Center for Embedded and Cyber-Physical Systems
University of California, Irvine

Cooperative CPU-GPU Frequency Capping (Co-Cap) for Energy Efficient Mobile Gaming

Jurn-Gyu Park, Chen-Ying Hsieh, Nikil Dutt, Sung-Soo Lim

Center for Embedded and Cyber-Physical Systems
University of California, Irvine
Irvine, CA 92697-2620, USA

{jurngyup, chenyinh, dutt}@ics.uci.edu, sslim@kookmin.ac.kr

CECS Technical Report 15-05
Dec 10, 2015

Cooperative CPU-GPU Frequency Capping (Co-Cap) for Energy Efficient Mobile Gaming

Jurn-Gyu Park, Chen-Ying Hsieh, Nikil Dutt
School of Information and Computer Science
University of California, Irvine, CA, USA
jurngyup, chenyinh, dutt@ics.uci.edu

Sung-Soo Lim
School of Computer Science
Kookmin University, Seoul, South Korea
sslim@kookmin.ac.kr

ABSTRACT

Mobile platforms are increasingly using Heterogeneous MultiProcessor Systems-on-Chip (HMPSoCs) with differentiated processing cores and GPUs to achieve high performance for graphics-intensive applications such as mobile games. Traditionally, separate CPU and GPU governors are deployed in order to achieve energy efficiency through Dynamic Voltage Frequency Scaling (DVFS), but miss opportunities for further energy savings through coordinated system-level application of DVFS. We present Co-Cap, a cooperative CPU-GPU DVFS strategy that orchestrates energy-efficient CPU and GPU DVFS through coordinated CPU and GPU frequency capping to avoid frequency over-provisioning while maintaining desired performance. Unlike traditional approaches that target a narrow set of mobile games, our Co-Cap approach is applicable across a wide range of mobile games. Our methodology deploys a training phase followed by a deployment phase, allowing not only deployment across a wide range of mobile games with varying graphics workloads, but also across new mobile architectural platforms. Our experimental results across a large set of over 70 mobile games show that Co-Cap improves energy per frame by 10.6% and 10.0% (23.1% and 19.1% in CPU dominant applications) on average and achieves minimal frames per second (FPS) loss by 0.5% and 0.7% (1.3% and 1.7% in CPU dominant applications) on average in training- and deployment sets, respectively, compared to the default CPU and GPU governors, with negligible overhead in execution time and power consumption on the ODROID-XU3 platform.

Categories and Subject Descriptors

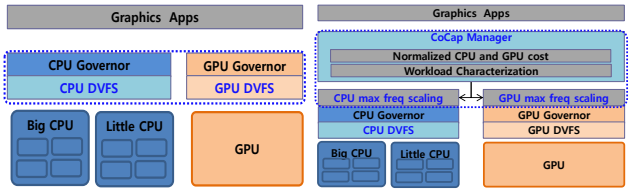
C.1.4 [**Parallel Architectures**]: Mobile processors; D.4.7 [**Organization and Design**]: Real-time systems and embedded systems

Keywords

Power management policies, Dynamic Voltage and Frequency Scaling, Heterogeneous Multi-core platform

1. INTRODUCTION

Mobile platforms are increasingly demanding high performance and longer battery life at the same time resulting in the move towards Heterogeneous MultiProcessor Systems-on-Chip (HMPSoC) for high performance, coupled with various software governors to achieve energy efficiency through DVFS. For instance, the Exynos 5 (5422) HMPSoC integrates ARM's big.LITTLE Octa multi-core CPU and ARM's Mali-T628 MP6 GPU on the same chip. To achieve energy efficiency, traditionally independent CPU and GPU DVFS power management techniques are common for commercial platforms, and some recent efforts have proposed integrated CPU-GPU governors [11] [10] [7] for energy efficiency, but have focused on a small set of mobile games that have specific workload characteristics. (Figure 1(a)).



(a) General Mobile System (b) Proposed Co-Cap System

Figure 1: System Comparison

Traditional DVFS approaches suffer from two drawbacks: 1) they are not able to achieve energy efficiency across a wide range of mobile games that exhibit varying CPU and GPU workloads, and 2) these approaches are customized for specific mobile platforms. To address both of these issues, in this paper we present Co-Cap (Figure 1.(b)), a methodology that achieves energy efficient DVFS across a wide range of mobile games, and that is also easily portable to newer mobile architectural platforms.

Device	Release	SoC	CPU	Max. F. (Ghz)	GPU	Max. F. (Mhz)
Nexus 4	Feb. 12	Snapdragon S4	Quad	1.5	Adreno 320	400
Galaxy S4	Mar. 13	Exynos 541x	Octa b.L	1.2/1.6	SGX 544	480
Nexus 5	Oct. 13	Snapdragon 800	Quad	2.3	Adreno 330	450
Galaxy S5	Mar. 14	Exynos 542x	Octa b.L	1.3/1.9	Mali-T628	543
Nexus 6	Oct. 14	Snapdragon 805	Quad	2.7	Adreno 420	600
Galaxy S6	Mar. 15	Exynos 7420	Octa b.L	1.5/2.1	Mali-T760MP8	772

Figure 2: Mobile Platform Trends.

Mobile platforms face the dual challenges of rapidly changing (heterogeneous) architectures, and the continuing emergence of a plethora of mobile games. For instance, Figure 2 shows the progression of the Nexus and Samsung Galaxy series platform architecture, demonstrating the rapid changes in (heterogeneous) processor, and GPU configurations. New energy efficient DVFS governors have to be developed rapidly

for each new architectural family. On the other hand, at the application level, we are seeing a range of mobile games that exhibit diverse CPU and GPU workloads, that require widely different strategies for achieving energy efficiency while delivering acceptable game performance. For instance, Figure 3 shows that each game application can be located in a specific quadrant of a CPU/GPU workload intensity matrix. Angry Birds has very low CPU and GPU workloads, while GFX benchmark is GPU-bound and Jetski Race is very CPU-bound; some applications such as GPUbench are more balanced in terms of CPU and GPU workloads. For these four different types of graphics workloads, we define them as No CPU-GPU dominant, CPU dominant, GPU dominant, CPU-GPU dominant workloads respectively.

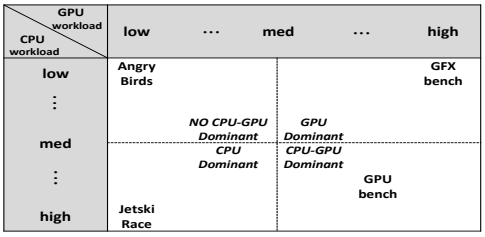


Figure 3: Different types of CPU/GPU Workload.

The traditional approach of separate DVFS governors for CPU and GPU are unable to achieve good energy efficiency while delivering acceptable performance across this wide range of mobile game applications. Even recent efforts [11] [10] [7] using integrated CPU-GPU governors are only targeted towards specific classes of mobile games. Furthermore, these approaches do not provide a methodology for easily porting their strategies across newer architectural platforms that appear in rapid succession (e.g., Nexus and Galaxy series in Figure 2).

To address these dual issues, this paper presents Co-Cap, an energy-efficient cooperative CPU-GPU frequency capping strategy for mobile games. Our paper makes the following specific contributions:

- We present a methodology for cooperative CPU-GPU frequency capping to achieve energy efficiency for a diverse set of mobile games
- Our capping strategy avoids unnecessarily higher frequency of CPU and GPU considering both FPS and per-frame energy saving on top of the default CPU and GPU governors
- We present characterization of diverse mobile graphics gaming workloads to enable efficient dynamic application of frequency capping
- We demonstrate efficacy of Co-Cap across 70 real mobile graphics applications, using a representative training and deployment set achieving over 12% improvement in average energy efficiency with minimal loss in performance

2. RELATED WORK

DVFS is a traditional energy conservation strategy that has been applied for both CPUs and GPUs in desktop and mobile space. For graphics-intensive applications such as mobile games, typically frames per second (FPS) and energy per frame (or FPS per watt) are used as performance and energy consumption metrics, respectively. Some efforts have begun analyzing graphics-intensive rendering applications (e.g., 3D games) in mobile devices: Gu *et al.* [5], [4] proposed CPU graphics rendering workload characterization and CPU DVFS for 3D Games, under the assumption that mobile devices such as PDAs and mobile phones do not have integrated mobile GPUs. With the emergence of high performance mobile GPUs, Dietrich *et al.* [2], [3] introduced CPU DVFS for mobile graphics rendering as an extension of [5], [4]; however these efforts didn't focus on GPU or cooperative CPU-GPU DVFS schemes, but addressed CPU DVFS for mobile GPU graphics rendering.

Park *et al.* [9] developed micro-benchmarks for mobile graphics workload characterization, analyzed the results of benchmarks, and introduced several opportunities for improved DVFS design of mobile GPU graphics rendering, but did not present specific DVFS strategies. Pathania *et al.* [11] proposed an integrated CPU-GPU DVFS algorithm for power management for mobile games. However, their work didn't consider quantitative evaluation for energy saving (e.g., per-frame energy or FPS per watt). Pathania *et al.* [10] also proposed a power-efficient integrated CPU-GPU DVFS strategy by developing power-performance models predicting the impact of DVFS on mobile gaming workloads , and Kadjo *et al* [7] presented a queuing model to represent the interaction between CPU, GPU, and Display; however, their work was applied to a specific set of games exhibiting a narrow range of CPU-GPU workloads and they did not show applicability across a wide range of games exhibiting diverse CPU-GPU workloads (Figure 3).

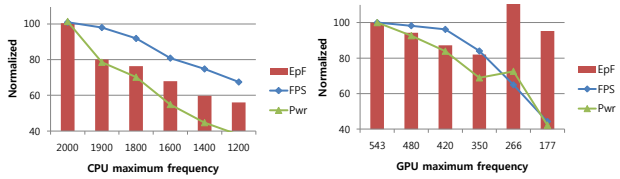
In this work, we complement shortcomings of previous efforts by proposing Co-Cap. Our work is fundamentally different from previous integrated CPU-GPU DVFS techniques in that our approach dynamically scales the maximum frequency of CPU and GPU according to the normalized CPU and GPU cost on top of the default CPU- and GPU governors. Although frequency capping for energy efficiency was initially introduced in [8], their approach was restricted to the CPU governor; in contrast to the best of our knowledge, our work is the first to introduce a coordinated CPU and GPU maximum frequency capping technique that achieves energy efficiency (lower energy per frame) across a diverse range of mobile games while delivering acceptable performance (FPS).

3. CO-CAP OVERVIEW

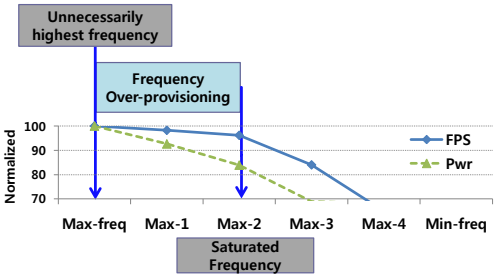
3.1 Motivation

Mobile platforms pose a challenge for simultaneous reduction of energy consumption while delivering acceptable performance across a wide range of mobile gaming applications. As motivating examples, Figure 4.(a) and 4.(b) show the normalized FPS, power consumption (Pwr), and energy per frame (EpF) on one CPU dominant (Q3Zombie Map4) and the other GPU dominant (Action Bike) game benchmarks by changing CPU and GPU maximum frequencies allowing dynamic frequency changes under standard governors implemented in Linux. Even though FPS declines little until 1900Mhz in Figure 4(a) and until 420Mhz in Figure 4(b), the power consumption is dramatically reduced compared to the FPS reduction. This shows that limiting the maximum frequencies that can be reached during dynamic frequency scal-

ing gives significant opportunity for performance and power consumption optimizations in modern CPU/GPUs. In other words, we can exploit this frequency over-provisioning to achieve energy saving up to 20% in Figure 4.(a) and up to 15% in Figure 4.(b) within little FPS (3%) decline comparing with the default governors by scaling CPU and GPU maximum frequencies. We call the frequency beyond which



(a) Q3Zombie Map4 (b) Action Bike



(c) The Saturated Frequency

Figure 4: Motivating Examples

the FPS degrades as "saturated frequency" (Figure 4(c)). By identifying these saturated frequencies after characterization of CPU- and GPU graphics workloads, we can save energy with minimal performance degradation by eliminating CPU and GPU frequency over-provisioning.

3.2 Co-Cap Methodology

Co-Cap orchestrates cooperative CPU-GPU dynamic maximum frequency capping (scaling) by avoiding frequency over-provisioning of CPU or GPU considering both performance (FPS) and per-frame energy saving on top of the default CPU- and GPU governors. Co-Cap has two phases: a training phase and a deployment phase as shown in Figure 5.

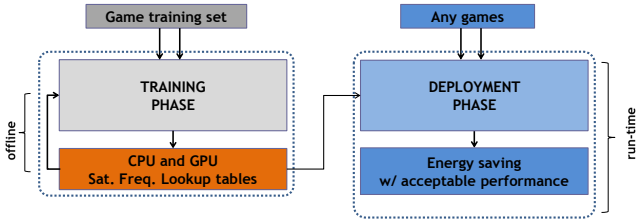


Figure 5: Co-Cap Overview.

In the training phase, we build CPU and GPU saturated frequency lookup tables offline using a training set. Then in the deployment phase, Co-Cap uses these saturated frequency lookup tables at runtime to set the appropriate CPU and GPU frequency caps based on the characteristics of the executing mobile game.

3.2.1 *Training Phase*

As shown in Figure 6, the training phase is composed of three steps: data capturing step, saturated frequency lookup table building step, and refinement step. The main objective of the training phase is to build the saturated frequency lookup tables where the maximum frequency values for both CPU and GPU can be obtained by CPU and GPU workload cost indices.

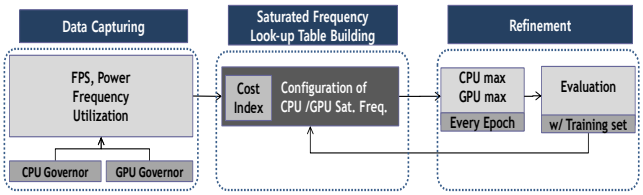


Figure 6: Co-Cap Training Phase.

Data Capturing Step: As shown in Table 1, all data including frequency and utilization can be captured dynamically from each software component of CPU and GPU governors, and power sensor driver (or using power monitor).

Table 1: Captured Data

Category	SW component	Metrics Data
CPU	CPU Governor	per-cpu Utilization, Frequency
GPU	GPU Governor	Utilization, Frequency
Power	Power Sensor Driver	CPU, GPU, DRAM
Perf.	Android and GPU Governor	FPS

The sample training set shown in Figure 7 includes mobile games that cover different quadrants of the CPU-GPU workloads shown in Figure 3. These workload variations

Normalized CPU\GPU Cost (Index)	0 ~ 100/N (0)	100/N ~ 200/N (1)	...	(N-2)x100/N ~ (N-1)x100/N (N-2)	(N-1)x100/N ~ 100 (N-1)
0 ~ 100/N (0)	App A	App B	App ...	App E	App F
100/N ~ 200/N (1)	App C	App D	App...	App G	App H
...	App ...	App ...	App ...	App ...	App ...
(N-2)x100/N~(N-1)x100/N (N-2)	App H	App I	App ...	App L	App M
(N-1)x100/N ~ 100 (N-1)	App J	App K	App ...	App N	App O

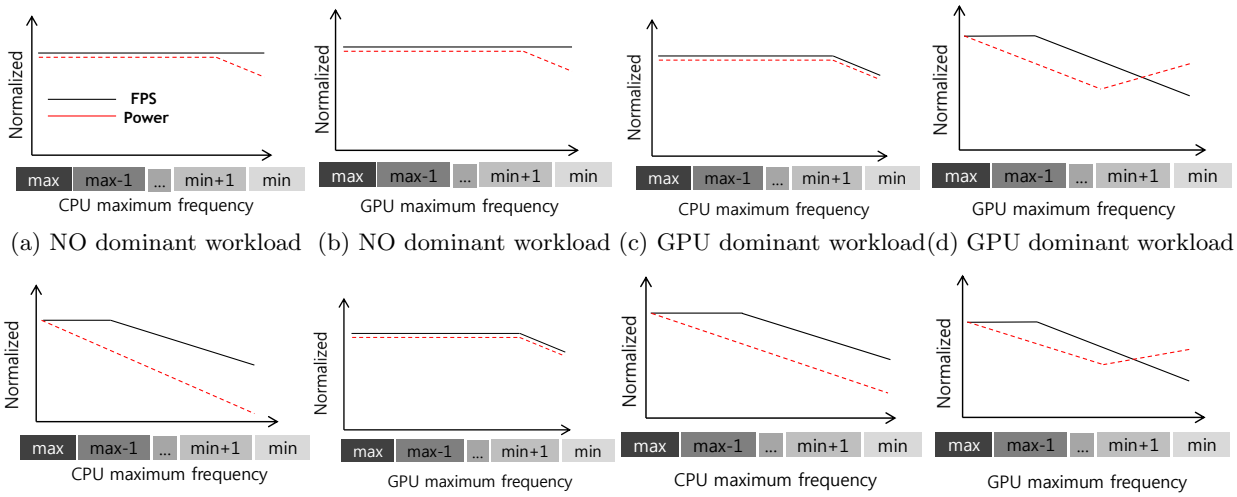
Figure 7: A Sample of the Training Set.

are typically quantified using a cost metric that is a product of the utilization and frequency [1]. Accordingly, we deploy normalized CPU- and GPU costs as shown in Equation(1), where the cost is defined as the product of the processor current average utilization and its current average frequency divided by the product of the maximum utilization and its maximum frequency.

$$Normalized\ Cost = \frac{Curr\_Util. \times Curr\_Freq.}{Max\_Util. \times Max\_Freq.} \quad (1)$$

For CPU utilization, the highest CPU utilization among CPU cores is used according to the assumption that there is usually one graphics rendering thread mainly affecting the graphics performance for most mobile graphics applications, and the utilization of the thread is mostly highest among threads.

Frequency Lookup Table Building Step: Using the normalized CPU and GPU costs, we determine CPU/GPU cost index, which is used as the indices in the saturated frequency lookup table and the representation of the range of the normalized cost. For instance, if the number of quadrants is NxN as shown in Figure 7, the normalized CPU/GPU cost from 0 to 100/N is 0. Now, let's assume that we use a new game on a new platform. By observing the CPU and GPU utilization and frequency for a specific amount of



(a) NO dominant workload (b) NO dominant workload (c) GPU dominant workload (d) GPU dominant workload

(e) CPU dominant workload (f) CPU dominant workload (g) CGU dominant workload (h) CGU dominant workload

Figure 8: Effects of CPU (or GPU) maximum frequency capping on FPS and Power.

time, the game will be located in a specific quadrant of the training set and will have the corresponding *CPU_cost_index* and *GPU_cost_index*, (and how to select an appropriate set of training games will be described in the next section).

Before describing how to configure the saturated frequency look-up tables, we need to explain the general trend of maximum frequency set-up for the four different application quadrant categories shown in Figure 3: No CPU-GPU (No) dominant, CPU dominant, GPU dominant, and CPU-GPU (CGU) dominant. Figure 8.(a) and (b) show the performance/power consumption graph trends of No-dominant workloads. FPS remains the same (i.e., the 60 maximum FPS) for all CPU and GPU frequencies, which means that the saturated frequencies of CPU and GPU are available up to the minimum frequency of CPU and GPU for energy saving without FPS reduction. Figure 8.(c) and (d) show the performance/power consumption pattern of GPU dominant workloads. For GPU frequency, FPS is flat until max-1 frequency and power consumption reduces gradually until min+1 frequency. Therefore, in order to achieve energy saving with little FPS degradation, the GPU saturated frequency can be reduced to max-1 frequency. For CPU capping frequencies, FPS and power consumption are almost similar except the minimum capping frequency, which means that the CPU saturated frequency should be higher than the minimum frequency in order to prevent FPS reduction. Figure 8.(e) and (f) show the performance/power consumption of CPU dominant workloads. Here FPS is almost similar until max-1 frequency and power consumption gradually decreases with CPU capping frequency drop, which means that the CPU saturated frequency could be available up to the max-1 frequency. For GPU capping frequencies, FPS and power consumption are almost similar except the minimum capping frequency, which means that the GPU saturated frequency should be higher than the minimum frequency in order to prevent FPS reduction. Finally in Figure 8.(g) and (h), CGU-dominant workload is the combination of CPU dominant workload and GPU dominant workload. Therefore, in CPU and GPU, max-1 frequency can be configured to the saturated frequency.

Using this characterization process, the saturated frequency of each application in the training set can be configured appropriately as shown in Equation (2):

$$\begin{aligned} &Saturated\ Frequency = max(F_{fps}, F_{lp}) \\ &where\ F_{fps} = lowest\ maximum\ frequency \\ &\qquad\qquad with\ minimal\ (<3\%) FPS degradation \\ &\qquad F_{lp} = frequency\ for\ lowest\ power\ consumption \end{aligned} \tag{2}$$

In other words, from the captured data of each application, we choose the higher frequency among the lowest maximum frequency having little FPS (up to 3%) decline and the lowest maximum frequency having lowest power consumption.

Frequency Lookup Table Refinement Step: If there is continuous maximum utilization during a certain amount of time, the saturated frequency should be scaled up dynamically in order to prevent FPS reduction. We deploy a heuristic where, if there is successive (e.g., 3 epochs) extreme CPU or GPU utilization (e.g., 100%), the saturated frequency is dynamically scaled up to the one-level higher CPU or the two-level higher GPU frequency.

In addition, since the games deployed in the training set (Figure 7) may not fully cover all CPU-GPU cost entries, we can heuristically speculate missing entries by using interpolated values of adjacent entries in the table. These speculated entries are then evaluated to ensure that Co-Cap's application of these frequency caps does not result in loss of performance through significant FPS drop.

Finally, CPU and GPU saturated frequency lookup tables can be obtained through repeated overall Co-Cap evaluation for all applications in the training set.

3.2.2 *Deployment Phase*

In the deployment phase (Figure 9), data capturing, maximum frequency setting, and evaluation steps can be executed at runtime using the CPU and GPU saturated frequency lookup tables. This ensures evaluation of applications across a wide range of games exhibiting diverse CPU and GPU workloads.

Data Capturing Step: This step is exactly the same as the training phase (all data except power consumption as shown in Table 1 can be captured dynamically in every epoch).

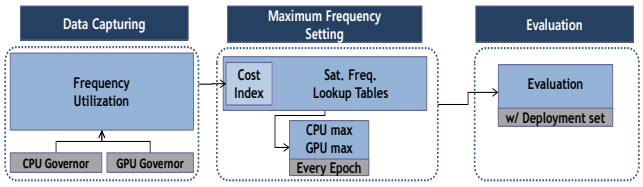


Figure 9: Co-Cap Deployment Phase.

Maximum Frequency Setting Step: In this step, the corresponding CPU and GPU maximum frequencies for next epoch dynamically chosen from the CPU and GPU saturated frequency lookup tables (configured offline in the training phase) after calculation of CPU and GPU cost index at the end of the current epoch using the results of data capturing step.

Evaluation Step: We evaluate applications of the deployment set, using the detailed evaluation methods as described in the experimental setup of the next section.

4. EVALUATION OF CO-CAP

4.1 Experimental Setup

We evaluate Co-Cap on the ODROID-XU3 development board installed with Android 4.4.2 and Linux 3.10.9; Table 2 summarizes our platform configurations. The ODROID platform is equipped with four TI INA231 power sensors measuring the power consumption of big CPU cluster (CPU-bc), little CPU cluster (CPU-lc), GPU and memory respectively. The CPU supports cluster-based DVFS at nine frequency levels (from 1.2Ghz to 2.0Ghz) in CPU-bc and at seven frequency levels (from 1.0Ghz to 1.6Ghz) in CPU-lc, and GPU supports six frequency levels (from 177Mhz to 543Mhz).

Table 2: Platform Configuration

Feature	Description
Device	ODROID-XU3
SoC	Samsung Exynos5422
CPU	Cortex-A15 2.0Ghz and Cortex-A7 Octa-core CPUs
GPU	Mali-T628 MP6, 543Mhz
System RAM	2Gbyte LPDDR3 RAM at 933MHz
Mem. Bandwidth	up to 14.9GB/s
OS(Platform)	Android 4.4.2
Linux Kernel	3.10.9

Benchmark Sets: we use 70 gaming applications to evaluate our Co-Cap manager. In our experiments we used a 30-app training set shown in Figure 10 and the remaining 40 games as the deployment set shown in Figure 11. The training set and the deployment set are composed of game benchmarks derived from: previously published papers, traditional graphics benchmarks for performance comparison of commercial products such as GFX bench or 3D mark, popular Android games like Angry Birds and Call of Duty, and games of popular game engine companies such as Unity or Gameloft's unreal engine.

The 30-app training set (Figure 10) is based on 4 categories of workloads (No, GPU, CPU, CGU dominant) as shown in Figure 3. We used cost indices of 2 and 3 to categorize the four different types of graphics workloads in this platform. However, each dominance area also could have different workloads, which will require different CPU and GPU frequencies. Therefore, additionally we made efforts to choose nine applications in each dominance area, which correspond to Low, Medium, and High workloads in terms of the normalized CPU and GPU cost. This requires 36 applications (4 categories x 9 applications) for the training set.

Normalized CPU \ GPU Cost (Index)	0–20 (0)	20–40 (1)	40–60 (2)	60–80 (3)	80–90 (4)	90–100 (5)
0–20 (0)	Angry Birds	Trial Xtreme4	CustomRacer	G.P.A Geom1	G.P.A Geom2	Action Bike
20–40 (1)	Dhoom3	Dino Hunter	Jetski Jump Bike Racing Herculous	Dream Bike	Anomaly2	3D mark
40–60 (2)	Call of Duty :Hereos	DH 2014 D-Day	Cont. Killer S1 Dinosaur	Buggy Bandit	3Dmark-nor	Frontline2 S.1
60–80 (3)			Shoot EmDown		GPUbench	
80–90 (4)		Street Drive		Robocop		
90–100 (5)	Jetski Race	Q3Zombie M.4 Turbo-fast	GFX Driver OH			

Figure 10: The 30-app Training Set.

CPU \ GPU Cost Index	0	1	2	3	4	5
0	Ninja Fruit Extreme Bike Battlefield	Fast_bike Motocross B&G	GFX ALU	G.P.A FS1	G.P.A FS2	GFX Alpha G.P.A FS3
1	D3:speed Real Soccor Amored Car	Csr_racing Gozilla Anomaly2 Low 3D Chess	Con. Killer S2 Cite bike Anomaly2 Nor	Epic Citadel		GFX T-rex
2	FF Legacy	Comm: War 3D Terminator Moto Racer	300_game Lion Hunting	Ridge_racer	Frontline2 S.2	FC:WW2
3		Q3Zombie M.1		Antutu 2.7		
4		GT-racing2	Mnt. Sniper	Edge of T.		
5		Real Driving 3D Q3Zombie M.2 Driver				

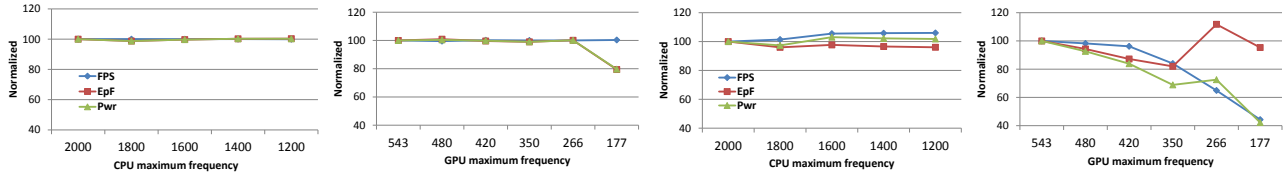
Figure 11: The 40-app Deployment Set.

However, our observations show that games having over 80 normalized CPU and GPU cost rarely exist (these quadrants are shaded in dark gray in Figure 10 and Figure 11); also it was very difficult to find games for some specific quadrants that are vacant. For these vacant quadrants, the saturated frequencies can be heuristically speculated by using interpolated values of adjacent quadrants. With these observations in hand, we believe a 30-application training set (covering the entries shown in Figure 10) are sufficient to configure successfully the CPU and GPU saturated frequencies using our heuristic.

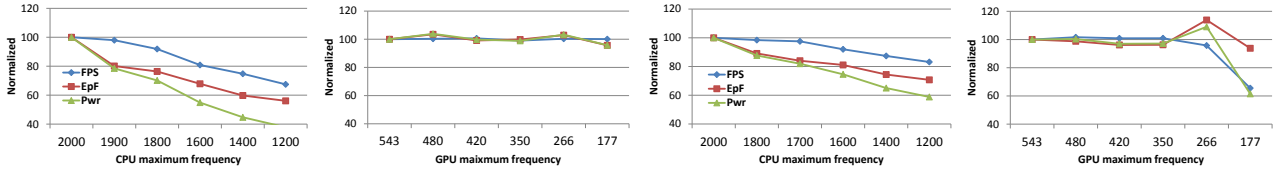
For the 40-app deployment set shown in Figure 11, we use the rest of the 70 games; of course any other games can be tested additionally. We also note that the No-dominant area has more games compared to other areas because many popular games were located in this area on ODROID-XU3 which is deploying high-performance CPUs and GPUs as we described in the start of this section.

For measurements and comparison: after capturing all data of Table 1 in every epoch (i.e., 100ms), we compared power consumption of CPU-bc, CPU-lc, GPU, and memory. The average power consumption of CPU-lc in the training set was only 8.6% of the CPU-bc, CPU-lc, and GPU power, therefore, in this work we only consider CPU-bc cost, and use the default maximum frequency(i.e., 1600Mhz) as the saturated frequency for CPU-lc.

In order to achieve a fair comparison with the baseline, we made efforts to find games that have very similar graphics workloads in every execution. In particular, the traditional benchmarks for graphics performance comparison had exactly same workloads in each execution. Because we focus on FPS, power consumption, and energy consumption during graphics rendering, we started all the benchmarks and games manually, and then began measurements manually at the start of drawing on the screen with the execution of a script file, which in turn starts measurement of power consumption using the power sensor driver and stops the measurement after a fixed amount of time. Therefore, all data such as FPS and power consumption can be automat-



(a) NO domi. (Trial Xtreme4) (b) NO domi. (Trial Xtreme4) (c) GPU domi. (Action bike) (d) GPU domi. (Action bike)



(e) CPU domi. (Q3-Zombie) (f) CPU domi. (Q3-Zombie) (g) CGU domi. (Robocop) (h) CGU domi. (Robocop)

Figure 12: FPS, Power and EpF Results of Different Types of Graphics Workloads.

ically captured during run-time, allowing us to get the final average results after a fixed amount of time.

We then compare our proposed Co-Cap manager, which is implemented within the Linux kernel layer, with the default CPU and GPU governors (i.e., Interactive CPU governor and ARM's Mali Midgard GPU governor) using FPS, power (CPU-bc, CPU-lc, and GPU) and EpF. To minimize variance across measurements, we perform repeated executions per application to get the average results.

Overhead: In order to evaluate performance and power consumption overhead, we measured the execution time of data capturing (3-4us) and Co-Cap management (1-3us) functions. The total overhead (4-7us) time can be totally negligible compared to the epoch (i.e., 100 ms) in terms of performance (FPS degradation). Moreover, any noticeable power increase was not observed in terms of average power consumption when we add the data capturing and the Co-Cap management functions.

4.2 Experimental Results

4.2.1 *Different types of graphics workloads*

Figure 12 shows the effects of CPU (or GPU) maximum frequency capping on FPS, power, and EpF for some examples of the training set. For illustration, we show a typical example application from each different graphics workload, that provides specific examples for the general schemes of graph pattern analysis shown in Figure 8. (For readability, average FPS and power of the baseline are like these: Trial Xtreme4 (FPS: 60, Power: 1790mW), Action Bike (FPS: 56, Power: 3090mW), Q3-Zombie (FPS: 43, Power: 3100mW), Robocop (FPS: 56, Power: 3250mW).

4.2.2 *CPU/GPU Saturated Frequency Lookup Tables*

Figure 13 shows the final saturated frequency lookup tables configured as output of the training phase. For the No-dominant workload, available lowest frequencies were configured. For the CPU-dominant workload, CPU saturated frequency is almost near to the maximum CPU frequency (i.e., 1800 - 1900Mhz), and GPU saturated frequency is higher than the minimum frequency (i.e., 266 - 325Mhz). For the GPU-dominant workload, GPU saturated frequency is almost near to the maximum frequency (i.e., 420 - 480Mhz), and CPU saturated frequency is higher than the minimum frequency (i.e., 1400 - 1600Mhz). Finally, for the CGU-dominant workload, lower frequencies than those of CPU- or GPU-dominant workloads were configured.

CPU \ GPU Cost Index	0	1	2	3	4	5
0	1200	1200	1200	1400	1600	1600
1	1200	1200	1200	1400	1600	1600
2	1400	1400	1400	1600	1600	1600
3	1800	1800	1800	1800	1700	1600
4	1900	1900	1800	1800	1800	1600
5	1900	1900	1900	1900	1900	1600

(a) CPU

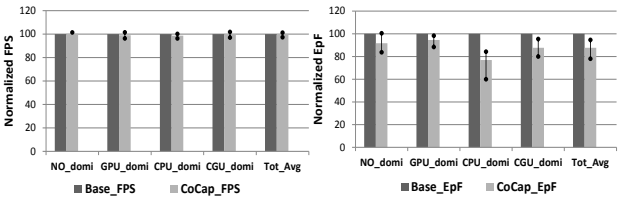
CPU \ GPU Cost Index	0	1	2	3	4	5
0	177	177	266	420	420	480
1	177	266	350	420	420	480
2	266	266	350	420	420	480
3	266	266	350	350	420	480
4	266	266	350	350	420	480
5	266	266	350	350	420	480

(b) GPU

Figure 13: Saturated Frequency Lookup Tables

4.2.3 *From Training Set*

Figure 14 shows the average results of each characterized graphics workload. Co-Cap achieves EpF improvement of 8.3%, 5.4%, 23.1%, 12.3%, and 10.6% on average in No-, GPU-, CPU-, CGU-dominant, and total applications respectively and achieves little FPS decline (0.5% on average) for total applications, (FPS decline by -0.3%, 1.0%, 1.3%, 0.6%, and 0.5% on average in No-, GPU-, CPU-, CGU-dominant, and total applications respectively).



(a) Average results of FPS (b) Average results of EpF

Figure 14: Avg results of the Training Set

Figure 15 shows that the results of the 30-app training set. It is observed that the EpF is improved in all benchmarks except Dhoom3 and the FPS is successfully maintained at almost similar to the baseline for all benchmarks. (workload of Dhoom3 is so lightweight that could not have additional power reduction in spite of the minimum CPU and GPU frequency.)

4.2.4 *From Deployment Set*

Figure 16 shows that the results of each application of the 40-app deployment set. Figure 17 shows the average results

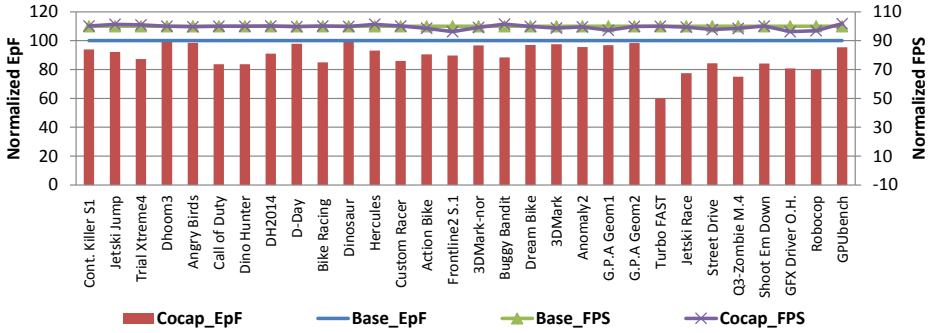


Figure 15: The results of the Training Set.

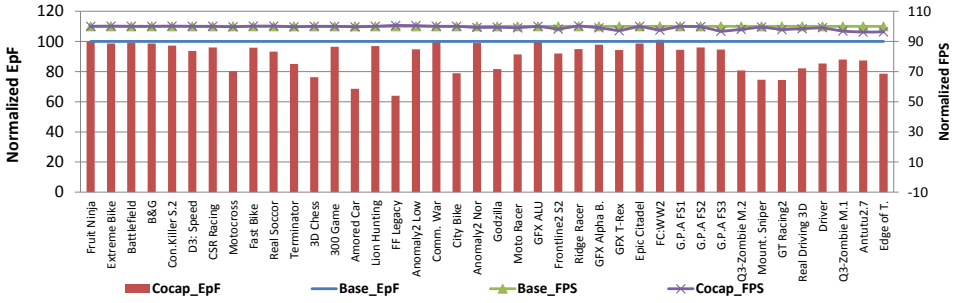
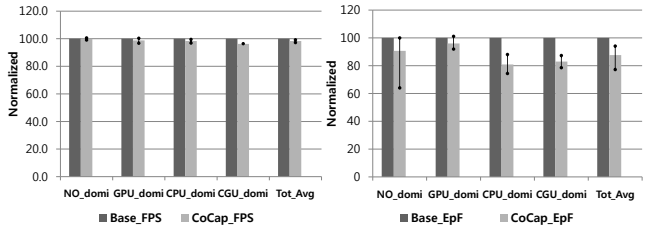


Figure 16: The results of the Deployment Set.

of each characterized graphics workload. Co-Cap achieves EpF improvement of 9.3%, 4.0%, 19.1%, 17.0%, and 10.0% on average in No-, GPU-, CPU-, CGU-dominant, and total applications respectively and achieves negligible FPS decline (0.7% on average) for total applications. These results clearly show that Co-Cap is able to achieve significant improvement in EpF with little FPS decline for all types of graphics workloads in the training and the deployment sets.



(a) Average results of FPS (b) Average results of EpF

Figure 17: Average Results of the Deployment Set

4.3 Analysis and Discussion

As shown in Figure 17(b), EpF improvement differs based on the types of graphics workloads for both the training set and the deployment set. We do observe that CPU dominant workload applications have better EpF improvement compared to GPU dominant workloads. Note that the default CPU governor supports cluster-based DVFS. We speculate that the main rendering process of graphics applications on Android system is executed on one single core even though there are four big CPU cores and four little CPU cores. Therefore, if we use a slightly lower maximum frequency removing frequency over-provisioning compared to the default maximum frequency, we can easily get significant power reduction with little FPS loss for CPU dominant applications. However, for GPU dominant workloads, GPU is especially dedicated for rendering tasks, therefore power reduction rate of GPU dominant workload (Figure 12.(d)) is less than the CPU dominant workload (Figure 12.(e)) for same FPS degradation. In addition, when we reduce GPU maximum frequency, total power consumption and EpF do not decrease gradually as shown in Figure 12.(d). For No-dominant workloads, the minimum frequency of CPU-bc or GPU is still quite high for some lightweight No-dominant graphics applications such as Ninja Fruit, Extreme Bike, and Battlefield as shown in Figure 16.

Our proposed Co-Cap methodology shows promise for improving energy efficiency across a wide range of mobile games, and also in being rapidly applicable for newer platforms as they emerge. However, our initial efforts still need to address several open issues, such as: How do we establish completeness using this training set? Can we use more sophisticated methods (e.g., Neural Networks or Machine Learning) to augment our existing heuristics for saturated frequencies? How do we incorporate the effects of memory frequency capping, given that memory utilization varies dynamically during game execution? These and other extensions are currently under active investigation.

5. CONCLUSION

In this paper, we proposed *Co-Cap, an energy-efficient CPU-GPU dynamic maximum frequency capping technique.* In the training phase, we first dynamically captured data such as utilization and frequency, estimated graphics workloads using the normalized CPU and GPU cost, and then configured the saturated frequency of CPU and GPU in

each cost window. We then evaluated the efficacy of Co-Cap using a new set of games. Our experimental results using 70 real games on the ODROID platform show that Co-Cap improves energy per frame by 10.6% and 10.0% (23.1% and 19.1% in CPU dominant applications) on average and achieves little FPS decline by 0.5% and 0.7% (1.3% and 1.7% in CPU dominant applications) on average for the training- and deployment set respectively, compared to the default CPU- and GPU governors, with negligible overhead in execution time and power consumption on ODROID-XU3. Our ongoing and future work include: 1) Proposing a smart Co-Cap using more sophisticated methodology such as neural networks machine learning algorithms. 2) Developing a memory-aware cooperative CPU-GPU DVFS governor. Finally, while this methodology was targeted mainly for mobile games, it can also be applicable for various types of CPU-GPU integrated graphics applications.

6. REFERENCES

[1] Y. Bai and P. Vaidya. Memory characterization to analyze and predict multimedia performance and power in embedded systems. In *ICASSP*, 2009.

[2] B. Dietrich and S. Chakraborty. Power management using game state detection on Android smartphones. In *MobiSys*, June 2013.

[3] B. Dietrich and S. Chakraborty. Lightweight graphics instrumentation for game state-specific power management in Android. In *Multimedia Systems*, 2014.

[4] Y. Gu and S. Chakraborty. Power management of interactive 3D games using frame structure information. In *VLSI Design*, Jan. 2008.

[5] Y. Gu, S. Chakraborty, and W. T. Ooi. Games are up for DVFS. In *DAC*, July 2006.

[6] C.-Y. Hsieh, J.-G. Park, N. Dutt, and S.-S. Lim. Memory-aware cooperative CPU-GPU DVFS governor for mobile games. In *ESTIMedia*, Oct. 2015.

[7] D. Kadjo, R. Ayoub, M. Kishinevsky, and P. V. Gratz. A control-theoretic approach for energy efficient cpu-gpu subsystem in mobile systems. In *DAC*, 2015.

[8] X. Li, G. Yan, Y. Han, and X. Li. Smartcap: User experience-oriented power adaptation for smartphone's application processor. In *DATE*, 2013.

[9] J.-G. Park, C.-Y. Hsieh, N. Dutt, and S.-S. Lim. Quality-aware mobile graphics workload characterization for energy-efficient DVFS design. In *ESTIMedia*, Oct. 2014.

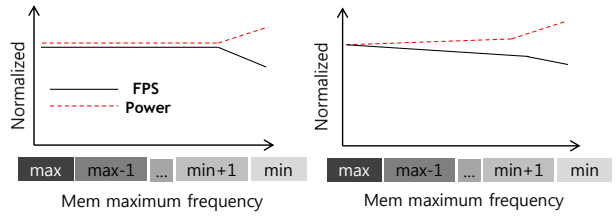
[10] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra. Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs. In *DAC*, 2015.

[11] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated CPU-GPU power management for 3D mobile games. In *DAC*, June 2014.

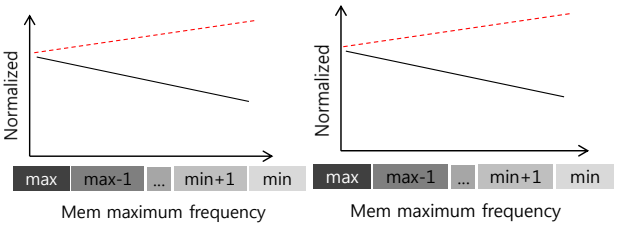
APPENDIX A
Effects of Memory Frequency Capping

We investigate the effects of the memory maximum frequency capping on FPS and the total power (CPU-bc, CPU-lc, GPU, and DRAM) as shown in Figure 18 and 19. In this appendix, we describe the general trend of memory frequency capping and the results. (Based on the results, we did not apply the memory frequency capping into the Co-Cap manager).

First, we explain the general trend of memory frequency capping effects for the four different application categories (No-, CPU-, GPU-, and CGU-dominant quadrants) shown in Figure 18. Additionally, as shown in Figure 19, we add a typical example application in each different graphics workload for illustration.

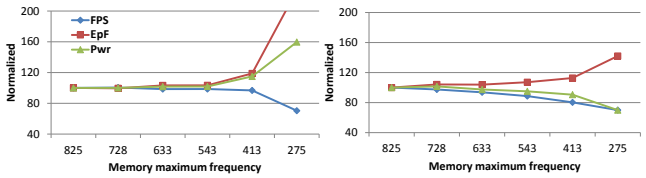


(a) NO dominant workload (b) CPU dominant workload

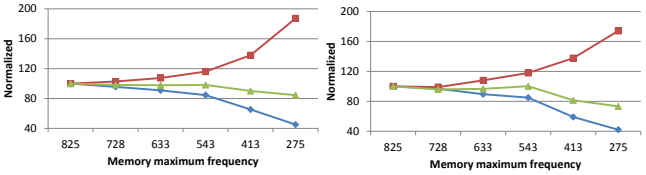


(c) GPU dominant workload (d) CGU dominant workload

Figure 18: Effects of Memory maximum frequency capping on FPS and Power.



(a) NO domi.(Trial Xtreme4) (b) CPU domi. (Q3-Zombie)



(c) GPU domi. (Action Bike) (d) CGU domi. (Robocop)

Figure 19: Results of Memory maximum frequency capping on FPS, Power, and EpF.

Figure 18.(a) and (b) (the illustration in Figure 19.(a) and (b)) show the performance/power consumption graph trends of No- and CPU-dominant workloads respectively. FPS and the total power remain the similar for overall memory frequencies except the very low frequencies, which means that configuring the saturated frequencies of memory does not have opportunities for power saving.

In addition, Figure 8 (c) and (d) (the illustration in Figure 19.(c) and (d)) show the performance/power consump-

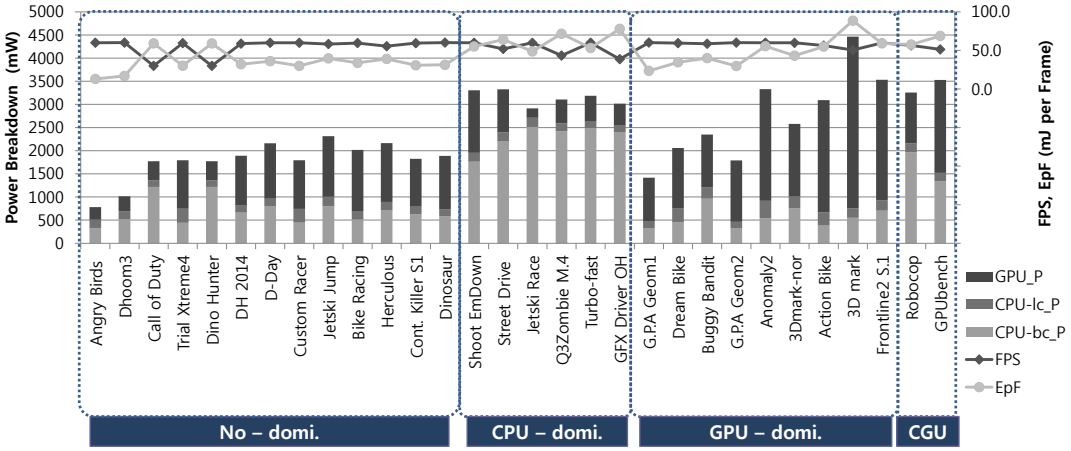


Figure 20: FPS, Power breakdown, and EpF of the Training Set.

tion pattern of GPU- and CGU- dominant workloads respectively. For these workloads, the opportunities for FPS and power saving do not exist. In other words, as the maximum memory frequency scales down, FPS degrades and the total power increases (even worse). This makes memory capping unattractive for performance and power saving. Therefore, according to the observation on the effects of memory maximum capping, we use the default maximum memory frequency (i.e., 825Mhz) as the saturated frequency for memory.

The similar results on the impacts of memory frequency scaling are also can be observed in the recent work [11]. However, memory-aware cooperative CPU-GPU DVFS governor for energy saving of mobile games can be observed in our previous related work [6].

APPENDIX B
Characteristics of the Training Set

As described in Section 4.1, we used a 30-app training set (Figure 10) for the training phase. In order to understand more accurately each application of the training set and compare comprehensively results of applications, Table 3 and Figure 20 show the characteristics of the training set: 1) CPU/GPU average utilization, frequency, and normalized cost for a certain amount of execution time (the execution time may be different in each application). 2) Not normalized FPS, power breakdown (CPU-bc, CPU-lc, and GPU), and EpF.

In Table 3, applications are separated into four quadrants: No-dominant (from Angry Birds to Dinosaur), CPU-dominant (from Shoot Em Down to GFX Driver OH), GPU-dominant (from G.P.A Geom1 to Frontline2 S.1) and CGU-dominant (Robocop and GPUbench). And each column corresponds to CPU utilization, CPU frequency, normalized CPU cost, GPU utilization, GPU frequency, and normalized GPU cost. (For instance, GPUbench (the last application) has the values of CPU (93% util, 1639Mhz, 76) and GPU (95% util, 490Mhz, 86) related characteristics.

Additionally, as shown in Figure 20, we show the information of measured FPS, power (CPU-bc, CPU-lc, and GPU), and EpF (mJ per frame or Watt per FPS). For instance, Angry Birds have 60 FPS, 800 mW (GPU : CPU-lc : CPU-lc = 300 : 200 : 300), and 13 EpF (800/60). On the other hand, GPUbench has 50 FPS, 3500 mW (GPU : CPU-lc : CPU-bc = 2000 : 200 : 1300), and 70 EpF (3500/50).

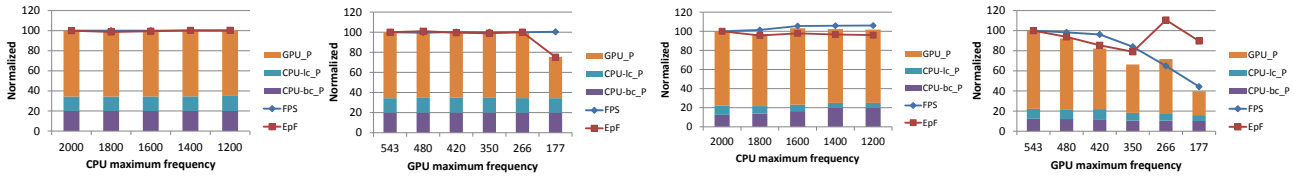
Using these information, we can characterize each application of the training set, and analyze the results more accurately.

Table 3: Util, Freq, and Cost of the Training Set

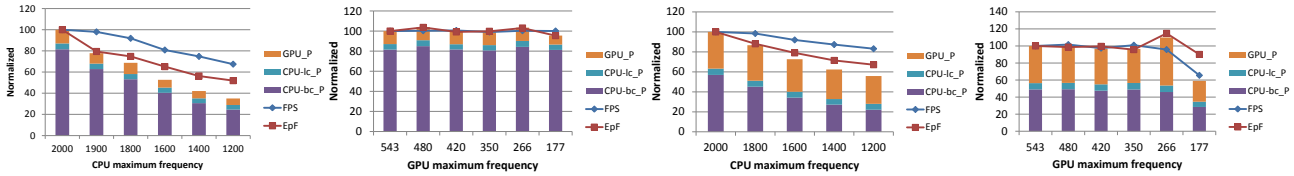
App	C_util	C_freq	C_cost	G_util	G_freq	G_cost
Angry Birds	5	1220	2	32	178	10
Dhoom3	57	1211	34	43	178	13
Call of Duty	59	1794	52	65	177	20
Trial Xtreme4	49	1212	29	74	268	36
Dino Hunter	59	1794	52	65	177	20
DH 2014	66	1277	41	67	261	31
D-Day	85	1349	56	76	353	49
Custom Racer	10	1219	6	86	356	55
Jetski Jump	59	1224	35	82	328	48
Bike Racing	50	1213	29	83	284	42
Herculous	56	1296	36	82	369	54
Cont. Killer S1	79	1221	47	79	341	48
Dinosaur	71	1219	42	79	334	47
Shoot EmDown	78	1737	66	84	382	57
Street Drive	79	1797	92	72	267	34
Jetski Race	92	2033	92	50	180	15
Q3Zombie M.4	95	2010	95	65	265	31
Turbo-fast	99	2004	99	71	177	22
GFX Driver OH	95	2000	95	84	267	41
G.P.A Geom1	0	1200	0	86	420	66
Dream Bike	45	1219	27	86	429	66
Buggy Bandit	66	1451	50	86	409	65
G.P.A Geom2	0	1200	0	98	480	86
Anomaly2	53	1206	32	93	475	81
3Dmark-nor	70	1340	48	85	394	62
Action Bike	34	1216	20	99	520	94
3D mark	58	1206	34	99	540	99
Frontline2 S.1	80	1235	48	99	513	92
Robocop	87	1970	85	85	419	65
GPUbench	93	1639	76	95	490	86

APPENDIX C
Additional Detailed Results and Analysis

In this appendix, we add more detailed results and analysis: 1) Power breakdown of Figure 12 (FPS, power and EpF results of different types of graphics workloads) in Fig-



(a) NO domi. (Trial Xtreme4) (b) NO domi. (Trial Xtreme4) (c) GPU domi. (Action bike) (d) GPU domi. (Action bike)



(e) CPU domi. (Q3-Zombie) (f) CPU domi. (Q3-Zombie) (g) CGU domi. (Robocop) (h) CGU domi. (Robocop)

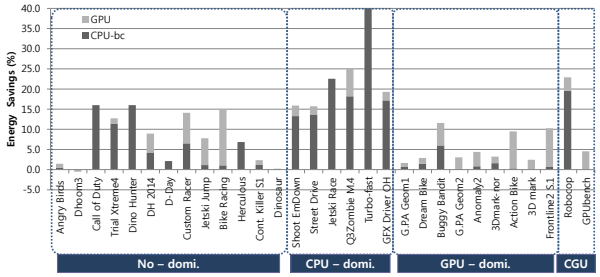
Figure 21: FPS, Power and EpF Results of Different Types of Graphics Workloads.

ure 21. 2) Energy savings of the training set in Figure 22.(a). 3) FPS degradation and power savings of the training set in Figure 22.(b).

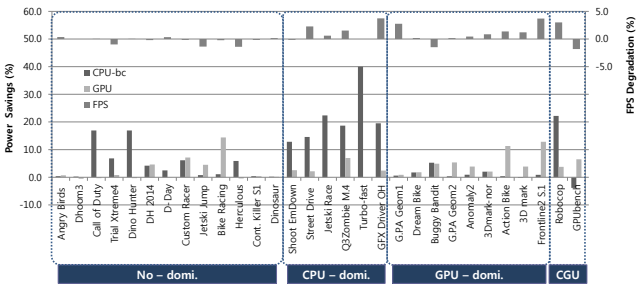
Using the power breakdown added in Figure 21, we can analyze that power (energy) savings can be obtained from which component (CPU or GPU). For example, Figure 21.(d) (the GPU-dominant example) shows that the main power contribution results from GPU component. On the other hand, Figure 21.(e) (the CPU-dominant example) shows that the main power contribution comes from CPU-bc component.

Figure 22.(a) shows the total (CPU-bc and GPU) energy savings per frame compared to the default policy. Our Co-Cap technique outperforms the default policies on all applications (we assume the Dhoom3 is in the margin of error, -0.9% - 0.9%). The results show a significant combined CPU-bc+GPU average energy savings of 10.6% across all the applications. On average, the contribution of CPU-bc component to the energy savings is 7.4% while the contribution of GPU component is 3.2%. A larger savings result from the CPU-bc component because the power reduction rate in the CPU-dominant applications (Figure 21.(e)) is faster than that of the GPU-dominant application (Figure 21.(d)). The energy savings are more remarkable on the CPU- and CGU-dominant applications (23.1% and 12.3% respectively). On the other hand, GPU-dominant applications have less energy savings (5.4%). In the No-dominant applications, the average energy savings are 8.3%, but the energy saving of each application was very various: the highest 16.3% for Call of Duty and the lowest -0.5% for Dhoom3.

The average CPU and GPU power savings are shown in Figure 22(b). The most remarkable CPU power savings are observed in the CPU-dominant applications; mostly because the CPU maximum frequency (the saturated frequency) is set lower than the default policy within minimal FPS degradation for these applications. Additionally, CPU intensive applications in the No-dominant quadrants such as Call of Duty and Dino Hunter and CPU intensive application like Robocop in the CGU-dominant quadrants also show significant CPU power savings. On the other hand, the GPU power savings are observed in the GPU-dominant applications and GPU intensive applications in the No-dominant quadrants such as Custom Racer, Jetski Jump and Bike Racing, but the percentage of GPU power savings are less than that of CPU power savings. From the observations of the results, we speculate that relative CPU- or GPU-intensiveness of workloads in No- and CGU-dominant quadrants are also important factors.



(a) Energy Savings of the Training Set.



(b) Power Savings of the Training Set.

Figure 22: Detailed Results on the Training Set.

FPS degradation: the top of Figure 22.(b) shows the performance degradation of our proposal in each application of the training set. An insignificant FPS degradation of a 0.5% on average was observed across all applications, but a few specific applications such as GFX Driver OH, Frontline2, and Robocop have a FPS degradation of more than 3%. Our analysis speculates that more completeness using the training set or more sophisticated methods will minimize the FPS degradation on specific applications, and those will be part of our future work.