Impact of Memory Frequency Scaling on User-centric Smartphone Workloads

Hashan R. Mendis[1], Wei-Ming Chen[2], Leandro Soares Indrusiak[1], Tei-Wei Kuo[2], Pi-Chen Hsiu[3]
[1]Real-time Systems Group, University of York, UK
[2]Dept. of Computer Science and Information Engineering, National Taiwan University, Taiwan
[3]Research Center for Information Technology Innovation, Academia Sinica, Taiwan
hrm506@york.ac.uk, d04922006@csie.ntu.edu.tw, lsi@cs.york.ac.uk, ktw@csie.ntu.edu.tw,
pchsiu@citi.sinica.edu.tw

ABSTRACT

Improving battery life in mobile phones has become a top concern with the increase in memory and computing requirements of applications with tough quality-of-service needs. Many energy-efficient mobile solutions vary the CPU and GPU voltage/frequency to save power consumption. However, energy-aware control over the memory bus connecting the various on-chip subsystems has had much less interest. This measurement-based study first analyse the CPU, GPU and memory cost (i.e. product of utilisation and frequency) of user-centric smartphone workloads. The impact of memory frequency scaling on power consumption and quality-of-service is also measured. We also present a preliminary analysis into the frequency levels selected by the different default governors of the CPU/GPU/memory components. We show that an interdependency exists between the CPU and memory governors and that it may cause unnecessary increase in power consumption, due to interference with the CPU frequency governor. The observations made in this measurement-based study can also reveal some design insights to system designers.

1 Introduction

As smartphone applications become enriched with newer features and smoother user-experience, their computing and memory requirements also increase, leading to higher power consumption. Therefore, energy-efficient personal computing without significant impact to user experience, has become the top-most research concern. From a hardware perspective, modern smartphone system-on-chips (SoCs) can contain multiple dedicated IPs and a heterogeneous power-efficient multi-core mobile processor (e.g. ARM big.LITTLE processor [1]). From a software perspective, OS driven power saving techniques such as Dynamic voltage and frequency scaling (DVFS) are commonly used, where the operating frequency of compute cores is scaled based on a performance metric such as system load.

The communication between on-chip IP components takes place via one or more internal buses and shared main memory. Hence, the energy consumed by the memory subsystem and bus fabric can also be significant, yet it has been given limited attention. As smartphones are increasingly used for media-rich, high-memory throughput applications, memory bus related power consumption can outweigh the CPU and GPU [4]. Therefore, several recent work have started focusing on memory and bus power saving techniques (e.g. [3] [6]).

This work investigates power consumption, on-chip resource usage characteristics and quality-of-service (QoS) related to memory bus frequency scaling, specifically for user-centric smartphone workloads. The overall contributions made in this paper are as follows:

- We characterise several common smartphone macro-workloads in terms of CPU, GPU and memory bus usage and show how different application workloads can result in varying resource usage.
- We investigate the effect on CPU and GPU usage by changing the memory bus frequency.
- We analyse the interplay between CPU and memory bus frequency scaling and give examples of how unnecessary energy dissipation can occur.
- From our observations, useful design suggestions are drawn, which can help system designers to efficiently balance power consumption and QoS.

2 Related work

2.1 Smartphone workload characterisation

In [5], it is shown that in certain micro-workloads that have a high memory footprint, RAM power consumption can exceed CPU power by a small margin. Their subsequent work shows the MIF (memory interface) and INT (internal) bus power consumption can use more energy than the RAM, CPU or GPU in idle states as well as in interactive scenarios (e.g. gaming) [4]. Pandiyan and Wu [18], show that 28-40% of total mobile system power consumption can be due to data movement across the memory hierarchy, especially in the case of applications with high cache miss rates (e.g. HD video playback).

ACM SAC '18 April 9-13, 2018, Pau, France
©2017 ACM. ISBN XXX-XXXX-XX-XXX/XX/XX...$xx.xx
DOI: XX.XXX/XXX_X

Initial studies by Gutierrez et al. [9], show that standard micro-benchmarks such as SPEC2000 do not accurately reflect the resource usage of user-centric interactive smartphone applications. Gao et al. [8] observe that offloading applications with high thread/data-level parallelism onto GPUs or accelerators can offer better energy efficiency. The speed of user interaction (e.g. scrolling speed on a web browser) can also impact the amount of CPU/GPU usage and respective memory bandwidth used by the application; as well as cause significant fluctuations in power consumption [20].

2.2 Smartphone power management

The Linux *ondemand* CPU frequency governor is a popular kernel-level entity that increases the clock frequency to the maximum when the CPU load is above a pre-defined threshold and decreases the frequency gradually when the CPU load is below a lower threshold [17]. However, load-dependent frequency governors often raise the frequency higher than required by the target QoS, thus wasting power consumption [12].

Co-operative, runtime CPU and GPU frequency scaling has been shown to reduce power consumption by 58% (at the desired QoS level), over the default ondemand governor, for mobile 3D games [19]. Similarly, Chen et al. [7] identify CPU-bound and GPU-bound phases of a mobile 3D game to set the CPU-GPU frequencies according to the game phase and user interactivity. They are able to outperform previous work [19], for highly interactive 3D games with frequent phase switching. The work in [19] has also been extended to incorporate off-chip memory access time and access rate (with a fixed memory frequency) into the performance-cost model [11].

Decreasing the memory frequency essentially increases the amount of time the CPU has to wait for memory fetches and thereby increases the CPU utilisation and decreases the GPU utilisation [19]. Therefore, there is a risk that the overall system power can still increase if the combined power reduction of the GPU and memory is not sufficient to outweigh the power increase of the CPU. Our study aims to investigate if this CPU-GPU-memory relationship can also be seen for non-3D-gaming workloads.

Memory-aware DVFS has been explored by Chaudhary et al. [6], where they use DMA and processor hardware performance counters (HPC), to predict the required bus bandwidth of the system and improve bandwidth utilisation. Nachiappan et al. [16] explore several greedy workload-aware multi-component (CPU/memory/other IPs) frequency selection policies that use application slack as a metric. Begum et al. [3] demonstrate that relaxing the bounds on performance-energy trade-off can lead to lower CPU/memory frequency tuning overhead and obtaining optimal multi-component frquencies is complex due to their interplay. In their most recent work, runtime predictive algorithms are used to tune the CPU and DRAM frequencies to stay below an inefficiency (wasted energy) budget [2]. However, note that unlike our study, several of the work discussed above (e.g. [6], [3], [2]) use non-interactive workloads. Furthermore, they do not investigate the interplay between the CPU and memory bus frequency governors.

3 Problem formulation

As outlined in Section 2, the memory controller and memory bus power consumption can be a significant contributor to the overall system power, especially in memory-bound applications. Newer mobile chipsets and kernel drivers are now equipped with functionality to control the memory bus frequency. Even though energy-efficient CPU-GPU frequency governing has been extensively explored, the impact of memory bus frequency scaling has not been investigated in detail. Whilst work such as in [19] and [11] find memory DVFS unattractive for saving power due to excessive CPU stalling, recent work in [2] and [6] demonstrates the benefits of controlling the memory frequency. Therefore, this study aims to answer the following research questions:

- Does reducing the memory frequency help to reduce the power consumption of mobile macro-workloads ? If so, is there a performance/QoS penalty ?
- Is there a relationship between the CPU and memory frequencies ?

4 Experimental design

4.1 Experimental platform

All measurements were conducted on a Samsung Galaxy S4 (i9500, Android 4.2.2) device with an Exynos 5410 chipset (8-core/2-clusters: 4x1.6GHz Cortex-A15 & 4x1.2GHz Cortex-A7). It has 2GB RAM (dual-channel 800 MHz LPDDR3 - 12.8 GB/s), a PowerVR SGX544 MP3 GPU (532MHz max GPU clock) and a 1080p WQXGA display (brightness fixed at 33%) . Dedicated image signal processors (ISP) and hardware video codec IPs are also included in the SoC.

The Monsoon power monitor device [15] was used to measure total device power consumption, with a 5KHz sampling rate. An android native program (written in C) was developed to collect periodic runtime CPU/GPU/memory performance statistics at 200ms intervals. Monitoring overhead was less than 1% of the total system load. USB-charging interferes with power measurement, therefore Wifi-based android device bridge (ADB) connection was used to control the device. Internet traffic and unnecessary background services were disabled. The *RepetiTouch* application was used to record/re-play touch events to reliably reproduce the workload against different treatments. Runtime framerate was captured using the `dumpsys` system command (periodically called only once per second), as measured by `SurfaceFlinger` service.

4.1.1 Measuring CPU/GPU utilisation and frequency

The *ondemand* [17] CPU governor is used for all primary experiments; 17 distinct CPU frequency levels are available. The `proc/stat` file system was read to calculate per core and total CPU utilisation (i.e. busy time/total time). A vendor-specific GPU frequency governor is enabled, which uses the GPU utilisation to select an appropriate frequency from 5 different levels. The runtime GPU utilisation and frequency is read from the `sysfs` interface.

4.1.2 Measuring and controlling the memory bus

The Exynos 5410 chipset has two memory bus groups and associated voltage regulators and vendor-specific drivers to control the MIF/INT bus frequencies. The MIF bus regulator controls the power to the bus connected to the main memory interfaces. The INT bus regulator provides power to the buses connected to the various internal IPs (e.g. signal processors, hardware accelerators, display controller etc.). The drivers use PPMUs (Platform Performance Monitoring Units) to monitor system load/usage. The behaviour of the default memory bus governor is similar to the conservative governor where it increases/decreases the bus frequency based on the utilisation. The max/min frequency can be adjusted at runtime (Listing 1), to fix the bus frequency at a specific level. There are 4 MIF bus frequency levels (max. 800MHz, min. 100MHz) and 11 INT bus frequency levels (max. 800MHz, min. 50MHz) provided. A user-space interface provided by the vendor, can be used to read the bus bandwidth and utilisation as shown in Listing 1. In 2011, Samsung introduced the above device frequency governing functionality into the mainline Linux kernel [10].

Listing 1: Exynos5410 memory bus frequency control and monitoring

```
* Enable kernel compilation flags:
CONFIG_SAMSUNG_NOCP_MONITOR=y
CONFIG_SAMSUNG_BW_MONITOR=y
* Enable and view bus bandwidth monitor:
echo 1 2 > /sys/devices/platform/exynos5-
    ↪ busfreq-int/devfreq/exynos5-busfreq-int/
    ↪ bw_monitor
cat /sys/kernel/debug/bw_monitor
* Set MIF/INT frequency:
echo 800000 > /sys/devices/platform/exynos5-
    ↪ busfreq-[mif/int]/devfreq/exynos5-
    ↪ busfreq-[mif/int]/max_freq
echo 800000 > /sys/devices/platform/exynos5-
    ↪ busfreq-[mif/int]/devfreq/exynos5-
    ↪ busfreq-[mif/int]/min_freq
```

4.2 Workloads

We primarily measure user-centric *macro-workloads* as shown in Table 1, that represent popular real-world smartphone use-cases and applications (e.g. web browsing, typing on the instant messaging application, scrolling on newsfeed or social media timeline etc.) [8] [20]. Several background workloads (e.g. *ffmpeg0, music0, ftp0*) are also chosen to analyse common non-GPU/non-interactive workloads.

The CPU and GPU share the system memory bandwidth. Therefore, we also use *micro-workloads* in this work (Table 1-bottom), to primarily investigate the CPU-memory frequency scaling interplay without interference from the GPU or other on-chip hardware components. These micro-workloads do not use the GPU and been designed to independently stress the CPU/memory subsystems [13]. Note also that the micro-workloads have different runtimes, with *micro2* and *micro0* having the shortest and longest execution time respectively.

4.3 Scenarios

The macro-workloads are run at different MIF/INT bus frequency levels; however, certain SoC components require a minimum MIF/INT frequency level for stable operation (e.g. display: MIF>200MHz, camera ISPs: MIF>=800MHz, INT >= 600MHz). For brevity, the MIF/INT frequencies are denoted as a frequency pair (e.g {800MHz, 800MHz}). Each macro-workload was tested under at least 3 distinct MIF/INT frequency levels as well as under the *default* MIF/INT frequency governor. In all macro-workloads, the CPU and GPU frequency governors were set to ondemand [17] and the vendor-specific implementation respectively.

The micro-workloads, were tested under the frequency settings as shown in Table 2, to inspect the CPU frequency scaling as the MIF/INT frequencies are changed. In *randMIF* and *randINT* the MIF/INT frequencies are independently varied (randomly selected from 800/400/200 MHz) every 0.5 seconds and in *randMem* they are jointly varied.

4.4 Metrics

For the macro-workloads, we measure the power consumption, the QoS (frame rate) of foreground applications and latency of background applications. QoS metric such as frame rate and latency gives us an indication of responsiveness. We intend to analyse the impact on power, QoS and latency by varying the MIF/INT bus frequencies.

System-level statistics such as memory bus saturation, bus bandwidth, dynamic CPU/GPU/MIF/INT frequency level and CPU/GPU utilisation are gathered. Memory bus saturation (i.e. utilisation) and CPU/GPU utilisation is calculated as the ratio of the busy clock cycles over the total clock cycles in a sampling period. The CPU utilisation metric represents the average utilisation across all CPU cores. There exists an interdependency between the CPU/GPU utilisation and the respective frequencies assigned by the frequency governors. Pathania et al. [19] introduces a unified CPU/GPU *cost* metric (cost = frequency × utilisation), to understand the overall work done by the CPU/GPU. We measure and calculate the CPU and GPU cost similar to [19], as well as the *memory cost* = memory util.×MIF-freq.×INT-freq.

For the micro-workloads we measure system-level performance metrics similar to the macro-workloads. We also measure the mean and sum of all CPU frequencies and the number of CPU frequency transitions, over the workload runtime. The sampling rate of measurements was increased (50ms interval) to improve accuracy.

5 Measurement study observations

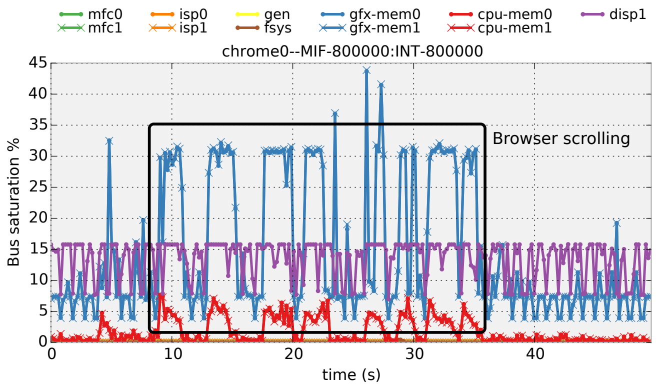
5.1 Macro-workload resource usage

The IP-level memory bandwidth is shown in Fig.1a. CPU/GPU related bus saturation (utilisation) spikes during browser-scrolling interactivity (*chrome0*). For interactive workloads, graphics (gfx) and display controller (disp) requires higher average memory bus bandwidth than the CPU. In Fig.1b we can see that the other IPs (e.g. MFC - hardware media codecs, ISP - hardware signal processors) can also significantly contribute to the total memory bandwidth, for applications such as video recording (*camera1*). Due to space-constraints, only two macro-workloads are presented in Fig.1. Other macro/micro workload system performance measurements are made available online [14].

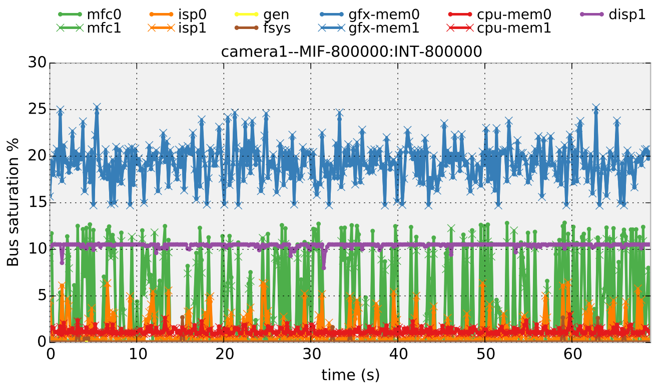
The resource usage for all macro-workloads using the default frequency governors are shown in Fig.2. *ffmpeg0, game0* and *camera1* have the highest average CPU, GPU and memory usage respectively. The measurements indicate that non-interactive tasks (e.g. background/idle/suspended-state tasks and *vlcplayer0*) have lower GPU/memory usage variation.

Table 1: Experimental workloads - macro and micro

User-centric smartphone macro-benchmarks	
idle0	Device in suspended state - No apps running, display off
idle1	Device in idle state - No apps running, display on
launcher0	Swipe left (7 times) on default home screens containing widgets and icons
ffmpeg0	Software decoding of 720p video, without video output using ffmpeg (7500 frames), display off
vlcplayer0	Video playback 720p/25fps (hardware decoding), 1 min
line0	LINE App. Instant messaging (normal typing speed)
line1	LINE App. Instant messaging (very slow typing followed by very fast typing speed)
line2	LINE App. Instant messaging (open photo album, select image and send to contact)
facebook0	Facebook Mobile app - Swipe up/down on timeline
facebook1	Facebook Mobile app - Open photo album, swipe left/right
camera0	Default Camera app. Tap to focus, take picture
camera1	Default Camera app. Record video 1080p/30fps 1 min
music0	Background (display off) audio playback using vlcplayer (44.1KHz, 128kbps), 1 min
ftp0	Background (display off) FTP download (20MB x 10). Rep. of downloading automatic software updates.
chrome0	Chrome mobile browser - bbc.com/news : swipe up/down
game0	3D game - Asphalt 8 car racing (game loading + 1 min gameplay)
Micro-benchmarks (multi-threaded) taken from [13]	
micro0	Serial and random memory read/write tests at increasing data sizes and increasing thread counts.
micro1	Designed to read data from RAM in bursts
micro2	The android port of Dhrystone integer benchmark (each thread executing copies of the same program)
micro3	Floating point add, multiply arithmetic operations (2, 32 operations per input word size)



(a)



(b)

Figure 1: Example of bus saturation/utilisation profile for two application workloads (a) Web browsing (*chrome0*) (b) Video recording (*camera1*). *disp*=Display Controller, *gfx*=Graphics, *mfc*=Hardware media codecs, *isp*=Hardware signal processors

Table 2: Micro-workload CPU, MIF/INT frequency settings (OD:ondemand, RND: random)

Name	**CPU (GHz)**	**MIF (MHz)**	**INT (MHz)**
default	OD	def. governor	def. governor
fixedAll	1.4	400	160
400-600	OD	400	160
800-800	OD	800	800
randMIF	OD	RND(800,400,200)	160
randINT	OD	800	RND(800,400,200)
randMem	OD	RND(800,400,200)	RND(800,400,200)

Note that non-gaming activities such as instant-messaging *(line2)* or web-browsing *(chrome0)* can have high GPU utilisation as well. Similarly, non-gaming workloads (e.g. *facebook1* and *chrome0*) can have higher peak memory costs than the 3D applications. Applications such as *camera1*, can show relatively higher memory usage than CPU/GPU usage, due to dedicated IPs generating memory traffic. Memory utilisation patterns for the same application can also vary depending on the use-case (e.g. *facebook0* and *facebook1*), due to the behaviour of the default memory governor.

At very high peak utilisation levels (e.g. *chrome0* and *facebook1*), memory bus contention and congestion occurs, resulting in memory transactions being buffered/dropped; this in turn can negatively impact user-experience. On certain workloads, we observed the memory bus frequency conservatively increased to the maximum, even though at that level the peak utilisation is low (e.g. *vlcplayer0*). However, due to the limited number of frequency levels, under-utilisation is unavoidable.

5.2 Impact of memory bus frequency on resource utilisation

Fig.3 shows the CPU and GPU cost with respect to different MIF/INT bus frequency levels. Overall, the general trend is that for interactive applications (e.g. Facebook, LINE instant messaging, Chrome web browsing), as the MIF/INT

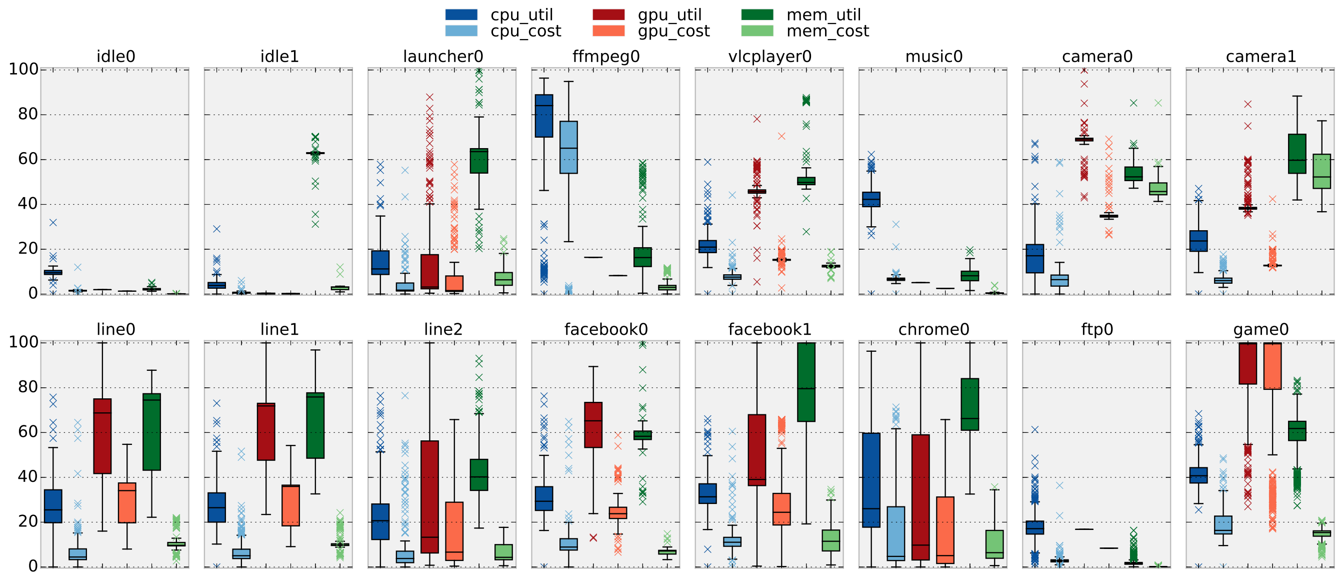


Figure 2: Macro-workload - CPU/GPU/memory utilisation and cost (MIF:default, INT:default)

bus frequency is decreased, the CPU/GPU cost is increased. This is mainly due to CPU/GPU governors increasing their frequency level to overcome the higher utilisation levels due to stalling (waiting for data to arrive). In our 3D game measurements (i.e. *game0*) the GPU cost does not change significantly as the MIF frequency is decreased by half, even though the CPU cost increases. This indicates that perhaps certain resource management algorithms which differentiate between CPU-GPU bound phases (e.g. [7]), can exploit memory frequency reduction appropriately to further reduce power consumption. Note that certain applications such as *camera0* and *camera1* have sub-IP specific minimum MIF/INT frequency levels required to operate.

In lightweight background applications (e.g. *ftp0*, *idle0*, *idle1*), a similar trend to interactive/foreground applications is not seen. In *music0*, we can only see a significant CPU cost increase only for the lowest MIF/INT frequency level. In several workloads a sharp rise in CPU and GPU cost is seen between MIF values 400 and 200 MHz, which indicates that system performance is more susceptible to MIF frequency changes than INT frequency changes. *ffmpeg0* shows a high CPU cost for the default case. This is mainly because the default CPU governor stays mostly above 1GHz and incidentally the MIF frequency governor fluctuates between 800MHz and 200MHz.

5.3 Impact of memory bus frequency on power consumption and Quality-of-Service

The power consumption for each macro-workload scenario is shown in Fig.4. The power distributions correlate well with the CPU cost distributions (Fig.2). For example, *launcher0* has a long tail power distribution because of swiping related CPU frequency spikes. High power consumption can be due to very high CPU usage (*ffmpeg0*) or combined high levels of memory and GPU costs *camera0, game0*. Background/non-interactive tasks have the lowest power consumption and variation due to low resource usage.

As shown in Fig.5 and Fig.6, the total system power con-

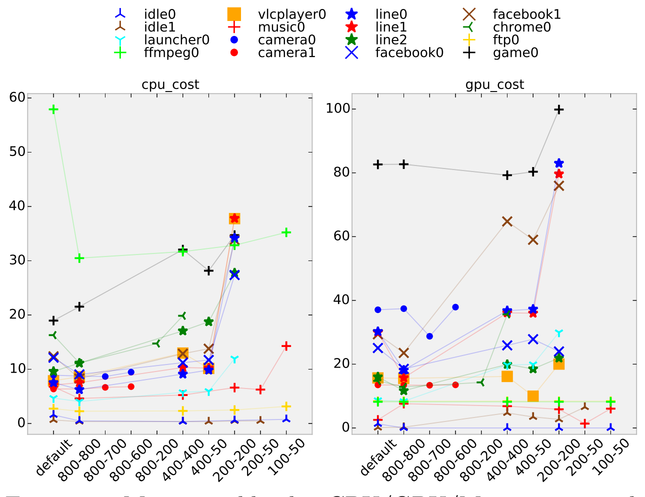


Figure 3: Macro-workload - CPU/GPU/Mem. cost with respect to MIF/INT bus frequency change

sumption and the QoS varies when the memory bus frequency is changed. In these visualisations, the bar plots are sorted and overlaid on top of each other. For example in Fig.5, in the *idle0* scenario, 800-800 has the highest normalised power level and 100-50 has the lowest.

In *line0*, up to 40% power consumption difference between high-low memory frequencies can be seen; indicating that memory frequency scaling does impact power consumption. MIF bus frequency changes cause larger power consumption differences than INT bus frequency changes. The lowest MIF/INT frequencies can adversely affect both the power consumption (e.g. *line0*, *line1* and *line2*) and QoS levels, due to corresponding CPU/GPU utilisation increase.

The default memory governor can at times be conservative, leading to unnecessarily high memory frequencies and power consumption. For the background/non-interactive applications, up to 5% mean power reduction can be obtained with

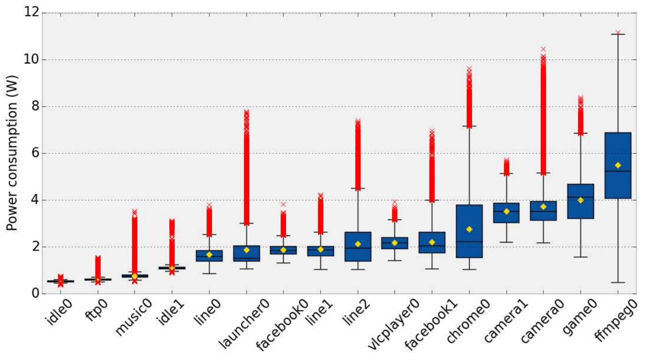


Figure 4: Macro-workload power consumption distribution (MIF:default, INT:default)

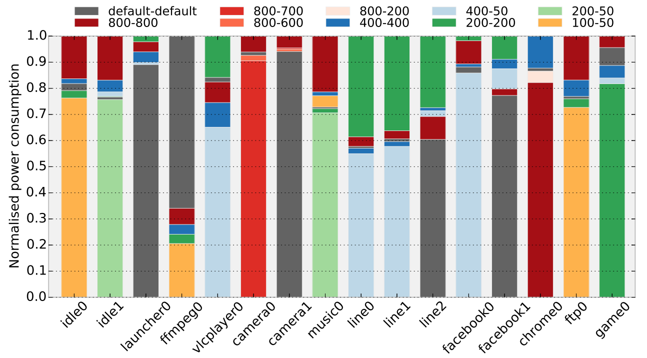


Figure 5: Macro-workload normalised mean power consumption, for all tested MIF/INT frequencies. Normalised within each scenario

minimal increase in application response-time, when using a fixed lower memory bus frequency setting over the default governor. Similarly, foreground applications such as *vlcplayer0* and *game0* showed 10-20% power reduction over the default governor at a QoS reduction of approx. 5%. For *chrome0*, the MIF/INT frequency at (400MHz, 200MHz) has comparable power consumption to the *default* case, but gave a 30% QoS improvement. These results indicate that the relationship between memory bus frequency, power consumption and QoS are not always linear.

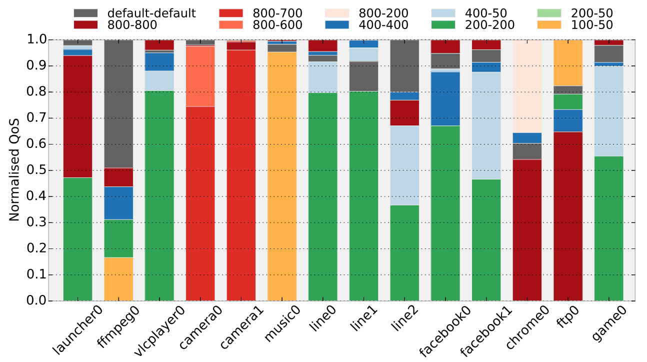


Figure 6: Macro-workload normalised mean QoS levels, for all tested MIF/INT frequencies. Normalised within each scenario

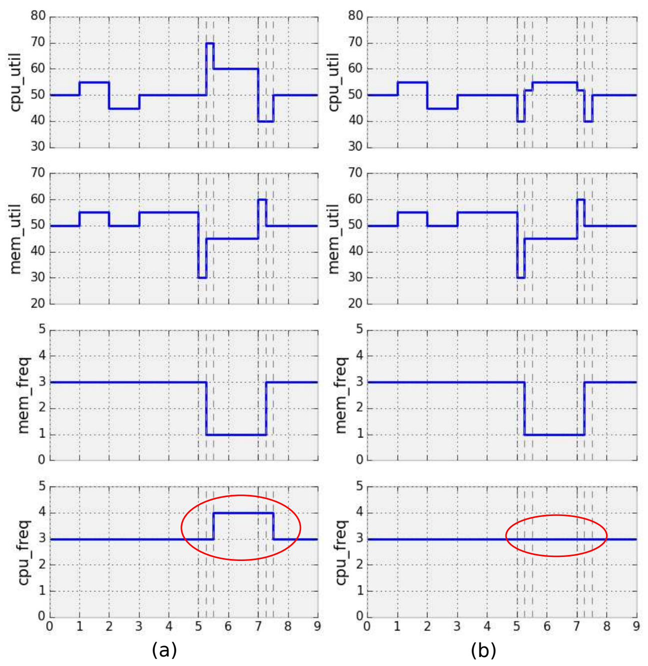


Figure 7: Example illustration of CPU frequency interference due to memory frequency scaling (x-axis denotes time)

5.4 Example of CPU and memory bus frequency governor inter-dependency

The example illustrations shown in Fig.7, are used to describe how memory frequency scaling can affect the decisions of the CPU governor. We assume the governors are not synchronised and unaware of each others decisions. Fig.7(a) first illustrates a case where CPU frequency is increased unnecessarily and Fig.7(b) illustrates an instance where a potential CPU frequency reduction opportunity was missed.

In - Fig.7(a), the memory utilisation goes low to 30 at $t = 5.0$ (t denotes time). At $t = 5.25$, the memory governor lowers the memory frequency to level 1, leading to a decrease in memory cost and a rise in CPU utilisation (due to processor stalling). At $t = 5.50$ the CPU governor samples the CPU utilisation and increases the CPU frequency to address the utilisation increase. In this situation, we assume the CPU load does not change significantly and if the memory frequency transition did not occur, the CPU frequency would not have changed.

The second scenario - Fig.7(b) is similar to the first, except in this instance, it is assumed that the CPU utilisation also goes low at the same instant ($t = 5.0$), and if the memory frequency governor did not interfere, the CPU frequency would have gone low to level 2 (between $t = 5.5 - 7.5$). Here, the system could have saved power by dropping the CPU frequency, but the CPU governors' decision was again interfered by the memory frequency transition.

5.5 Micro-workload analysis

The micro-workloads (Table 1) and treatments shown in Table 2 were used to further explore the memory bus and CPU frequency scaling dependency. As shown in Fig.8, *micro2* has a large variation in CPU utilisation and *micro0* and *micro1* have the highest and largest variation in memory bus utilisation. *micro0* is mostly memory-bound, whilst the in-

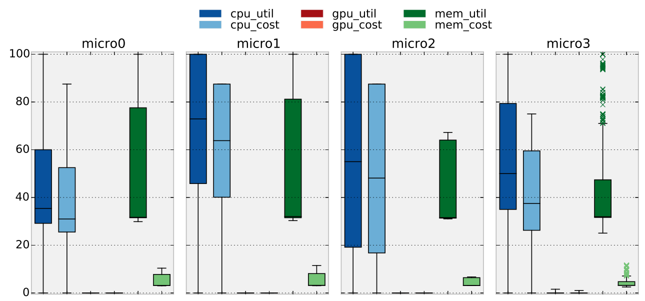


Figure 8: Micro-workload CPU/memory utilisation and cost (fixedAll test case)

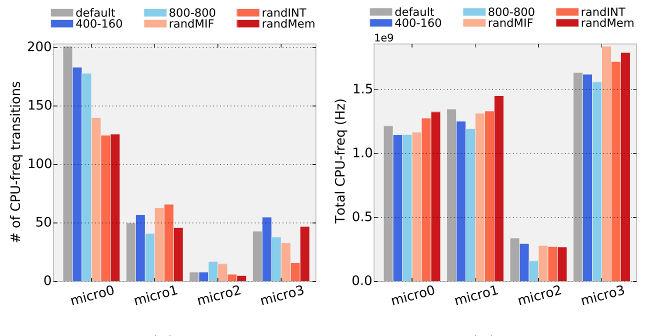


Figure 9: Micro-workload tests (a) Number of total CPU frequency transitions (b) Summation of CPU frequency over test runtime

verse is true for *micro3*. None of the micro-workloads have GPU load.

In the micro-workload experiment, we measure the number of CPU frequency transitions and the sum of all frequencies sampled, for each of the treatments (Table 2). If the metrics significantly differ between the different test cases, then it indicates that the MIF/INT frequency scaling has affected the decision of the CPU frequency scaling governor.

Fig.9a shows the number of CPU frequency transitions of the micro-workloads. We observe that depending on the treatment, the CPU frequency transitions can vary, which indicates that the memory frequency transitions have indeed interfered with the decisions of the CPU frequency governor. In all cases, the *default* treatment is different to the other treatments. For example, in *micro0*, where randomly changing the memory frequency (i.e. *rand**), resulted in over 25-30% less CPU transitions than the *default* case. Non-uniform CPU/memory utilisations can result in lower number of CPU frequency transitions than the *default* treatment (e.g. *randMIF*). In *micro1*, *randINT* produces slightly higher CPU frequency transitions than *randMIF*, which indicates that under certain workloads, the INT bus frequency can affect the CPU frequency governor more than the MIF bus frequency.

The summation of all CPU frequencies for the workload runtime is displayed in Fig.9b. We can see that high number of CPU frequency transitions do not necessarily result in a higher total CPU frequency (e.g. *micro0*), and vice-versa. In the case of *randINT*, lower number of transitions can result in higher total CPU frequency than *default* (in *micro0* and *micro3*). Fixing the memory bus frequency to the highest level can often reduce the total CPU frequency (e.g. *800-800* treatment). Furthermore, in all cases except *micro3*, randomly scaling the MIF bus frequency (i.e. *randMIF*) can result in lower total CPU frequency than the *default* memory frequency governor; this indicates inefficiencies with the default frequency governor.

6 Summary of key observations and system design suggestions

We now summarise the key observations made during the measurement-based study and present several resource management design suggestions drawn from the observations. It is hoped these suggestions would be beneficial for system designers to alleviate unnecessary energy dissipation.

- Usage-scenarios and QoS targets should be integrated to CPU frequency governor heuristics, to efficiently balance power consumption and quality of experience. Governors that *solely* focus on utilisation, can unnecessarily increase the frequency, even for non-performance critical, although CPU-bound, background applications (e.g. *ffmpeg0*).
- Lower memory bus frequencies can be exploited under background applications, as well as during idle/suspended states to save power consumption. Certain foreground applications such as video playback and photography can also benefit from lower memory bus frequencies with minimal impact to QoS.
- Workload prediction algorithms (e.g. [19] [11]) should also consider the memory bus traffic and resource contention generated by dedicated hardware IPs (e.g. signal processing IPs). For example, certain high memory bandwidth applications can have relatively lower CPU/GPU usage (e.g. camera recording).
- Symmetrically scaling different system-on-chip internal bus frequencies (e.g. MIF, INT) should be avoided, as their load and performance/power-saving impact can also be different. The ability to finely tune different bus frequencies can also increase the bus frequency governor's flexibility.
- Certain foreground application QoS issues can be addressed by setting the memory frequency at the highest, to handle bursty memory traffic (e.g. web browser scrolling, swiping through images). Furthermore, dynamic governor sampling periods (based on rate of memory transactions), can balance accuracy and monitoring overhead.
- In-depth statistical analysis of the workload and CPU/GPU/memory frequency governors can assist load prediction heuristics. For example the impact on memory frequency scaling on CPU frequency scaling can differ based on the uniformity of the memory bus utilisation.
- Due to the lack of *co-operation* and *synchronisation* between the CPU and memory bus frequency governors, the memory bus frequency transitions can interfere the CPU frequency governor's decisions (Section 5.4). Therefore, a *cooperative/holistic frequency governing framework*, which can assign an appropriate frequency setting for all the different components/resources on the SoC is required. E.g. ensure that de-

creasing memory bus frequency does not change the CPU frequency. Such an interconnected governor should take into account the performance/QoS requirements, state of computing resources and also the communication subsystems (i.e. bus, memory controllers etc.) when making a frequency selection. A unified governor can have more control and flexibility on the performance of the different on-chip resources to avoid unnecessary power consumption.

7 Conclusion and future work

This work presented on-chip resource usage and power consumption measurements of popular user-centric smartphone workloads. In particular, we analysed the impact of memory frequency scaling on power consumption and its interplay with the CPU and GPU. We presented the CPU, GPU and memory cost of user-centric macro-workloads under default system settings. Our measurements indicated that under certain application types (e.g. idle states, background applications, hardware-accelerated video playback, 3D gaming) fixing the memory bus frequency at lower frequencies can provide higher power savings (5-20%) with marginal QoS degradation, compared to the default memory frequency governor. We illustrated two cases where the memory bus frequency governor interferes with the CPU frequency governor, to either unnecessarily increase power consumption or fail to save power. Micro-workloads were used to further demonstrate the CPU frequency governor inconsistencies caused by the memory frequency interference. Lastly, we presented several system design suggestions related to CPU/GPU/memory bus resource management drawn from our observations. As future work, we are working towards implementing an interconnected CPU-GPU-memory bus frequency governing framework which can help further reduce the power consumption of smartphones.

Acknowledgement

This work was funded by the HiPEAC 2016 collaboration grant (The FP7 HiPEAC Network of Excellence)

8 References

[1] ARM. big.LITTLE technology. https://developer.arm.com/technologies/big-little, 2013.

[2] R. Begum, M. Hempstead, G. P. Srinivasa, and G. Challen. Algorithms for CPU and DRAM DVFS under inefficiency constraints. In *Int. Conf. on Comp. Design (ICCS)*, page 161âĂŞ168, 2016.

[3] R. Begum, D. Werner, M. Hempstead, G. Prasad, and G. Challen. Energy-performance trade-offs on energy-constrained devices with multi-component DVFS. In *Int. Symp. on Workload Characterization (IISWC)*, pages 34–43, 2015.

[4] A. Carroll and G. Heiser. The systems hacker's guide to the galaxy energy usage in a modern smartphone. In *Asia-Pacific Workshop on Systems (APSYS)*, pages 5–12, 2013.

[5] A. Carroll, G. Heiser, et al. An analysis of power consumption in a smartphone. In *USENIX*, pages 1–14, 2010.

[6] N. Chaudhary, T. Pallavi, et al. Bus bandwidth monitoring, prediction and control. In *Conf. on Advances in Comp., Comm. and Informatics (ICACCI)*, pages 1152–1158, 2015.

[7] W.-M. Chen, S.-W. Cheng, P.-C. Hsiu, and T.-W. Kuo. A user-centric CPU-GPU governing framework for 3D games on mobile devices. In *IEEE/ACM Conf. on Computer-Aided Design (ICCAD)*, pages 224–231, 2015.

[8] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C. J. Wu. A study of mobile device utilization. In *Symp. on Perf. Analysis of Sys. and Software (ISPASS)*, pages 225–234, 2015.

[9] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-system analysis and characterization of interactive smartphone applications. In *Int. Symp. on Workload Characterization (IISWC)*, pages 81–90, 2011.

[10] M. Ham. Introduce devfreq: generic DVFS framework with device-specific opps. https://lwn.net/Articles/445044/, May 2011.

[11] C. Y. Hsieh, J. G. Park, N. Dutt, and S. S. Lim. Memory-aware cooperative CPU-GPU DVFS governor for mobile games. In *IEEE Symp. on Embedded Sys. For Real-time Multimedia (ESTIMedia)*, pages 1–8, 2015.

[12] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Int. conf. on Power aware Comp. and Sys.*, pages 1–5, 2010.

[13] R. Longbottom. Android benchmarks by Roy Longbottom. http://www.roylongbottom.org.uk/android%20benchmarks.htm, 2017.

[14] H. Mendis. System performance measurement data. https://goo.gl/mQBrGS.

[15] Monsoon. Power monitor. https://www.msoon.com/LabEquipment/PowerMonitor/, 2017.

[16] N. C. Nachiappan, P. Yedlapalli, N. Soundararajan, A. Sivasubramaniam, M. T. Kandemir, R. Iyer, and C. R. Das. Domain knowledge based energy management in handhelds. In *Int. Symp. on High Perf Comp. Arch. (HPCA)*, pages 150–160, 2015.

[17] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Linux Symposium*, pages 215–230, 2006.

[18] D. Pandiyan and C. J. Wu. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *Int. Symp. on Workload Characterization (IISWC)*, pages 171–180, 2014.

[19] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated CPU-GPU power management for 3D mobile games. In *Design Automation Conf. (DAC)*, pages 1–6, 2014.

[20] S. Patil, Y. Kim, K. Korgaonkar, I. Awwal, and T. S. Rosing. Characterization of userâĂŹs behavior variations for design of replayable mobile workloads. In *Mobile Comp, Apps., and Services Conf.*, pages 51–70, 2015.