSDK/Pages/default.aspx

© 2012 by Taylor & Francis Group, LLC

```
typedef int (*ADL_OVERDRIVE5_CURRENTACTIVITY_GET) (int, ADLPMActivity *);

HINSTANCE hDLL = LoadLibrary(_T("atiadlxx.dll")); // try to load native DLL
if(hDLL == NULL){       // if fails (32-bit app on 64-bit OS), load 32-bit DLL
    hDLL = LoadLibrary(_T("atiadlxy.dll"));
}

ADL_Overdrive5_CurrentActivity_Get = (ADL_OVERDRIVE5_CURRENTACTIVITY_GET)↩
    GetProcAddress( hDLL, "ADL_Overdrive5_CurrentActivity_Get");
ADLPMActivity activity;

activity.iSize = sizeof(ADLPMActivity);
ADL_Overdrive5_CurrentActivity_Get (0, &activity);
```

Listing 37.2. Retrieving the current P-state parameters using ADL OverDrive5.

`ADLPMActivity` structure, which, among others, contains the following P-state members:

- `iCurrentPerformanceLevel`. Current P-state ID,
- `iEngineClock`. GPU engine clock in tens of kHz,
- `iMemoryClock`. Memory clock in tens of kHz,
- `iVddc`. Core voltage level in mV, and
- `iActivityPercent`. GPU utilization in %.

The greatest advantage of the OD5 P-state tracking is its simplicity. All parameters are retrieved with a single function call. Each P-state is uniquely identified by `iCurrentPerformanceLevel`, where a higher value corresponds to a higher P-state. The first parameter of the `ADL_Overdrive5_CurrentActivity_Get` is the adapter index. The previous example assumes the default adapter; hence, the value is 0.

37.5 Conclusion

P-states tracking is essential for all profilers. Each measured value should be stamped with the current state in order to be properly interpreted and filter out unwanted values. Special care must be taken during the interpretation, since the state change might be recorded with one-frame delay. So, not only the current state is important, but also the frame in which a transition occurs. Another problem can arise if the state transition happens during the measured interval that spans multiple frames. If we measure frequent or periodical events, the intervals that enclose state changes could be just ignored. In the case of events that are not periodic, we should subdivide the measured interval in order to catch the frame in which the transition occurs.

© 2012 by Taylor & Francis Group, LLC

The performance state policies differ widely according to graphics hardware, current drivers, and vendors' preferences. NVIDIA aggressively raises the P-state to the highest value as soon as a 3D application is detected. If the application is less demanding, the P-state is gradually decreased. AMD has a different policy and starts with the lowest performance state. In the case of both vendors, the GPU utilization is tracked precisely, and the P-state is changed accordingly.

Thus far, OpenGL does not have the ability to track P-states. In order to reach all relevant parameters, like working frequencies or GPU utilization, we have to use vendor-specific APIs. NVAPI gathers detailed information, but most of its functionality is still hidden, while ADL openly provides an easy-to-use interface to the required information, and, furthermore, enables customization of P-states parameters. Since power consumption becomes more and more important with each new generation of graphics cards, we can expect a further development of P-state-accessing APIs. Perhaps even OpenGL will provide an insight into what is really happening beneath the powerful graphics cards' coolers.

Bibliography

[AMD 10] AMD. "AMD PowerTune Technology." http://www.amd.com/us/Documents/PowerTune_Technology_Whitepaper.pdf, December 2010.

[NVIDIA 11a] NVIDIA. "NVIDIA Driver Settings Programming Guide." PG-5116-001-v02, http://developer.download.nvidia.com/assets/tools/docs/PG-5116-001_v02_public.pdf, January 19, 2011.

[NVIDIA 11b] NVIDIA. "Understanding Dynamic GPU Performance Mode, Release 280 Graphics Drivers for Windows—Version 280.26." RN-W28026-01v02, http://us.download.nvidia.com/Windows/280.26/280.26-Win7-WinVista-Desktop-Release-Notes.pdf, August 9, 2011.

[NVIDIA 11c] NVIDIA. "NVAPI Reference Documentation (Developer)," Release 285, http://developer.nvidia.com/nvapi, September 15, 2011.

© 2012 by Taylor & Francis Group, LLC