Power and Performance Aware Memory-Controller Voting Mechanism

Milena Vratonjic, Harmander Singh*, Gautam Kumar**, Roumi Mohamed**, Ashish Bajaj**, Ken Gainey
Qualcomm Atheros, Inc.; *Qualcomm Technologies Inc.; **Qualcomm India Private Limited
E-mail: milenav@qti.qualcomm.com

Abstract

Modern System-on-Chips (SoCs) integrate a graphics unit (GPU) with many application processor cores (CPUs), communication cores (modem, WiFi) and device interfaces (USB, HDMI) on a single die. The primary memory system is fast becoming a major performance bottleneck as more and more of these units share this critical resource. An Integrated-Memory-Controller (IMC) is responsible for buffering and servicing memory requests from different CPU cores, GPU and other processing blocks that require DDR memory access. Previous work [2] was focused on appropriately prioritizing memory requests and increasing IMC/DDR memory frequency to improve system performance – which came at the expense of higher power consumption. Recent work has addressed this problem by using a demand based approach. This is accomplished by making the IMC aware of the application characteristics and then scaling its frequency based on the memory access demand [1]. This leads to lower IMC and DDR frequencies and thus lower power. The work presented here shows that instead of lowering the frequency, greater total system power savings can be achieved by *increasing* IMC frequency at the beginning of a use-case that has moderate GPU utilization. The primary motivation behind this approach is that it allows GPU, with its inherent ability to execute a larger number of parallel threads, to access memory faster and therefore complete its processing portion of the execution pipeline faster. This, in turn, allows relaxation of the timing requirements imposed on the CPU pipeline portion and consecutive cycles, thus saving on total system power. An algorithm for this technique, along with the silicon results on an SoC implemented in an industrial 28nm process, will be presented in this paper.

Keywords

Memory, DDR, IMC, Low-Power

1. Introduction

An Integrated-Memory-Controller (IMC) is responsible for scheduling the memory access from different units, such as a GPU or many CPUs, and also sets the frequency of the DDR to meet performance requirements. Most modern microprocessors (e.g. Intel's Core i7 [3][4], Sandy Bridge [5]) have an integrated memory controller in order to reduce memory latency. An IMC increases a system's performance and reduces the cost by eliminating the need for an external memory controller. However, it also poses a constraint to using only a certain type of memory. For that reason, some CPUs have a dedicated external memory controller (e.g. Centaur memory controller chip in IBM's POWER8 microprocessor [6]). The Centaur chip [6] is using DDR3 memory, but a future version can use DDR4 without the need for POWER8 to be replaced itself.

The goal of the work presented in this paper is to find the optimum IMC frequency such that the total power of the system is minimized without any performance or reliability degradation. Such an optimal IMC frequency would be selected dynamically and adjusted for each use-case. Using the proposed method, the measured results show both power and performance are improved. The proposed method is implemented in an industrial 28nm SoC. Silicon measurements show up to a 15% reduction in power for web-browsing and gaming use-cases.

The base method that is currently implemented is called a demand based approach [1]. It finds the minimum IMC frequency that satisfies the system performance requirement. Performance counters in the L2 cache controller are used to collect statistics in such a way that can be used to appropriately scale the memory bus and DDR frequency. The effective bandwidth is then used to compute a new higher/lower IMC frequency and, consequently, bus and DDR frequencies to accommodate the increase/reduction in memory activity. These system counters are configured to continue monitoring the bandwidth. A similar scenario is used for the CPU's cache and main DDR. If performance requirements are not met, the algorithm increases the GPU's and CPU's operating frequencies and adjusts for a higher IMC frequency. This mechanism is part of the GPU's and CPU's Dynamic Clock and Voltage Scaling (DCVS) algorithm and is called demand based approach. With such demand based approach, we start a run with lower IMC frequencies until there is a demand for higher either due to increased memory access or in case performance is not met.

Our solution proposes that instead of starting with the minimum IMC frequency which satisfies performance requirements, the memory controller should start a use-case run

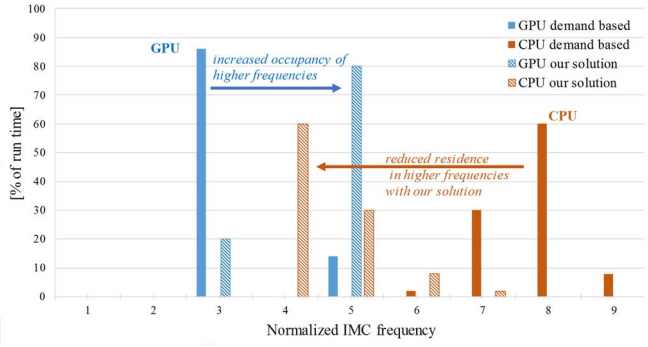

Figure 1: IMC frequency occupancy during a run: demand based algorihtm vs. our solution

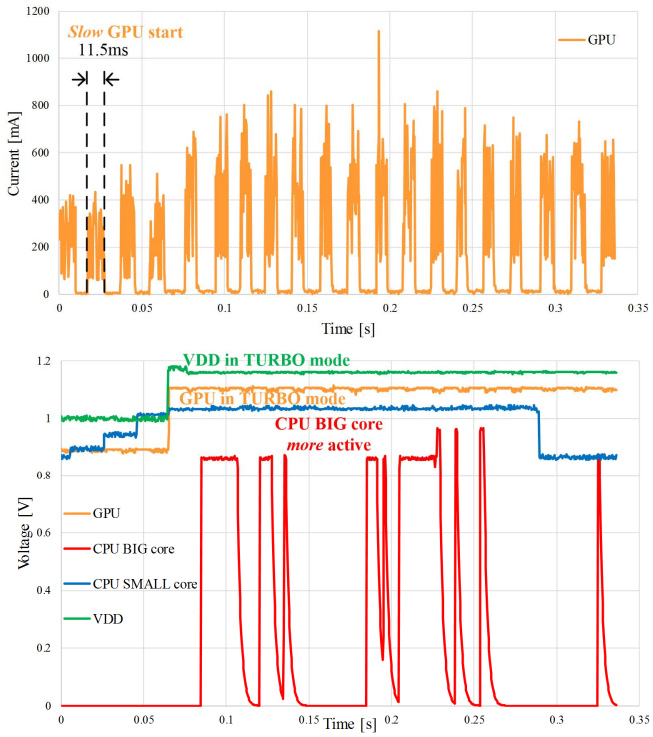

Figure 2: Demand based implementaiton with IMC starting at default frequency.

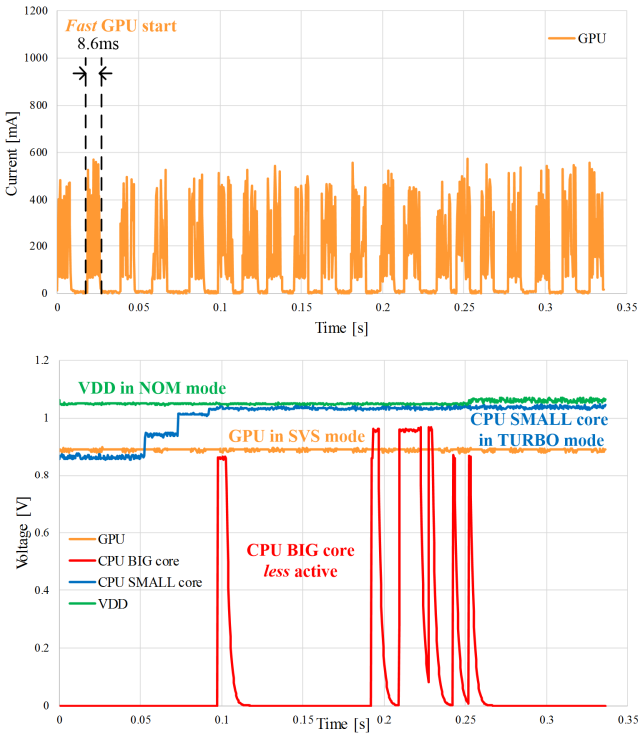

Figure 3: Proposed solution with IMC starting at almost twice higher than default frequency.

with a higher IMC frequency – especially for use-cases which have moderate to high GPU utilization. This will result in the GPU occupying a higher IMC frequency spectrum, as opposed to a low IMC frequency range, illustrated in Figure 1. The CPU will, contrary to the GPU, exhibit reduced occupancy of higher IMC frequencies and will operate at lower CPU frequencies (shown later), resulting in total system power reduction and also improved system performance.

2.1. Current Implementation

Figure 2 shows the current implementation based on a demand based approach for a web-browser use-case example. The GPU rendering timeline is initially starting at 11.5ms. This slow start then forces the CPU's DCVS algorithm to increase both its operating and IMC frequencies so that it can compensate and complete the run without performance degradation.

2.2. Proposed IMC Solution

In our proposed solution, the IMC frequency starts at a higher level immediately at the beginning of a run. This allows for the GPU unit to complete its processing portion of the load faster. In turn, the timing requirements for the CPU portion and the consecutive execution of the processing pipeline are relaxed. This results in reduced total system power, considering the steep sensitivity of CPU's power at tighter delay targets and power overhead on the whole system imposed by operating at higher supply voltages required to support higher GPU and IMC frequencies later in the run to make-up for the slow start.

The default IMC frequency for a use-case is determined by finding the minimum frequency that satisfies its performance requirements. However, starting with a higher initial IMC frequency for the GPU (which for the web-browser use-case example used for illustration purposes in Figure 3 is almost twice as high as the default), its active time reduces from 11.5ms to 8.6ms. The GPU completes its processing portion of the task faster and the demand on the CPU to complete its portion is now relaxed. As can be seen from Figure 2 and Figure 3, the BIG core of the CPU cluster, which has high contribution in the total system power, is less active in the proposed approach as compared to the demand based implementation. The SMALL CPU core, which is more energy efficient as compared to the BIG core, is instead utilized more often. The system is not forced to operate in a higher voltage mode. Figure 3 shows the voltages for digital logic (VDD), GPU, as well as BIG core to be lower (as compared to Figure 2) with the proposed solution in place. Performance is not degraded, yet total power is reduced.

Figure 4 shows how the optimal IMC frequency is determined for the web-browser use-case that is analyzed in Figure 2 and Figure 3. Shown are silicon measurement results with proposed solution implemented on a 28nm SoC. The initial IMC frequency in the use-case is chosen out of the entire frequency range available in the system. Utilizing the minimum frequency (which is about half of the value of the default frequency) the use-case will still complete the operation without

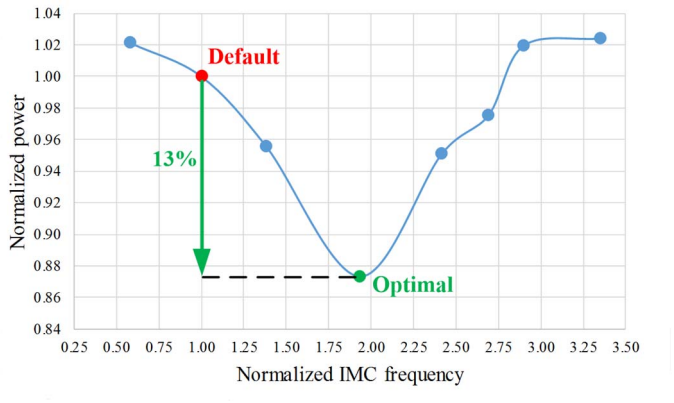

Figure 4: Optimal IMC frequency for Controlled Scroll in a Web-browser use-case (WQXGA screen panel)

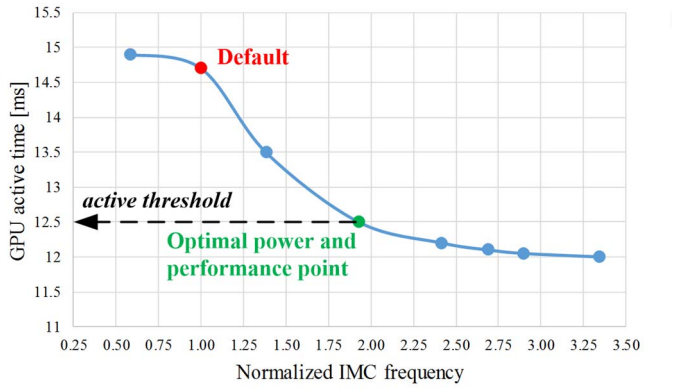

Figure 5: GPU activity vs. optimal IMC frequency

a loss of performance. The penalty is in the high power overhead, as the CPU's DCVS algorithm will have to request higher IMC frequency and higher CPU operating frequencies, similar to previously illustrated behavior in Figure 2, to compensate for the slow GPU start. Thus, total power is much higher than if the GPU/IMC were to start with higher operating frequencies. Further increases of the DDR frequency beyond the optimum point does not translate to more power benefits. There is no further reduction in the GPU timeline that the CPU system can utilize to reduce the demand on its timing. The system overhead needed to support such high IMC operational frequencies diminishes any power savings.

It is important to note that Figure 4 only illustrates the initial starting IMC frequency that affects how the GPU dominated use-case will be executed throughout the run. It does not imply that the IMC will be constant throughout the run, but rather illustrates the initial value set by the GPU unit. During the use-case run, the IMC frequency will change as the CPU will demand for the IMC frequency values that are much higher than default, following with higher demands from GPU as well in the consecutive cycles as the system tries to compensate for slow start and imposing power overhead in return.

3. Algorithm for selecting optimal IMC frequency

As the IMC frequency is increased, the GPU active time is reduced. This behavior is illustrated in Figure 5. The GPU's active time reduces monotonically until a point is reached where the average activity flattens out. At this point, further reduction in the GPU active time is insignificant even as IMC frequency is further increased. This is because the GPU activity actually reaches its capacity and further increasing the frequency does not significantly improve the GPU active timeline. The optimal power and performance point coincides with a threshold in the GPU active time reduction at which IMC frequency reaches its optimum value. This threshold, *active threshold,* is used in the algorithm described in Figure 6 as an indicator that the optimal power and performance point has been achieved.

Our algorithm is incorporated into the GPU's default DCVS algorithm, described in Figure 6, which is based on the idle time duration between the GPU active windows. It monitors the idle time between the times when GPU is active by picking up the largest idle period (gap) among the last consecutive *N* gaps, in a manner of a sliding window. If such defined idle gap is consistently larger than the defined threshold (referred to as the *idle threshold* in Figure 6), the algorithm requests for lower GPU frequency and/or lower IMC frequency. At any point in time, if the algorithm discovers that the gap is less than the *idle threshold*, it increases the power level by increasing the GPU and/or the IMC frequency. Each power level corresponds to a predefined frequency of the GPU and the default IMC frequency. It is possible for the same GPU frequency that there exist several predefined default IMC frequencies. Each {GPU $freq_1$, IMC default $freq_1$}, {GPU $freq_1$, IMC default $freq_2$} pair correspond to a separate power level, in this case two. At a given power level, determined by the primary DCVS algorithm (top segment of the algorithm), the GPU is starting with a pre-set, i.e. default IMC frequency at the beginning of a use-case run. At the same time, the bottom segment of the algorithm is activated and the active time of the GPU is being monitored. If the active time of the GPU is bigger than the *active threshold*, the IMC frequency will be increased to facilitate further reduction in the GPU's active time, as depicted in Figure 5. Once the GPU active time reaches the *active threshold*, the indication is raised that the optimal IMC frequency is reached. This optimal IMC frequency is then recorded and set as a new default starting point (i.e., IMC frequency) for the current GPU power level. It takes only a few active GPU cycles of iterations for the proposed GPU DCVS algorithm to be fully effective in the current run of the use-case that is being executed. In the second run of the same use-case, the GPU DCVS algorithm will now have the new optimal starting IMC frequency which was determined in the previous run. The GPU DCVS algorithm continuously monitors the system requirements, and the proposed modifications allow for a dynamic improvement that achieves optimal power and performance balance for each use-case.

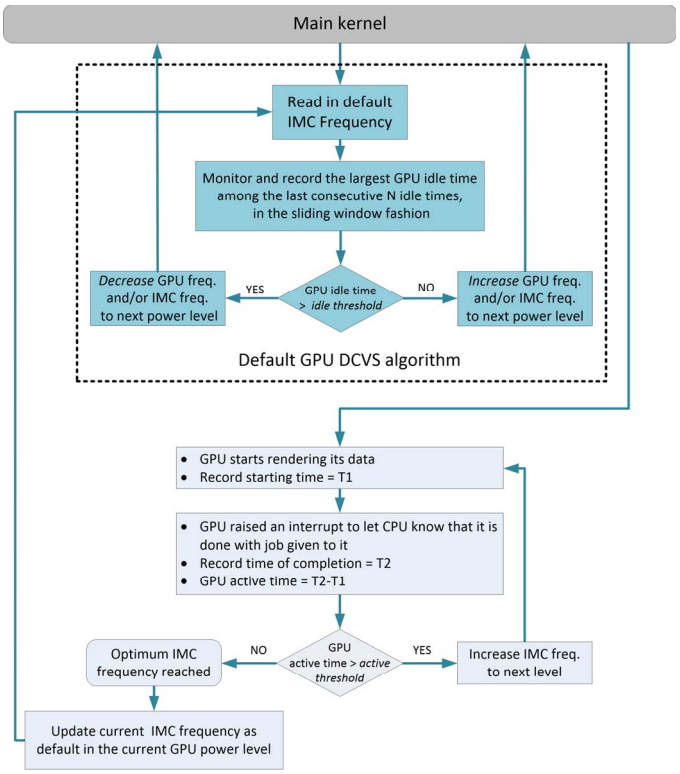

Figure 6: GPU's DCVS algorithm

4. Results and summary

In use-cases with moderate to high GPU utilization, the algorithm presented in this paper shows that the optimal starting GPU IMC frequency is higher than the currently chosen default value selected using demand based approach. Table I shows a dashboard of typical use-cases ran on modern-day mobile SoCs and the multiplier of the increase in IMC starting frequency from the default value to reach the optimal power and performance point.

The proposed algorithm is implemented on an industrial 28nm SoC, and the measurements show up to a 15% reduction in power for web-browsing and gaming use-cases. This is achieved by time aware control of the IMC frequency which is the optimization parameter used to achieve optimal power and performance tradeoff between GPU, CPU and other system components.

In use-cases where the GPU is highly utilized, the initial IMC frequency is already very high. Further increase of IMC frequency does not help to reduce total system power because the reduction in active GPU time is negligible whereas the cost in terms of power for a system to support and run at such high IMC (DDR) frequencies is very high. This is the case, for example, in the "Egypt 60fps" use-case in Table I. For all other use-cases with moderate to high GPU utilization (13%-75%), the results in Table I show that the optimal IMC frequency is 1.3 to 3.3 times bigger than the default.

Table I: Optimal IMC freq. for various use-cases commonly ran on modern mobile SoCs

Use-case	GPU DCVS Tuning			
	FHD Panel		*WQXGA panel*	
	GPU utilization [%]	Optimal IMC freq. [x Default]	GPU utilization [%]	Optimal IMC freq. [x Default]
Powerlift	52	1.6	66	1.7
Controlled Scroll in Contacts	43	1.3	69	1.8
Controlled Scroll in Web-Browser	28	1.3	50	1.8
Scroll Fling in Contacts	54	1.7	75	1
Scroll Fling in Web-Browser	26	2.4	46	3.3
Angry Birds 60fps	60	2.7	74	1.8
Live Wallpaper Bubble	13	1.6	18	2.7
Egypt 60fps	90	1.1	99	1

5. Acknowledgement

The authors would like to acknowledge help from the GPU, CPU, system and memory teams within QCA, QTI and Qualcomm India Private Limited for their support and numerous useful discussions.

6. References

[1] S. Kannan, "Dynamic scaling of memory and bus frequencies", USPTO US 14-176,268, April 16th 2015.

[2] J. Carter, et. al, "Impulse: Building a smarter memory controller", Fifth International Symposium on High-Performance Computer Architecture, 1999, pp. 70-79.

[3] R. Singhal, "Inside Intel next generation Nehalem architecture", Hot Chips: A Symposium on High Performance Chips, Vol. 20, 2008, Stanford, Palo Alto, CA.

[4] S. Kottapalli, J. Baxter, "Nehalem-ex CPU architecture", Hot Chips: A Symposium on High Performance Chips, Vol. 21, 2009.

[5] M. Yuffe, et. al, "A Fully integrated multi-CPU, GPU and memory controller 32nm processor", ISSCC, 2011.

[6] J. Stuecheli, "Next Generation POWER microprocessor", Hot Chips: A Symposium on High Performance Chips, Vol. 25, 2013, Stanford, Palo Alto, CA.

[7] Min Kyu Jeong, et. al, "A QoS-aware memory controller for dynamically balancing GPU and CPU bandwidth use in an MPSoC", DAC, San Francisco, Jun 2012.

[8] N. Agarwal, et al, "Selective GPU caches to eliminate CPU-GPU HW cache coherence", Int. Symposium on High Performance Computing Architecture (HPCA), 2016.