Reverse engineering power management on NVIDIA GPUs - A detailed overview

Martin Peres
Ph.D. student at LaBRI
University of Bordeaux
Hobbyist Linux/Nouveau Developer
Email: martin.peres@labri.fr

***Abstract*—Power management in Open Source operating systems is currently limited by the fact that hardware companies do not release enough documentation to write the most power-efficient drivers. This problem is even more present in GPUs despite having the highest performance-per-Watt ratio found in today's processors. This paper presents an overview of GPUs from a power management point of view and what power management features have been found when reverse engineering modern NVIDIA GPUs. This paper finally discusses about the possibility of achieving self-management on NVIDIA GPUs and also discusses the future challenges that will be faced by the community when autonomic systems like Broadcom's videocore®will become mainstream.**

I. Introduction

Historically, CPU and GPU manufacturers were aiming to increase performance as it was the usual metric used by consumers to select what to buy. However with the rise of laptops and smartphones, consumers have started to prefer higher battery-life, slimmer, quieter, and cooler devices. Noise was also a concern among some desktop users who started using water cooling in place of fans. This led CPU/GPU manufacturers to not only take performance into account, but also performance-per-Watt. Higher performance-per-Watt means lower heat dissipation for the same amount of computing power, which in turn allows shrinking the heatsink/fan that keep the processor cool. This results in slimmer and/or quiter devices. Decreasing power usage is also a major concern for datacenters and supercomputers, as they already consumed 1.5% of the USA's electricity production in 2007 [1].

As power management (PM) is a non-functional/auxiliary feature, it is usually non-standard, poorly documented and/or kept secret. As some PM features require the intervention of the host system, a driver is often needed to obtain the best performance-per-Watt. This is not a problem for closed-source OSes such as Microsoft Windows®or Mac OS®as the manufacturer usually supplies a proprietary driver for these platforms. It is however a major problem for Open Source operating systems like Linux as those features are not documented and cannot be re-implemented easily in an Open Source manner.

The absence of documentation and the lack of open hardware also sets back research and open source enthusiasts as it mandates reverse engineering the undocumented features which may not even be advertised. This lack of documentation is most present in the GPU world, especially with the company called NVIDIA which has been publicly shamed for that reason by Linus Torvalds [2], creator of Linux. With the exception of the ARM-based Tegra, NVIDIA has never released enough documentation to provide 3D acceleration to a single GPU after the Geforce 4 (2002). Power management was never supported nor documented by NVIDIA. Reverse engineering the power management features of NVIDIA GPUs would allow to improve the efficiency of the Linux community driver for NVIDIA GPUs called Nouveau [3]. This work will allow driving the hardware more efficiently which should reduce the temperature, lower the fan noise and improve the battery-life of laptops.

Nouveau is a fork of NVIDIA's limited Open Source driver, xf86-video-nv [4] by Stephane Marchesin aimed at delivering Open Source 3D acceleration along with a port to the new graphics Open Source architecture DRI [5]. As almost no documentation from NVIDIA is available, this driver mostly relies on reverse engineering to produce enough knowledge and documentation to implement the driver. The Nouveau driver was merged into Linux 2.6.33, released in 2009, even though 3D acceleration for most cards has been considered experimental until 2012. I personally joined the team in 2010 after publishing [6] a very experimental implementation of power management features such as temperature reading and reclocking, which are essential to perform Dynamic Voltage/Frequency Scaling (DVFS). Since then, power management is being investigated and implemented in the driver whenever a feature is well understood.

This paper is an attempt at documenting some of the features that have been reverse engineered by the Nouveau team and comparing them to the state of the art. Some of the documented features have also been prototyped and implemented in Nouveau. A brief evaluation of the GPU power consumption has also been carried out.

Section II presents an overview of the power-related functionalities needed to make the GPU work. Performance counters and their usage is detailed in section III. In section IV, we document the hardware fail-safe feature to lower temperature and power consumption. Section V introduces NVIDIA's RTOS embedded in the GPU. Finally, section VI presents the vision of autonomic computing, the challenges in introduces in Open Source driver development and how NVIDIA GPUs could fit to this model.

II. General overview of a modern GPU

This section is meant to provide the reader with information that is important to understand the following sections.

A. General-Purpose Input/Output (GPIO)

A General-Purpose Input/Output (GPIO) is a pin of a chip that can be software-selected as a binary input or output at run time.

Input GPIO pins are used by NVIDIA to sense whether the external power supply or the SLI brige is connected, if the GPU is overheating (in the case of an external temperature sensor) or to read the fan's rotational speed. Output GPIO pins are used by NVIDIA to drive a laptop's backlight, select the memory and core voltages or adjust the fan speed.

Some GPIO pins can be connected to special features like hardware pulse-width modulation (PWM) controllers which allow, for instance, varying the amount of power sent to the fan or the backlight of a laptop's screen. Some chips also have a hardware tachometer which calculates the spin rate of the fan. All of these operations could be done in software but they would require the CPU to wake up at least dozens of times per second, issue a read/write request through the PCIe [7] port and then go back to sleep again. Waking up the CPU for such trivial operations is inefficient and should thus be avoided.

B. Energy sources

A modern desktop GPU draws its power from the PCIe port or PCIe connectors (6 or 8 pins). The PCIe port and 6-pin PCIe connectors can each source up to 75W while the 8-pin PCIe connector can source up to 150W.

These power sources all provide different voltages that are way higher than the operating voltage of the GPU. The DC-DC voltage conversion is done by the voltage controller which sources from the PCIe port and connectors and outputs power to the GPU at its operating voltage. The output voltage is software-controllable using a GPIO pin. A GPIO pin is also usually used by cards with PCIe connectors to sense if they are connected to a power source. This allows NVIDIA's driver to disable hardware acceleration when the connectors are not connected, avoiding the GPU from draining too much current from the PCIe port. In some cases, a GPIO pin can also control the emergency shutdown of the voltage controller. It is used by the GPU to shutdown its power when the card overheats.

There are currently only up to two power domains on NVIDIA cards, one for the GPU and one for memory. The memory's power domain is limited to 2 possible voltages while the GPU's usually has at least 3 and can have up to 32 voltage possibilities.

On some high-end cards, the voltage controller can also be queried and configured through an I^2C interface. Those expensive controllers can sense the output power along with the power draw of each of the energy sources.

Figure 1 illustrates a simple view of the power subsystem.

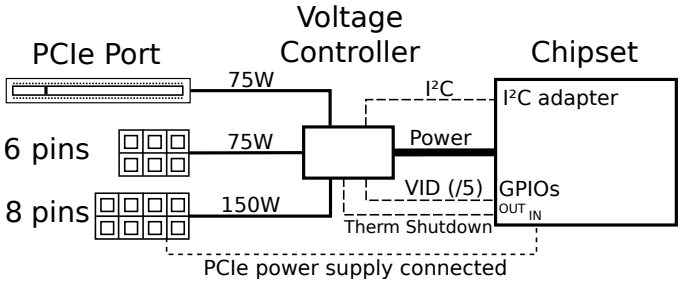

Figure 1. Overview of the energy sources of the GPU

C. Temperature management

Just like modern CPUs, GPUs are prone to overheating. In order to regulate their temperature, a temperature sensor needs to be installed. This sensor could be internal to the chip or be external using an I^2C [8] sensor. Usually, external sensors can also drive a fan's speed according to the card's temperature. When using the internal sensor, the driver is usually responsible for polling the temperature and updating the fan speed.

D. Clock tree

Having the possibility to change frequencies on the fly is another feature of modern GPUs aimed at lowering power consumption. On NVIDIA GPUs, there are two clock sources, the PCIe reference clock (100 MHz) and an on-board crystal (usually 27 MHz). The frequency of the clocks is then increased by using a Phase-locked loop (PLL). A PLL takes the input frequency F_{in} and outputs the frequency F_{out}. The relation between F_{in} and F_{out} in the simplest PLL is detailed in equ. 1. The parameters N and M are integers and have a limited range. This range depends on the PLL and is usually documented in the Video BIOS (VBIOS). This means that not all output frequencies are achievable.

$$F_{out} = F_{in} * \frac{N}{M} \tag{1}$$

There are different kinds of PLLs inside a single GPU and across GPU generations. For instance, some PLLs may actually have 2 stages and a power-of-two post divider like shown by equ. 2. This second stage or the power-of-two divider could also be set in bypass mode.

$$F_{out} = F_{in} * \frac{\frac{N}{M} * \frac{N2}{M2}}{2^p} \tag{2}$$

The PLLs are not the only components that can alter the clock frequency. There may also be clock dividers. Those dividers can only lower the frequency of the input clock. There are usually two kinds of dividers used by NVIDIA, the power-of-two dividers and the integer dividers.

The power-of-two dividers are only able to lower the clock frequency by a power-of-two factor as detailed by equ. 3. They are easily constructed by chaining flip-flops and routing the output of each flip-flop to a multiplexer. The multiplexer is then used to select the wanted division factor by selecting which flip-fop's output should be used as an output for the

divider. Integer dividers are more expensive than power-of-two dividers, but they can divide a clock by an integer as shown by equ. 4.

$$F_{out} = \frac{F_{in}}{2^p} \tag{3}$$

$$F_{out} = \frac{F_{in}}{n} \tag{4}$$

Another common component involved in the clock tree is the multiplexer. A multiplexer takes as an input several clocks and only outputs one of them. The selection of the output clock is controllable electronically and is exposed as a register for the driver. Figure 2 is an example of a 4-inputs multiplexer with 2 select-lines to be able to address every input.

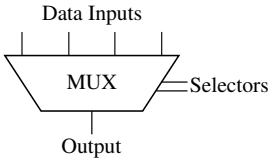

Figure 2. Example of a 4-data-input multiplexer

It would be simplistic to think that the GPU uses a single clock for the whole card. Indeed, a modern GPU has multiple engines running asynchronously which are clocked by different clock domains. There are up to 13 clock domains on Fermi [9] among which we can find the host, video RAM (VRAM), shader, copy (asynchronous copy engine) and ROP domains. The number of clock domains varies depending on the chipset and the chipset family.

Figure 3 gives a rough overview of how PGRAPH's clock can be generated. This part of the clock tree involves the two input clocks along with many PLLs and multiplexers. Many of these PLLs are actually shared and/or could be used to clock several engines. In this example, NVCLK could also be set to use the DOM6 clock. How this clock is generated is not detailed in this paper but it is equivalent to the way nvclk is.

NVIDIA GPUs' clock trees have been reverse engineered by using the performance counters (detailed in section III) which can count every clock cycle of most clock domains. The clocks can be read through nvatiming, a tool from the envytools repository [10]. This tool should be run before and after modifying the content of a register that is suspected to configure the clock tree. As a start, it is possible to check all the registers NVIDIA's proprietary driver reads from or writes to when reclocking the GPU. These reads/writes can be logged by tracing the Memory-Mapped IO (MMIO) accesses of the proprietary driver using the MMIO-trace feature of the Linux kernel [11].

E. The role of the video BIOS

On an IBM-PC-compatible computer, the BIOS is responsible for the Power-On Self Test (POST). During the POST phase, the BIOS performs some sanity checks and then initialises peripherals by giving them addresses in the linear address space and running their internal BIOS when applicable.

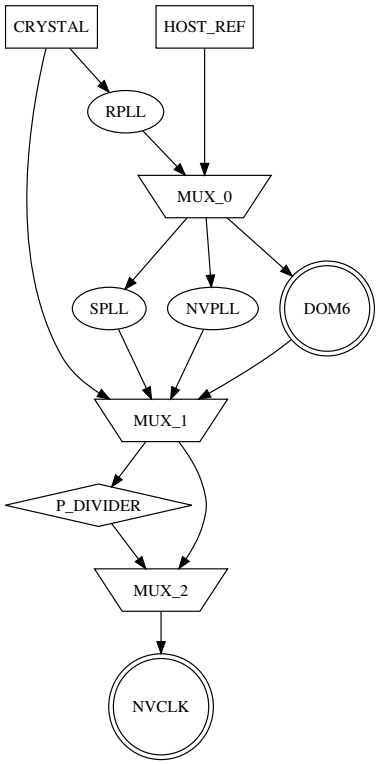

Figure 3. Overview of the clock tree for nvclk (core clock) on an nv84

Similarly, in the case of the video card, the video BIOS (VBIOS) is responsible for booting up the card enough to provide the VGA/VESA interface to the host. This interface is then used by the BIOS and the bootloader to display boot information on the screen.

In order to bring up this interface, the VBIOS configures the GPU's clock tree, configures the memory controllers, uploads default microcodes to some engines, allocates a framebuffer to store the content of the screen and finally performs graphic mode setting. The VBIOS is then called by the BIOS when an application on the host requests anything such as changing the resolution. The VBIOS is written in 16-bit x86 code and should be considered as being a low-level simple driver.

As NVIDIA does not select the VRAM chips, the voltage controller, where the GPIO pins are connected or what graphics connectors (HDMI, DVI, VGA, DP, etc...) are available, the card manufacturers need a way to tell the VBIOS how to set up the card at boot time. This is why NVIDIA stores the manufacturer-dependent information in tables in the VBIOS. These tables can be decoded by nvbios, found in the envytools collection [10]. This collection also contains nvagetbios and nvafakebios which respectively allow to download the VBIOS or to upload a new VBIOS non-permanently, respectively.

As manufacturers are using the same BIOS for multiple cards, some of the tables are indexed by a strap register. This register's value is set by the manufacturer by tying some GPU pins to specific voltages.

1) The GPIO & external device tables: They store which devices or functions are accessible through the chip's pins

and how to access them. The EXTDEV table references I^2C-accessible devices while the GPIO table only references GPIO-accessible functions. Both also represent what is accessible by a number, called a tag or type. In the case of the GPIO table, each entry contains at least a GPIO pin, the associated tag (for instance, PANEL_PWR), the direction (input or output), the default state (HIGH or LOW) and if the GPIO is inverted or not. In the case of the EXTDEV table, each entry contains the device type/tag, which I^2C bus it is accessible on and at which address. Possible devices could be the ADT7473 which is an external temperature management unit or the PX3540 voltage controller.

2) The thermal table: Its role is to store temperature-related parameters, as defined by the card manufacturer. The temperature sensor's parameters (offset and slope) can be adjusted. It also defines hysteresis and temperature thresholds such as fan_boost, downclock and shutdown which are respectively defining the temperature at which the fan should be set to 100%, the temperature at which the card should be down-clocked and the temperature at which the computer should be shut down. Finally, the fan response to the temperature can be linear or trip-point based. The thermal table then stores the parameters for either method.

3) The performance level table: It specifies up to 4 performance levels. A performance level is defined by a set of clock speeds, a core voltage, a memory voltage and memory timings and a PCIe link width speed. The voltage is stored as an ID that needs to be looked up in the voltage-mapping table. It is by altering this table that [12] managed to force NVIDIA's proprietary driver to set the clocks the way they wanted.

4) The voltage and voltage-mapping tables: The voltage table contains {*Voltage ID*, *voltage*} tuples describing the various supported voltages and how to configure the voltage controller. Those voltages are referenced by the voltage-mapping table which defines {*ID*, *voltage_min*, *voltage_max*} tuples. The voltage_min and voltage_max parameters of this table define an acceptable voltage range for the performance level referencing the ID.

5) The RAM type, timings and timings-mapping tables: Their role is to tell the driver which memory type and timings should be set when reclocking memory. The RAM type table is indexed by the strap. This information is necessary in order to program the memory controller properly as DDR3, GDDR3 and GDDR5 do not have the same configuration registers. The timings-mapping table contains several entries, each covering a memory frequency range. The values of each entry tell how to configure the memory controllers whenever the driver wants to use a frequency from within this range. Each entry contains sub-entries which are indexed by the strap register. Each sub-entry contains the timing ID of the timings table that should be used along with some memory-specific parameters. The timings table contains several entries which are referenced by the timings-mapping table. Each entry is either the values that should be written to the memory timings registers or the aggregation of several parameters that allow the driver to calculate these values. Unfortunately, the equations to calculate the values from the parameters greatly varied in the Geforce 8 era and are not completely understood on some GPUs.

F. The reclocking process

Reclocking is the act of changing the frequencies of the clocks of the GPU. This process drastically affects the performance and the power consumption of the GPU. The relation between clocks, power consumption and performance is very hardware- and task-dependent. There is however a known relation between the voltage, the frequency of the clock and the final power consumption for CMOS circuits [13] as shown by equ. 5, 6 and 7.

$$P = P_{dynamic} + P_{static} \tag{5}$$

$$P_{static} = VI_{leak} \tag{6}$$

$$P_{dynamic} = CfV^2 \tag{7}$$

P is the final power consumption of the circuit and results from both the dynamic ($P_{dynamic}$) and the static (P_{static}) power consumption of the circuit.

The static power consumption comes from the leakage power of the transistors. This power consumption is the product of the voltage V at which the circuit operates and the leakage current I_{leak}. This leakage current is mostly influenced by the hardware design, the voltage at which the transitor is driven and the temperature. From a software point of view, the static power consumption can only be lowered by dropping the voltage or by shutting down the power (power gating) of the units of the GPU that are not in use. On NVIDIA hardware, power gating an engine can be done by clearing its associated bit in one register. Patents [14] and [15] on engine-level power gating from NVIDIA are informative about the way power gating may be implemented on their hardware.

The dynamic power consumption is influenced by the capacitance of the transistor gates C (which decreases with the transistor size), the frequency at which the circuit is clocked f and the voltage at which the circuit operates V. As C is only dependent on the way transistors where etched, it cannot be adjusted at run time to lower power consumption. Only f and V can be adjusted at run time by respectively reprogramming the clock tree's PLLs and reprogramming the voltage controller. While f has an immediate impact on performance, V has none even though it needs to be increased with f in order for the chip to be able to meet the needed switching time. Another way to decrease the dynamic power consumption is to cut the clock of the parts of the chip that are not used at the moment to compute something meaningful (clock gating). The actual implementation in NVIDIA's proprietary driver has not been reverse engineered yet but hints of how it works may be found in patents [16] and [17].

The repartition of the power consumption between the dynamic and the static power consumption is very dependent on the etching process. Static power consumption used to be negligeable [18] but it became an increasing problem when shrinking the transistors. Figure 4 illustrates the trends in static and dynamic power dissipation as foreseen by the International Technology Roadmap for Semiconductors in 2003. However,

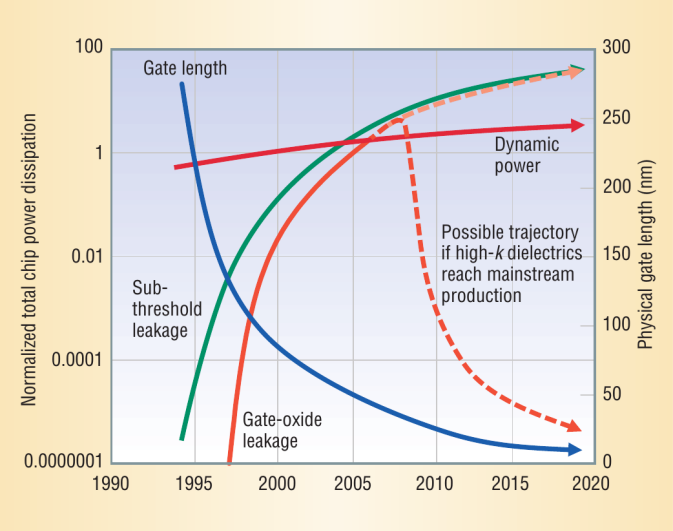

Figure 4. Total chip dynamic and static power dissipation trends based on the International Technology Roadmap for Semiconductors [18]

this figure does not take into account the recent progress in etching techniques such as Intel's tri-gate.

During the reclocking process, both the frequency and the voltage are modified in order to reach the wanted performance level. Practical tests showed that reclocking to the right performance level a Geforce GTX 480 can achieve a 28% lower power consumption while only decreasing performance by 1% for a given task [12]. The process of reclocking the GPU based on the current load of the task is called Dynamic Voltage/Frequency Scaling (DVFS).

In hardware such as NVIDIA's where clock gating is well implemented, DVFS's role is to lower the static power consumption of the circuit by lowering the operating voltage. Indeed, clock gating cuts all of the dynamic power consumption but does not influence the static power consumption. Engine-level power gating cannot cut the power consumption of an engine without loosing context which means it cannot be used when applications are using the engine. In this case, the only way to lower the power consumption is to lower the voltage at the expense of performance. If DVFS is implemented properly, the performance should still be enough for the need of the applications running on the GPU. Reading the GPU load can be performed by reading the hardware performance counters which are described in section III.

Due to the fact that GPUs are almost entirely undocumented and that the driver's interfaces are mostly closed, DVFS has been studied little in practice on GPUs, contrarily to CPUs. The only stable open-source implementation of discrete-GPU DVFS that I know of is available in the Radeon driver [19] and has been available since 2010. Some insightful comments from one of the engineers who made this happen are publicly available [20]. Reclocking on NVIDIA GPUs with Open Source software has been an on-going task since 2010 [6] and I presented my first experimental DVFS implementation at the X.org developer conference in 2011 [21]. Since then, Ben Skeggs has also managed to implement experimental DVFS support for some selected cards of the Kepler family which may be published this year.

The differences found in GPUs compared to CPUs are the multiple clock domains and the fact that not all clocks can be adjusted at any time, mainly due to the real time constraint imposed by streaming pixels to the screen. This constraint is imposed by the CRT Controller (CRTC) which reads the pixels to be displayed to the screen from the video memory (framebuffer) and streams it through the VGA/DVI/HDMI/DP link. As reclocking memory requires putting the VRAM in a self-refresh mode which is incompatible with answering memory requests, and as reclocking the PLLs takes a few microseconds, reclocking cannot be done without disturbing the output of the screen unless done during the screen's vertical blank period. This period was originally introduced to let the CRT screen move the electron beam from the bottom-right to the top-left corner of the screen. This period lasts about 400 to 500 μs and is more than enough time to reclock memory. However, on a screen refreshed 60 times per second, vblank only happens every 16.6ms. In the case where the user connects several monitors to the GPU, the vblank periods are not usually synchronised which prevents reclocking memory tearlessly on all monitors. In this case, the NVIDIA proprietary driver selects the highest performance level and deactivates DVFS.

Contrarily to memory reclocking, engine reclocking can be done at any time as long as the GPU is idle. The GPU can generally be forced to idle by disabling command-fetching and waiting for the GPU to finish processing. The engine reclocking process is not fully understood on NVIDIA cards although I found an empirical solution that managed to reclock the GPU millions of times without crashing on the Tesla chipset family. Further investigation is still needed.

The PCIe port which is used to link the CPU and the GPU also consumes some power. The PCIe standard specifies the link should be an aggregation of multiple serial lanes. Each lane has a bandwith of 250MB/s to 2GB/s depending on the PCIe version. The maximum number of lanes is 32 which means the maximum bandwidth achievable on a PCIe port is 64GB/s. Not all this bandwidth is useful at any time, yet, it consumes a lot of power as can be seen on Figure 5. To save power, it is possible to lower both the speed of individual lanes and the lanes count dynamically depending on the instanteous PCIe link load.

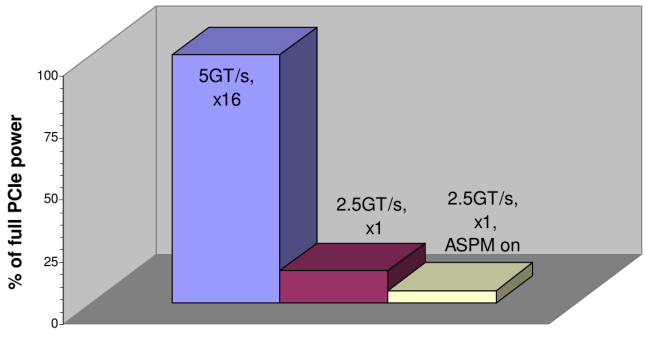

Figure 5. Maximum power consumption of the PCIe port at various link configurations [22]

Active State Power Management (ASPM) is another technique that is used to lower the power consumption of the PCIe

link by introducing two low-power states. The L0 power state only concerns one side of the link (usually the GPU) while the L1 state concerns both ends. The L1 state yields the lowest power consumption although L1's exit time is longer than L0's. This increases the PCIe link's latency in some cases. Figure 5 shows that using ASPM can divide by 3 the power consumption induced by a PCIe lane. Unfortunately, ASPM is also linked to stability problems on Linux, as explained by Matthew Garrett [23].

III. Performance counters

The (hardware) performance counters are blocks in modern microprocessors that count low-level events such as the number of branches taken or the number of cache hits/misses that happened while running a 3D or a GPGPU application. On NVIDIA's Kepler family, there are 108 different GPGPU-related monitorable events documented by NVIDIA.

A. Usage

Performance counters provide some insight into how the hardware executes its workload. They are a powerful tool to analyse the bottlenecks of a 3D or a GPGPU application. They can be accessed through NVIDIA PerfKit [24] for 3D applications or through Cupti [25] for GPGPU applications.

The performance counters can also be used by the driver in order to dynamically adjust the performance level based on the load usage of the GPU.

Some researchers also proposed to use performance counters as an indication of the power consumption with an average accuracy of 4.7% [26].

B. How does it work

The performance counter blocks are fairly well understood thanks to the work of Marcin Kościelnicki and Christoph Bumiller. There are several blocks of performance counters in modern NVIDIA GPUs, PCOUNTER, the MP counters and PDAEMON's counters. As PCOUNTER and the MP counters are very similar, we will first detail how PCOUNTER works in the Tesla family (Geforce 8) before introducing the MP counters. The PDAEMON counters will however be detailed in section V.

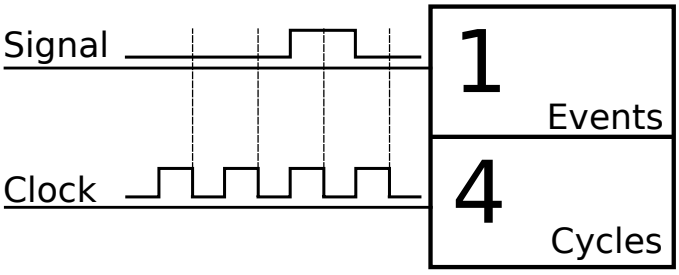

Figure 6. Example of a simple performance counter

PCOUNTER receives hardware events through internal connections encoded as a 1-bit value which we call signal. This signal is sampled by PCOUNTER at the rate of clock of the engine that generated the event. An event counter is incremented every time its corresponding signal is sampled at 1 while a cycles counter is incremented at every clock cycle. This simplistic counter is represented by figure 6.

However, it is expensive to have a counter for every possible signal. The signals are thus multiplexed. Signals are grouped into domains which are each clocked by one clock domain. There are up to 8 domains which hold 4 separate counters and up to 256 signals. Counters do not sample one signal, they sample a macro signal. A macro signal is the aggregation of 4 signals which have been combined using a function. An overview of this logic is represented by figure 7.

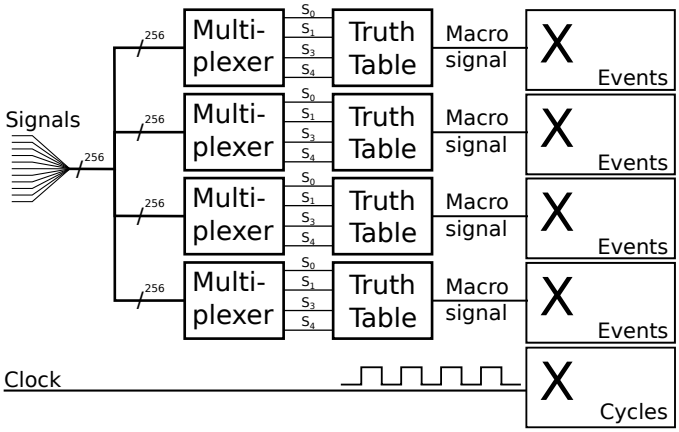

Figure 7. Schematic view of a domain from PCOUNTER

The aggregation function allows to specify which combination of the 4 signals will generate a 1 in the macro signal. The function is stored as a 16 bit number with each bit representing a combination of the signals. With $s_x(t)$ being the state of selected signal x (out of 4) at time t, the macro signal will be set to 1 if the bit $s_3(t) * 2^3 + s_2(t) * 2^2 + s_1(t) * 2^1 + s_0(t)$ of the function number is set.

As an example, to monitor signal s_0 only, the aggregation function should be programmed so as no matter the state of signals 1, 2 and 3, whenever signal 0 is high, the output of aggregation function should be high too. As can be seen in table I, the function number should have all the odd bits set and all the even bits cleared. The function number is thus $1010101010101010_{(2)}$.

Table I. Truth table to monitor s_0 only

s_3	s_2	s_1	s_0	Decimal	Selection
0	0	0	0	0	
0	0	0	1	1	X
0	0	1	0	2	
0	0	1	1	3	X
0	1	0	0	4	
0	1	0	1	5	X
0	1	1	0	6	
0	1	1	1	7	X
1	0	0	0	8	
1	0	0	1	9	X
1	0	1	0	10	
1	0	1	1	11	X
1	1	0	0	12	
1	1	0	1	13	X
1	1	1	0	14	
1	1	1	1	15	X

The 4 counters of the domain can be used independently (quad-event mode) or used together (single-event mode). The single-event mode allows counting more complex events but

it is not discussed here. PCOUNTER also has a record mode which allows saving the content of the counters in a buffer in VRAM periodically so that the driver does not have to poll the counters as often. The full documentation of PCOUNTER can be found at hwdocs/pcounter.txt in envytools [10]. Unfortunately, the semantic of most PCOUNTER signals is still unknown.

The MP counters do not support the record and the single-event modes. However, contrarily to PCOUNTER, the MP counters are bound to a channel. This means they are only counting the events for the application that requested them and not monitoring the GPU as a whole. Each application can configure the MP counters the way they want without interfering with any other application. Configuring the MP counters can be done through the pushbuffer and the help of a handful software methods. A limitation of the MP counters is that they can only monitor PGRAPH-related signals. All the hardware events monitorable through NVIDIA's Cupti [25] are now also monitorable on Kepler thanks to the work of Christoph Bumiller. Support for Fermi and older card is currently being implemented by Samuel Pitoiset as part of his Google Summer of Code 2013 project [27].

IV. PTHERM : THERMAL & POWER BUDGET

PTHERM is a piece of hardware that monitors the GPU in order to make sure it does not overheat or exceed its power budget.

A. Thermal management

The primary function of PTHERM is to make sure the GPU does not exceed its temperature budget. PTHERM can react to some thermal events by automatically setting the fan speed to 100%, lowering some frequencies of the GPU, or shutting down the power to the card.

Reading the temperature from the internal sensor can be done simply by reading a register which exposes the temperature in degrees Celsius. The sensor's calibration was performed in factory and stored in the card's fuses. This allows the GPU to monitor its temperature even at boot time and without the need of an external driver.

PTHERM generates thermal events on reaching several temperature thresholds. Whenever the temperature reaches a threshold, an interrupt request (IRQ) can be sent to the host for it to take actions to lower the temperature. The IRQ can be sent conditionally, depending on the direction of the temperature (rising, falling or both). The hardware can also take actions to lower the temperature by forcing the fan to 100% or by automatically lowering the clock frequency. The latter feature will be explained in the following subsections. However, only 3 thresholds can generate automatic hardware response. The others are meant to be used by the driver.

In the case where the GPU is using an external temperature sensor, hardware events are gathered through the chip's GPIO pins which are connected to the external sensor. The external chip is then responsible for monitoring the temperature and comparing it to certain thresholds. These thresholds are programmable via I^2C and are set at boot time during the POST procedure and again when the driver is loaded by the host computer.

B. Power reading

Estimating the power consumption can be done in real time using two different methods.

The first one is to read the power consumption by measuring the voltage drop across a shunt resistor mounted in series with the chip's power line. This voltage drop is linear with the current flowing through the power line with a factor of R_{shunt}. The instantaneous power consumption of the chip is then equal to the voltage delivered by the voltage controller times the measured current. This method is explained in figure 8.

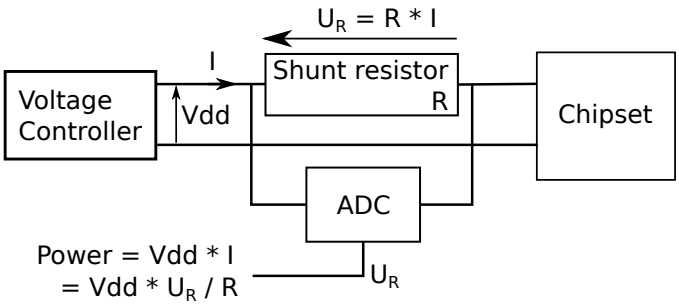

Figure 8. Measuring the power consumption of a chip

However, this technique requires an Analog-to-Digital-Converter (ADC) and some dedicated circuitry. The cost of this solution is quite high as it requires dedicated hardware on the PCB of the GPU and a fast communication channel between the ADC and the chip. Also, fast ADCs are expensive. Therefore, it explains why only the voltage controllers from high-end cards can output this information.

Another solution is to monitor power consumption by monitoring the block-level activity inside the chip. As explained by one of NVIDIA's patents [28]. Power consumption can be estimated by monitoring the activity of the different blocks of the chip, giving them a weight based on the number of gates they contain, sum all the values, low-pass filter them then integrate over the refresh period of the power estimator. This is very much like the approach explained in [26], where performance counters were used to compute the power consumption, except that it is done by the hardware itself. This solution was introduced by NVIDIA in 2006, is explained in patent [28] and is told to produce a power estimate every 512 clock cycles of an unspecified clock. In our case, it seems to be the host clock, sourced by the PCIe port and usually running at 277 MHz. Polling the power estimation register seems to validate this theory as the refresh rate seems to be around 540 kHz.

NVIDIA's method is thus very fast and cheap as it only needs a small gate count increase on the GPU. Moreover, the output of this method can be used internally to dynamically adjust the clock of some engines to stay inside the power budget. This technique is explained in the following subsection.

Unfortunately, on GPU families Fermi and newer, NVIDIA stopped specifying the activity block weights which disables power readings. It is still possible to specify them manually to get the power reading functionality back. However, those weights would have to be calculated experimentally.

C. FSRM: Dynamic clock frequency modulation

PTHERM's role is also to keep the GPU in its power and thermal budget. When the GPU exceeds any of its budgets, it needs to react by lowering its power consumption. Lowering the power consumption is usually done by reclocking the GPU but full reclocking cannot be done automatically by the hardware because it cannot calculate all the parameters and follow the reclocking process on its own.

Letting the driver reclock the GPU when getting close to overheating is acceptable and PTHERM can assist by sending an IRQ to the driver when the temperature reaches a threshold. However, in the case where the driver is not doing its job, because it is locked up or because the driver is not loaded, the chip should be able to regulate its temperature without being forced to cut the power to the GPU.

In the case of meeting the power budget, reacting quickly to an over-current is paramount to guarantee the safety of the power supply and the stability of the system in general. It is thus very important to be able to reclock often.

It is not possible to reprogram the clock tree and adjust the voltage quickly enough to meet the 540 kHz update rate needed for the power capping. However, the clock of some engines can be slowed down. This will linearly affect the dynamic part of the power consumption albeit not as power efficient as full reclocking of the GPU because the voltage is not changed.

A simple way to lower the frequency of a clock is to divide it by a power of two although the granularity is too coarse to be used directly for the power capping capability. It is however possible to lower the average clock frequency by sometimes selecting the divided clock and then selecting the original clock the rest of the time. For the lack of a better name, I decided to call this frequency-modulation technique Frequency-Selection Ratio Modulation (FSRM). FSRM can be implemented by using the output of a Pulse-Width Modulator (PWM) to a one bit multiplexer. When the output of the PWM is high, the original clock is being used while the divided clock is used when the output is low. Any average clock frequency between the divided clock and the original clock is thus achievable by varying the duty cycle of the PWM.

Figure 9 presents the expected frequency response of the above system along with what has actually been measured through the performance counters when tested on NVIDIA's hardware implementation. Judging by the differences, it seems like NVIDIA also added a system to smooth the change between the slowed and the normal clock. The difference is also likely explained by the fact that the clock selection may only happen when both the original and the divided clock are rising. This also raises the problem of synchronising the original and the divided clock as the divided clock has to go through more gates than the original one. In order to synchronise them, the original clock would have to be shifted in time by adding redundant gates. This issue is known as clock skew [29].

D. FSRM usage & configuration

FSRM is used to modulate PGRAPH clock's frequency. PGRAPH is the main engine behind 2D/3D acceleration and GPGPU, and probably responsible for most of the power consumption.

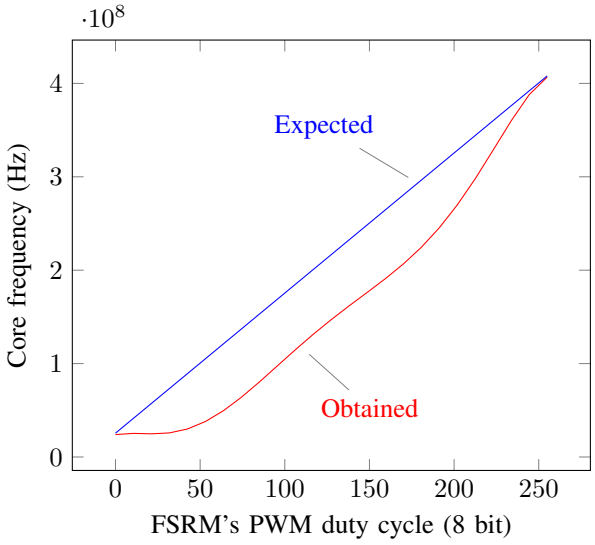

Figure 9. Frequency of the core clock (@408MHz, 16-divider) when varying the FSRM

There are several events that can trigger use of FSRM:

- PGRAPH idling;
- Temperature reaching one of the several thresholds;
- Power usage reaching its cap;
- Driver forcing FSRM.

Whenever one of these events happens, the divided clock and the FSRM value get updated following to their associated configuration. The clock divisor can be set to 1, 2, 4, 8 or 16 while the FSRM value can range from 0 (use the divided clock all the time) to 255 (use the normal clock all the time). However, even though each temperature threshold can specify an independent clock divisor, they have to share the same FSRM value.

Some preliminary tests have been performed on an nv84 to lower the clock to the maximum when PGRAPH is idle and this resulted in a about 20% power reduction of the computer while the impact on the framerate of a video game was not measurable. Some cards do not enable this feature by default, possibly suggesting that it may lead to some instabilities. This is however really promising and will be investigated further.

In the case of the power limiter, another mode can be selected to automatically update the FSRM value. This allows lowering the frequency of PGRAPH as little as possible in order to stay within the power budget. This mode uses 2 hysteresis windows, one containing the other entirely. Each window will increase the effective frequency (using FSRM) whenever the power reading is below its low threshold and decrease the effective frequency when the reading is above its high threshold. The increase and decrease in FSRM values are independent for both hysteresis windows and can be set arbitrarily. However, the outer window is supposed to increase or decrease the effective frequency rapidly while the inner window is supposed to make finer adjustments.

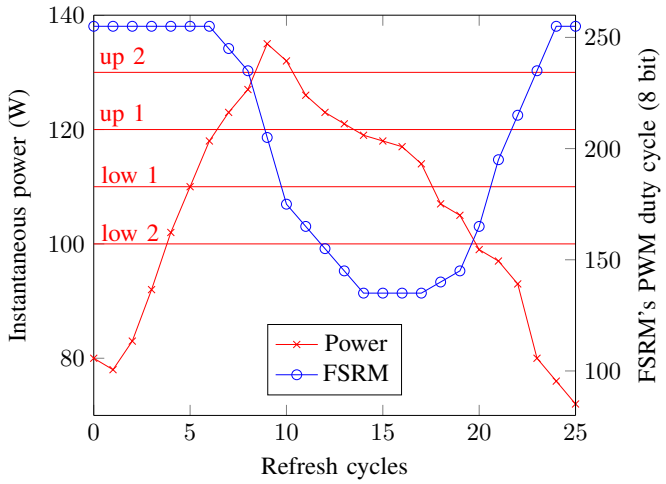

Figure 10. Example of the power limiter in the dual window mode

An example can be seen in figure 10 where the outer window is set to [130W, 100W] while the inner window is set to [120W, 110W]. The outer window will increase the FSRM value by 20 when the power is lower than 100W and will decrease it by 30 when the power is above 130W. The inner window will increase the FSRM value by 5 when the power is between 120 and 130W and will decrease it by 10 when the power is between 100 and 110W. The FSRM value is limited to the range [0, 255].

E. Power limiter on Fermi and newer

As no power reading is possible by default on GPUs of the Fermi family and newer, the power limiter cannot be used. This came to me as a surprise as NVIDIA started advertising the power limitation on Fermi. This suggests that they may have implemented another way of reading and lowering the power consumption of their GPUs. I unfortunately do not have access to such a card but the independent and proprietary tool GPU-Z [30] proposes a way to disable this power cap, as can be seen on figure 11.

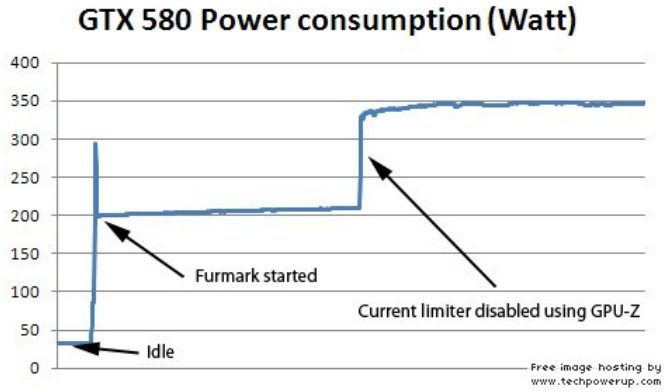

Figure 11. Effect of disabling the power limiter on the Geforce GTX 580. Copyrights to W1zzard from techpowerup.com.

The first spike would seem to suggest a very slow response to exceeding the power budget. It is thus possible that NVIDIA would use its driver to poll the voltage controller's power reading and decide to reclock the card to a lower performance level. Since GPU-Z is proprietary, more reverse engineering will be needed to understand and document this feature.

V. PDAEMON : An RTOS embedded in your GPU

PDAEMON is an engine introduced on nva3, a late GPU from the Tesla family. This engine is fully programmable using the instruction set (ISA) FμC (Flexible MicroCode).

FμC was introduced in nv98 for PCRYPT, an engine meant to offload some cryptographic operations. FμC was then reused for PDAEMON and many more engines of the Fermi family. This feature-rich ISA is now being used to implement some of the interface between the host and the hardware of some engines thus bringing a lot more flexibility to the hardware's interfaces with the host. An assembler has been written for this ISA and is available under the name envyas in the envytools repository [10]. An experimental Open Source FμC compiler [31] has been developed by Shinpei Kato and his research team for their needs. It is implemented as an LLVM backend and will require some modifications in order to be able to produce binaries for PDAEMON.

PDAEMON is an engine meant to offload some operations usually performed by the host driver. It is clocked at 200 MHz, has a memory management unit (MMU), has access to all the registers of the GPU and direct access to PTHERM. It also supports timers, interrupts and can redirect the interrupts from the GPU to itself instead of the host. Several independent communication channels with the host are also available. Surprisingly, it also has performance counters to monitor some engines' activity along with its own. In other words, PDAEMON is a fully-programmable "computer" with low-latency access to the GPU that can perform more efficiently whatever operation the host can do. However, it cannot perform heavy calculations in a timely fashion because of its limited clock frequency.

For more information about PDAEMON's capabilities, please read hwdocs/pdaemon.txt and all the files from hwdocs/fuc/ found in the envytools repository [10].

A. Usages of PDAEMON

PDAEMON's usage by NVIDIA has not been fully reverse-engineered yet. However, the uploaded microcode contains the list of the processes executed by the RTOS developed by or for NVIDIA. Here are some of the interesting processes:

- Fan management;
- Power gating;
- Hardware scheduling (for memory reclocking);
- Power budget enforcement;
- Performance and system monitoring.

Its usage list could also be extended to cover:

- Parsing the VBIOS;
- Implementing full DVFS support;
- Generating power-usage graphs.

B. Benefits of using PDAEMON

PDAEMON has clearly been developed with the idea that it should be able to do whatever the host system can do. One of the practical advantages of using PDAEMON to implement power management is that the CPU does not need to be woken as often. This lowers the power consumption as the longer the CPU is allowed to sleep, the greater the power savings are.

Another benefit of using PDAEMON for power management is that, even when the host system has crashed, full power management is still available. This has the benefit of checking the thermal and power budget of the GPU while also lowering the power consumption of crashed system.

VI. The GPU as an autonomic-ready system

In 2001, IBM proposed the concept of autonomic computing [32] to aim for the creation of self-managing systems as a way to reduce their usage complexity. The idea was that systems are getting more and more complex and, as such, require more knowledge from the technicians trying to maintain them. By having self-managing systems, the user could write a high-level policy that would be enforced by the system itself, thus hiding complexity.

As an example, a modern NVIDIA GPU could perform the following self-functions:

self-configuration: The GPU is responsible for finding a configuration to fill the user-requested tasks.

self-optimization: Using performance counters, the GPU can optimise performance and also lower its power consumption by using DVFS.

self-healing: As the GPU can monitor its power consumption and temperature, it can also react to destructive behaviours by lowering the frequency of the clocks.

self-protection: Isolation between users can be provided by the GPU (integrity and confidentiality) while availability can be achieved by killing long-running jobs run by users. The GPU can also store secrets like HDCP [33] keys or even encrypt/decrypt data on the fly using PCRYPT.

Although the aim of autonomic computing is primarily oriented towards lowering human maintenance cost, it can also be extended to lower the development cost. By having self-managing subsystems for non-functional features, integration cost is lowered because of the reduced amount of development needed to make it work on a new platform. Ideally, a complete system could be assembled easily by using unmodified autonomic subsystems and only a limited amount of development would be needed to make their interfaces match.

This approach does not make sense in the IBM-PC-compatible personal computer market as the platform has barely evolved since its introduction in the way that components interact (POST procedure, x86 processors and high speed busses). This renders the development of drivers executed on the CPU cheaper than having a dedicated processor for the driver's operations. However, in the System-On-Chip (SoC) world where manufacturers buy IPs (intellectual property blocks) and assemble them in a single-chip system, the dedicated-processor approach makes a lot of sense as there is no single processor ISA and operating system (OS) that the IP manufacturer can depend on. In this context, the slimmer the necessary driver for the processor and operating system, the wider the processor and OS choice for the SoC manufacturer.

This is the approach that Broadcom chose for its Videocore technology which is fully controllable by a clearly-documented Remote Procedure Calls (RPC) interface. This allowed the Raspberry Pi foundation to provide a binary-driver-free Linux-based OS on their products [34]. However, the Open Source driver only uses this RPC interface and is thus not a real driver as it is just some glue code. This led the graphics maintainer of Linux to refuse including this driver in Linux as the code running in Videocore is still proprietary and bugs there are unfixable by the community [35].

Broadcom's Videocore is a good example of both the advantages and the drawbacks of autonomic systems. Adding support for it required very limited work by the Raspberry Pi foundation but it also means the real driver is now a black box running on another processor which cannot be traced, debugged, or modified easily. From an open source operating system point of view, it also means that it will become much harder to study the hardware and propose new drivers. In the case of NVIDIA, this situation could happen if PDAEMON's code was not readable and writable by the host system.

VII. Conclusion

In this paper, we have presented an overview of the source of power consumption in modern processors along with different techniques implemented by NVIDIA to lower it. We also presented how different features available in the GPU can be combined to improve its efficiency. It is my hope that this article will spur more interest in the community to study, document and improve on the current drivers to make them more power-efficient.

Future work will focus on creating a stable reclocking process across all the modern NVIDIA GPUs. This work will allow us to implement a DVFS algorithm. More work will also be done on power- and clock-gating which are the two main power management features which have not been documented yet.

Acknowledgment

The author would like to thank everyone involved in graphics on Linux or other FLOSS operating systems for laying out the foundations for the work done in Nouveau. I would however like to thank in particular Marcin Kościelnicki for reverse engineering and writing most of the documentation and tools found in envytools. I would also like to thank the Nouveau maintainer, Ben Skeggs, for figuring out many VBIOS tables along with the clock tree on most cards and reviewing the code I wrote for Nouveau DRM. I should also thank Roy Spliet for working with me on figuring out how to calculate the memory timings based on the parameters found in the memory timings table for the nv50 family and proof-reading this paper. I would also like to thank Ilia Mirkin for our discussions and his help on rewording this paper. Finally, I would like to thank my Ph.D. supervisor Francine Krief for funding me and letting me work on this project when time permits.

REFERENCES

[1] EPA, "EPA report to congress on server and data center energy efficiency," Tech. Rep., 2007. [Online]. Available: http://hightech.lbl.gov/documents/data_centers/epa-datacenters.pdf

[2] "Aalto talk with linus torvalds [full-length]," Jun. 2012. [Online]. Available: https://www.youtube.com/watch?v=MShbP3OpASA\&t=2894s

[3] Nouveau Community, "Nouveau, the community-driven open source NVIDIA driver," 2006. [Online]. Available: http://nouveau.freedesktop.org/wiki/

[4] NVIDIA, "xf86-video-nv : NVIDIA's open source driver," 2003. [Online]. Available: http://cgit.freedesktop.org/xorg/driver/xf86-video-nv/

[5] X.org, "Direct rendering infrastructure." [Online]. Available: http://dri.freedesktop.org/wiki/

[6] M. Peres, "[nouveau] [RFC] initial power management vbios parsing, voltage & clock setting to nouveau." Sep. 2010. [Online]. Available: http://lists.freedesktop.org/archives/nouveau/2010-September/006499.html

[7] PCI-SIG, "PCI express," 2013. [Online]. Available: http://www.pcisig.com/specifications/pciexpress/resources/

[8] NXP, "I2C-Bus: what's that?" [Online]. Available: http://www.i2c-bus.org/

[9] C. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, no. 2, pp. 50–59, 2011.

[10] "Envytools - tools for people envious of nvidia's blob driver." [Online]. Available: https://github.com/pathscale/envytools

[11] Pekka Paalanen, "MMIO-trace," 2008. [Online]. Available: http://nouveau.freedesktop.org/wiki/MmioTrace/

[12] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, and S. Kato, "Power and performance analysis of GPU-accelerated systems," in *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, ser. HotPower'12. Berkeley, CA, USA: USENIX Association, 2012, p. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387869.2387879

[13] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, p. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924921

[14] R. Singhal, "POWER GATING TECHNIQUE TO REDUCE POWER IN FUNCTIONAL AND TEST MODES," U.S. Patent 20 100 153 759, Jun., 2010. [Online]. Available: http://www.freepatentsonline.com/y2010/0153759.html

[15] Z. Y. Zheng, O. Rubinstein, Y. Tan, S. A. Jamkar, and Y. Kulkarni, "ENGINE LEVEL POWER GATING ARBITRATION TECHNIQUES," U.S. Patent 20 120 146 706, Jun., 2012. [Online]. Available: http://www.freepatentsonline.com/y2012/0146706.html

[16] S. C. LIM, "CLOCK GATED PIPELINE STAGES," Patent WO/2007/038 532, Apr., 2007. [Online]. Available: http://www.freepatentsonline.com/WO2007038532A2.html

[17] K. M. Abdalla and R. J. Hasslen III, "Functional block level clock-gating within a graphics processor," U.S. Patent 7 802 118, Sep., 2010. [Online]. Available: http://www.freepatentsonline.com/7802118.html

[18] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[19] Radeon Community, "Radeon - an open source ATI/AMD driver." [Online]. Available: http://xorg.freedesktop.org/wiki/radeon

[20] Matthew Garrett, "Radeon reclocking," Apr. 2010. [Online]. Available: http://mjg59.livejournal.com/122010.html

[21] X.org, "X.org developer conference 2011," Sep. 2011. [Online]. Available: http://www.x.org/wiki/Events/XDC2011

[22] Betty Luk, "Understanding PCIe 2.0 bandwidth management," PCI-SIG, 2008. [Online]. Available: http://www.pcisig.com/developers/main/training_materials/get_document?doc_id=090831b9a2b1210b2822c06e469992d9d028f13d

[23] Matthew Garrett, "Matthew garrett responds to the ASPM power regression," Jun. 2011. [Online]. Available: https://lwn.net/Articles/449648/

[24] NVIDIA, "PerfKit." [Online]. Available: https://developer.nvidia.com/nvidia-perfkit

[25] ——, "CUDA profiling tools interface." [Online]. Available: https://developer.nvidia.com/cuda-profiling-tools-interface

[26] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Green Computing Conference, 2010 International*, 2010, pp. 115–122.

[27] Samuel Pitoiset, "Samuel pitoiset's open source blog," 2013. [Online]. Available: https://hakzsam.wordpress.com/

[28] H. Cha, R. J. Hasslen III, J. A. Robinson, S. J. Treichler, and A. U. Diril, "Power estimation based on block activity," U.S. Patent 8 060 765, Nov., 2011. [Online]. Available: http://www.freepatentsonline.com/8060765.html

[29] J. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. 39, no. 7, pp. 945–951, 1990.

[30] W1zzard, "TechPowerUp GPU-Z v0.7.1," 2013. [Online]. Available: http://www.techpowerup.com/downloads/2244/techpowerup-gpu-z-v0-7-1/

[31] Shinpei Kato, "guc," 2013. [Online]. Available: https://github.com/CS005/guc

[32] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[33] Digital Content Protection, LLC, "High-bandwidth digital content protection (HDCP)." [Online]. Available: http://www.digital-cp.com/

[34] Alex Bradbury, "Open source ARM userland," Oct. 2012. [Online]. Available: http://www.raspberrypi.org/archives/2221

[35] Dave Airlie, "raspberry pi drivers are NOT useful," Oct. 2012. [Online]. Available: http://airlied.livejournal.com/76383.html