Hindawi
Mobile Information Systems
Volume 2017, Article ID 9642958, 11 pages
https://doi.org/10.1155/2017/9642958

Research Article

An Adaptive and Integrated Low-Power Framework for Multicore Mobile Computing

Jongmoo Choi,[1] Bumjong Jung,[1] Yongjae Choi,[1] and Seiil Son[2]

[1] *Department of Software, Dankook University, Yongin, Republic of Korea*
[2] *Korea Communications Agency, Daejeon, Republic of Korea*

Correspondence should be addressed to Jongmoo Choi; choijm@dankook.ac.kr

Received 20 January 2017; Accepted 15 March 2017; Published 12 June 2017

Academic Editor: Karl Andersson

Employing multicore in mobile computing such as smartphone and IoT (Internet of Things) device is a double-edged sword. It provides ample computing capabilities required in recent intelligent mobile services including voice recognition, image processing, big data analysis, and deep learning. However, it requires a great deal of power consumption, which causes creating a thermal hot spot and putting pressure on the energy resource in a mobile device. In this paper, we propose a novel framework that integrates two well-known low-power techniques, DPM (Dynamic Power Management) and DVFS (Dynamic Voltage and Frequency Scaling) for energy efficiency in multicore mobile systems. The key feature of the proposed framework is adaptability. By monitoring the online resource usage such as CPU utilization and power consumption, the framework can orchestrate diverse DPM and DVFS policies according to workload characteristics. Real implementation based experiments using three mobile devices have shown that it can reduce the power consumption ranging from 22% to 79%, while affecting negligibly the performance of workloads.

1. Introduction

Intelligent services are actively introduced into mobile computing environments [1, 2]. For instance, modern black box devices support not only the traditional recording service but also ADAS (Advanced Driver Assistance Systems) such as blind spot monitoring, pedestrian detection, and automatic emergency braking. Smartphones provide voice recognition and image processing for better HCI (Human Computer Interface) and big data processing and deep learning for context-aware services with the consideration of user personality.

To support such intelligent services efficiently, mobile computing devices are equipped with multiple cores. Smartphones have the heterogeneous multicore architecture, called big.LITTLE, which consists of performance-optimized big cores and energy-optimized little cores with a single ISA (Instruction Set Architecture) [3]. Recently released embedded boards for IoT (Internet of Things) such as Raspberry Pi, Odroid, Edison, Jetson, and Artik also provide multiple cores [4–6].

However, employing multicore in mobile devices triggers a new issue. As the number of cores increases, the power consumption used by cores becomes a significant portion in mobile devices. The increased power consumption puts pressure on the energy resource in a mobile device due to the limitation in battery capacity [7]. In addition, it causes the high internal temperature in a mobile device [8], having a potential to result in the thermal runaway [9].

To address the multicore power consumption issue, two well-known techniques are devised [7, 10]. One is DPM (Dynamic Power Management), also known as *offlining*, which turns off individual cores in order to reduce the per-core power consumption [11]. The other technique is DVFS (Dynamic Voltage and Frequency Scaling), which decreases the frequency and voltage of a core. [12].

One interesting observation is that most mobile applications have limited parallelism [13–15]. For instance, Seo et al. analyze how the multiple cores are utilized in mobile devices using TLP (Thread Level Parallelism) and observe that TLP of most mobile applications is ranging from 1.4 to 3.9 [15]. It implies that the required active cores for the applications are less than 3.9 on average, disclosing the opportunity of DPM and DVFS.

In this paper, we propose a new low-power framework for multicore mobile devices. It supports both DPM and DVFS in an integrated manner. Also, it keeps track of the CPU load of applications and turns on/off cores or changes frequency adaptively so that it can reduce the power consumption while trying to minimize its impact on performance.

The framework consists of three components, namely, online resource usage monitor, lower-power controller, and policy manager. The online resource monitor gathers resource usage statistics such as CPU utilization and power measurement. The lower-power controller materializes the DPM and DVFS techniques using the Linux *CPU hotplug* and *governor* mechanism, respectively [16, 17].

Finally, the policy manager provides a rule regarding how to integrate DPM and DVFS techniques based on their overhead and effect on the power consumption. Also, it applies the integrated solution appropriately according to the resource usage characteristics of applications. In addition, it supports user interfaces so that a user can configure his/her own control parameters.

We have implemented the framework in the Linux kernel version 3.10 and have evaluated its effectiveness using three mobile devices. Experimental results show that the workload-aware adaptability of the framework can reduce the power consumption ranging from 22% to 79%, while not affecting the performance of workloads greatly. Also, we observe that the power reduction depends on the features of a mobile device, especially in the big.LITTLE architecture.

The rest of this paper is organized as follows. In Section 2, we explain previous studies related to this work. Then, our proposed framework is discussed in Section 3. Section 4 presents real implementation based experimental results. Finally, we summarize conclusion and future work in Section 5.

2. Related Work

As the energy efficiency becomes a critical issue, a lot of studies have been conducted for analyzing and enhancing the power consumption in mobile computing systems. In this section, we classify these studies into the following four categories: power consumption analysis, workload analysis, DPM/DVFS techniques, and system/application-level approach.

2.1. Power Consumption Analysis. Understanding of where and how power is consumed in mobile devices is a key requirement for efficient power management. Carroll and Heiser present a detailed analysis of power consumption in mobile phones using Google Nexus One, HTC Dream, and OpenMoko Neo Freerunner [18]. They provide not only the overall system power but also the breakdown of power consumed by the main hardware components. Their analysis shows that wireless communication and display are the heaviest power consumers. Also, CPU is identified as a heavy consumer especially for the CPU intensive workloads and in the suspended state.

As the number of cores increases, the power consumption used by cores also increases sharply. Several studies demonstrate that the power consumption increases with the number of active cores and as the voltage and frequency increase [6–8]. Tawara et al. also present how the power consumption affects the internal temperature using thermography [8]. Zhu and Shen observe that even though the power consumption increases with the number of cores, the first activated core incurs much higher power cost than each additional core does in the same processor due to the shared resources [6].

2.2. Workload Analysis. Even though employing multicore in mobile devices increases the power consumption, it also gives an opportunity to reduce the application execution time, eventually leading to energy saving. To explore the opportunity, several studies have examined how much parallelism is there in mobile applications [13–15].

Gao et al. analyze how the multiple cores are utilized in mobile devices using TLP (Thread Level Parallelism) [13]. They report that TLP of the most mobile applications including browser, map, music, and games is less than 2 on average, meaning that the number of active cores used by the majority of applications is below 2. Similar behavior is also observed by Zhang et al. [14]. Seo et al. investigate TLP using various mobile benchmark applications in mobile devices and observe that applications have TLP ranging from 1.4 to 3.9 [15]. These studies uncover the necessity to turn off cores or to adjust voltage and frequency adaptively according to different program execution phases.

2.3. DPM/DVFS Techniques. DPM and DVFS are two well-known techniques for dynamic energy-aware CPU managements. For DPM, Linux provides the CPU hotplug mechanism that allows cores to be added to or removed from a running kernel [16]. For DVFS, Linux supports the governor mechanism that permits user to adjust the CPU frequency [17]. There are several governors such as ondemand, powersave, and performance, which will be discussed further in Section 3.

Carroll and Heiser explore how to use DPM and DVFS to reduce power consumption [7, 19]. They propose a framework, called *medusa*, an offline-aware frequency governor, which integrates core offlining with frequency scaling. In addition, they find that modern smartphones have quite different characteristics, implying that policies that work well on one processor can lead to poor results on another. Tawara et al. design a framework, called *idle reduction*, which turns on/off cores or changes the CPU frequency dynamically according to the intensity of workloads [8].

These two works are closely related to our work in that they integrate DPM and DVFS and apply them adaptively based on workload characteristics. However, our proposed framework differs from them in the following three aspects: (1) we consider not only the homogeneous but also heterogeneous multicore devices, (2) we carefully separate policy and mechanism to support flexibility, and (3) we provide a configuration file with various control parameters so that a user can easily configure his/her preferred policy.

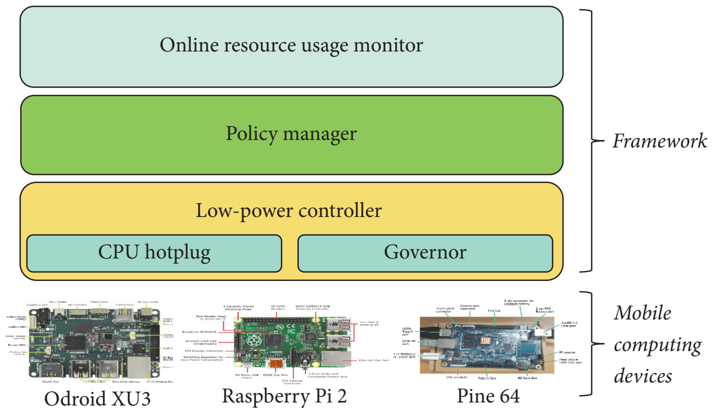


FIGURE 1: Structure of the proposed low-power framework.

Zhu et al. observe that the wakeup delay of the sleeping cores takes hundreds of microseconds, which is at least 100 times slower than the frequency change delay [20]. To hide this latency, they devise an anticipatory CPU wakeup for maintaining high performance. Song et al. propose a framework that applies low-power techniques based on user-perceived response time analysis [21].

Wamhoff et al. design a library that assigns heterogeneous and even boosted frequencies for accelerating performance [22]. Chiesi et al. present a power-aware scheduling algorithm on heterogeneous architectures to reduce the peak power [23].

DPM and DVFS are also studied in the real-time research area [10, 24, 25]. Bambagini et al. present a survey paper that discusses various energy-aware scheduling algorithms for real-time mobile systems [24]. Li and Broekaert design a DVFS based low-power scheduler that exploits slack times with an intratask intrusive approach [25]. Chen et al. devise a technique that models the idle intervals of individual cores to optimize the DVFS and DPM for real-time tasks [10].

2.4. System/Application-Level Approach. Roy et al. propose a new operating system, called *cinder*, for mobile phones and energy-constrained computing devices [26]. It supports two new abstractions, *reserves* and *taps*, which store and distribute energy for applications. Snowdon et al. design a framework; they refer to it as *Koala*, which provides a model, a generalized energy-delay policy, and a single parameter for tuning the system to an overall energy-management objective [27].

Shen et al. present a new operating system facility, called *power containers*, that controls the power and energy usage of individual fine-grained requests in multicore systems [28]. Zhu and Shen observe the energy disproportionality in multicore devices, where the first running CPU incurs much higher power cost than each additional core does [6]. Kwon et al. propose a framework that predicts the computational resource consumption on mobile devices using program analysis and machine learning [29].

Thiagarajan et al. design an infrastructure for measuring energy used by a mobile browser to render web pages such as email, e-commerce, and social networking sites [30]. Using the infrastructure, they observe that downloading and parsing CCS (Cascade Style Sheets) and JavaScript consume a large portion of energy and recommend how to design web pages for enhancing energy efficiency. Bui et al. propose techniques, namely, adaptive content painting and application-assisted scheduling, to improve the energy efficiency of web page loading [31].

3. Adaptive and Integrated Low-Power Framework

In this section, we first explain the overall structure of our framework and principle used to design the framework. Then, we discuss the details of each component in sequence.

3.1. Overall Structure. Figure 1 displays the overall structure of the adaptive and integrated low-power framework proposed in this paper. It consists of three components, called online resource usage monitor, policy manager, and low-power controller.

When we design our framework, we use the design principle that separates policy and mechanism to support diverse policies flexibly. The policy manager takes charge of the policy decision, while the low-power controller provides mechanisms for DPM and DVFS. The online resource monitor is used for supporting adaptability based on resource usage patterns including the CPU utilization and power consumption.

Currently, the framework works on three multicore-based mobile computing devices, namely, Odroid XU3 [32], Raspberry Pi2 [33], and Pine 64 [34]. Note that the framework can be installed any Linux-based devices since it utilizes the standard Linux interfaces only.

3.2. Low-Power Controller. The low-power controller component provides mechanisms for DPM and DVFS. Specifically, it makes use of the CPU hotplug mechanism [16] for DPM and the governor mechanism [17] for DVFS as shown in Figure 1.

The CPU hotplug is a mechanism that turns on and off an individual core dynamically. It was originally designed to

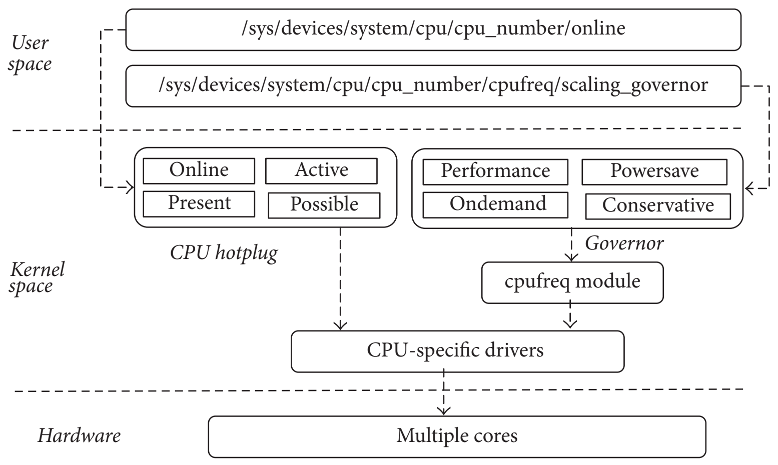


FIGURE 2: Internals of the low-power controller.

allow a failing core to be removed from a running system, but now it is popularly used for energy management.

Figure 2 shows the internal behavior of the low-power controller. The CPU hotplug mechanism supports a file, /sys/devices/system/cpu/cpu_number/online file, into user space using the *sysfs* file system that is a pseudo file system exporting various kernel information in Linux. A user can turn on or off a core by writing 1 or 0 to the file.

While turning on or off a core, the core goes through various states such as possible, present, active, and online [35]. At each state, the mechanism invokes various functions provided by CPU-specific drivers such as cpu_up(), cpu_online(), cpu_down(), and cpu_down_prepare(). Finally, the on/off request is applied into the designated core at hardware level.

The most time consuming job in the CPU hotplug mechanism is the context management. To turn off a core, it needs to save the context of a process that runs on the core and migrates the context to another core to continue its execution. Also, to turn on a core, it has to prepare a new scheduling queue for the core to be active. Hence, the latency for a core on/off is much higher than that for changing frequency of a core. We need to consider this difference for our integrated low-power management.

For DVFS, the low-power controller makes use of the governor mechanism. It allows changing the frequency of a core dynamically. It supports a file, /sys/devices/system/cpu/cpu_number/cpufreq/scaling_governor, into user space as shown in Figure 2.

Linux employs several default governors, namely, performance, powersave, ondemand, conservative, and userspace [17]. The distinctions of these governors are illustrated in Figure 3. In the performance governor, a core always runs at the maximum frequency. On the contrary, in the powersave governor, a core runs at the minimum frequency.

In the ondemand governor, a core runs initially at the minimum frequency. When a CPU load increases, the frequency becomes the maximum frequency immediately to minimize the effect of DVFS on the application execution time. When the load decreases, the frequency goes down gradually into the minimum frequency. In the conservative governor, a core runs initially at the minimum frequency like the ondemand governor. But, when the load becomes intensive, the frequency increases gradually. Finally, the userspace governor allows any frequency to be set by a user.

Commercial smartphones extend these governors, devising their own specific governors. For instance, Android uses a governor, called interactive governor, which behaves similar to the ondemand governor but responds more quickly [15].

Each governor is based on a subsystem, called *cpufreq*, which provides interfaces to explicitly set frequency on cores. This subsystem eventually makes use of the CPU-specific drivers that actually implement CPU on/off and frequency change functionality for various vendors including ARM, Intel, and AMD.

The low-power controller integrates the CPU hotplug and governor mechanisms and provides virtual interfaces such as increase_computing() and decrease_computing(). These interfaces invoke the mechanisms appropriately based on the decision dictated by the policy manager, which will be further discussed in Section 3.4.

3.3. Online Resource Usage Monitor. The online resource usage monitor gives information about how much resources are used by the current workload so that our framework can apply the low-power mechanisms adaptively. It makes use of the Linux *proc* file system, measuring various resource usage statistics such as CPU utilization, memory footprint, power consumption, and process activities. In addition, it reports the measured statistics via a web page using Node.js.

Figure 4 presents the CPU utilization statistics measured by the monitor on the Odroid XU3 device. This device has the big.LITTLE architecture, consisting of four little cores (ARM Cortex-A7) and four big cores (ARM Cortex-A15). The figure shows that the first little core (CPU0) runs at the frequency of 1.4 GHz whose utilization for user, system, and idle state is 6.79%, 3.61%, and 88.72%, respectively.

To measure the power consumption, the monitor makes use of the power measurement functionality

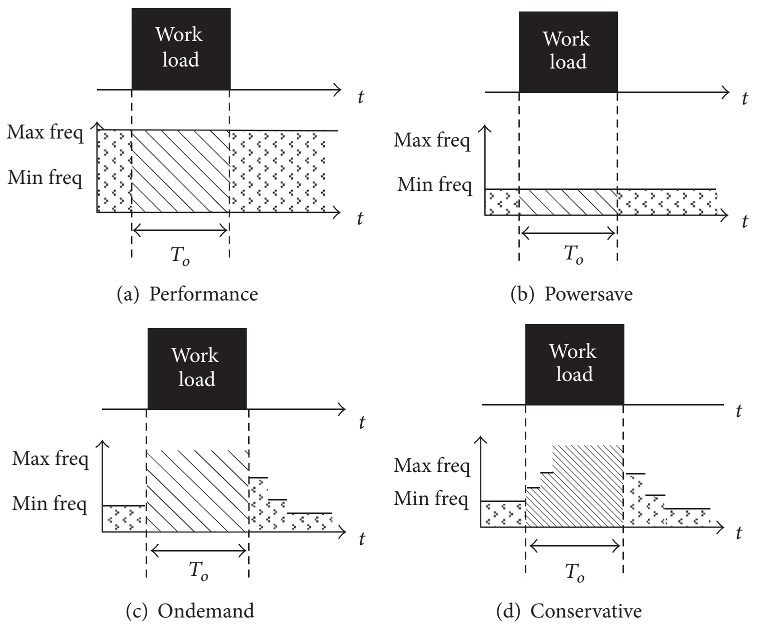
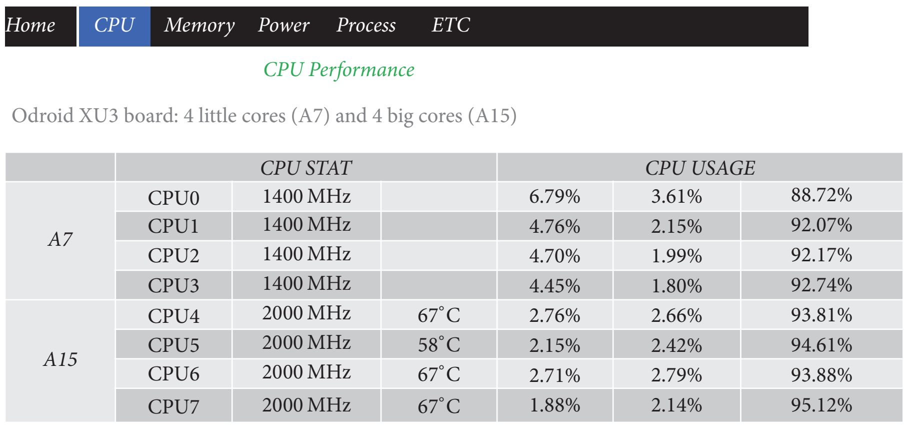


Figure 3: Frequency changes in various governors.



Odroid XU3 board: 4 little cores (A7) and 4 big cores (A15)

	CPU STAT			CPU USAGE		
A7	CPU0	1400 MHz		6.79%	3.61%	88.72%
	CPU1	1400 MHz		4.76%	2.15%	92.07%
	CPU2	1400 MHz		4.70%	1.99%	92.17%
	CPU3	1400 MHz		4.45%	1.80%	92.74%
A15	CPU4	2000 MHz	67°C	2.76%	2.66%	93.81%
	CPU5	2000 MHz	58°C	2.15%	2.42%	94.61%
	CPU6	2000 MHz	67°C	2.71%	2.79%	93.88%
	CPU7	2000 MHz	67°C	1.88%	2.14%	95.12%

Figure 4: CPU utilization reported by the online resource usage monitor on the Odroid XU3 device.

already equipped in a mobile device. The Odroid XU3 device supports such functionality, providing the power consumption of each unit including big core, little core, GPU, and RAM. For devices that do not support such functionality, Raspberry Pi 2 and Pine 64 in this paper, we utilize an external power meter that can quantify the overall power consumption by assessing the voltage driven into the power unit.

3.4. Policy Manager. Based on the measured information provided by the online resource usage monitor, the policy manager makes a policy that can lead to better energy efficiency. Then, it enforces the policy using interfaces supported by the low-power controller.

The policy manager introduces four control parameters, namely, *max_utilization*, *min_utilization*, *max_frequency*, and *min_frequency*. The former two parameters are used to trigger the policy manager to enforce its policy while the latter two parameters are used for DVFS. For instance, when the current utilization of a core is higher than the max_utilization, the policy manager is triggered to increase computing resource.

Since our framework integrates two techniques, DPM and DVFS, we need to have a rule about which technique is applied first. The policy manager supports two options. The first option is providing an interface so that a user can specify his/her preference. The second option is giving a default rule by considering the overhead of the two techniques and features of mobile devices.

To devise a guideline for the default rule, we analyze the overhead of the techniques and the power reduced by them using the online resource usage monitor. We make the

```
decrease_computing()
for each big core do
    if (core.freq > parameter.big.min_freq) then
        decrease core.freq to the next lower level
        return
for each big core do
    if (core.state == on) then
        turn off this core
        return
for each little core do
    if (core.freq > parameter.little.min_freq) then
        decrease core.freq to the next lower level
        return
for each little core do
    if (core.state == on) then
        turn off this core
        return
```

Algorithm 1: Pseudocode for decreasing computing resources in the policy manager.

```
increase_computing()
for each little core do
    if (core.state == off) then
        turn on this core
        return
for each little core do
    if (core.freq < parameter.little.max_freq) then
        increase core.freq to the next higher level
        return
for each big core do
    if (core.state == off) then
        turn on this core
        return
for each big core do
    if (core.freq < parameter.big.max_freq) then
        increase core.freq to the next higher level
        return
```

Algorithm 2: Pseudocode for increasing computing resources in the policy manager.

following three observations. First, the latency to turn off a core is much longer than that to change the frequency of a core. Second, the power saved by turning off a core is smaller than that by decreasing the frequency of a core. This result is also observed by Carroll and Heiser's study where they recommend that one should "offline cores conservatively and reduce frequency aggressively" [19]. This is partly because cores share various resources in the same processor [6]. Third, the power used by big cores is much higher than that by little cores.

Our observations lead us to design an algorithm, shown in Algorithm 1, which is invoked when we decrease computing resource. The algorithm prefers big cores to little ones and prefers DVFS to DPM by default. Specifically, it first tries to reduce the frequency among big cores. If it is not possible, it tries to turn off a big core. Again, if not feasible, it tries to reduce the frequency and to turn off a core among little cores. When one of the four "if" conditions satisfies, this function returns without going further.

Algorithm 2 presents an algorithm for increasing computing resources. The sequence of this algorithm is reverse to that of the one in Algorithm 1. It gives a higher priority for little cores. Then, it tries to turn on a core before increasing the frequency. The next higher level and the next lower level in the pseudocodes are determined by the governors, discussed in Figure 3.

The policy manager provides a configuration file so that a user can configure his/her preferred parameters. The parameters include min/max utilization, min/max frequency, DPM/DVFS preference, monitoring period, and monitoring number. The default values for the preference, monitoring period, and monitoring number are DPM preference, 200 milliseconds, and 5, respectively. The monitoring number is the number of consecutive measurements for calculating the moving average of the CPU utilization. When it is small, the policy tries to apply the low-power techniques aggressively. When a user prefers DVFS, the first/third part of the pseudocodes is changed with the second/fourth part.

Table 1: Multicore-based mobile devices.

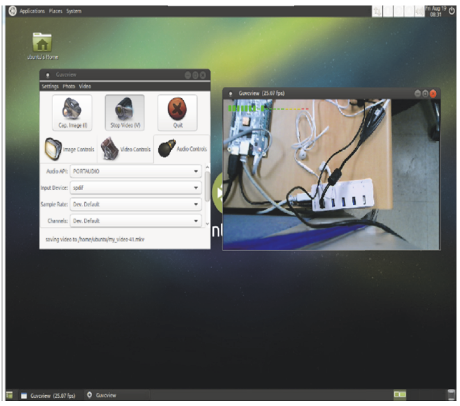
Device	Multicore description
Odroid XU3 [32]	Octa cores (Cortex-A15 4 cores, Cortex-A7 4 cores)
Raspberry Pi 2 [33]	Quad cores (Cortex-A7 4 cores)
Pine 64 [34]	Quad cores (Cortex-A53 4 cores)

4. Evaluation

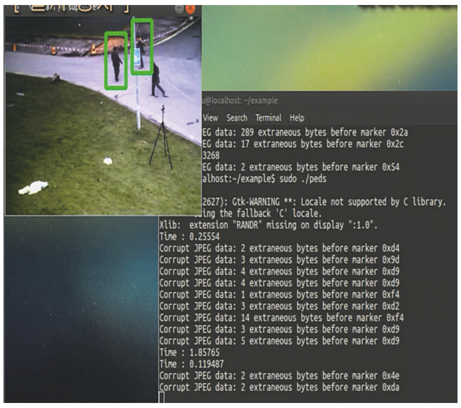
In this section, we first describe our experimental devices and workloads considered in this paper. Then, we discuss evaluation results from the power measurements to the power and energy saving achieved by the proposed framework.

4.1. Experimental Environments. We have implemented the framework on the Linux kernel version 3.10. The online resource usage monitor uses the proc file system for monitoring and reports usage statistics via a web page using Node.js. The low-power manager provides the integrated low-power interfaces based on the CPU hotplug and governor mechanism. The policy manager is implemented as a daemon process that analyzes resource usage statistics at every 200 milliseconds and applies the integrated low-power interfaces when the current CPU utilization is above/below the threshold of the max/min utilization, whose default values are 80% and 20% in this experiment.

Table 1 summarizes three mobile computing devices used in this study. The Odroid XU3 device has the big.LITTLE



(a) Video recording



```
u@localhost: ~/example
View  Search  Terminal  Help
EG data: 289 extraneous bytes before marker 0x2a
EG data: 17 extraneous bytes before marker 0x2c
3268
EG data: 2 extraneous bytes before marker 0x54
alhost:~/example$ sudo ./peds

2627): Gtk-WARNING **: Locale not supported by C library.
 ing the fallback 'C' locale.
Xlib:  extension "RANDR" missing on display ":1.0".
Time : 0.25554
Corrupt JPEG data: 2 extraneous bytes before marker 0xd4
Corrupt JPEG data: 3 extraneous bytes before marker 0x9d
Corrupt JPEG data: 4 extraneous bytes before marker 0xd9
Corrupt JPEG data: 4 extraneous bytes before marker 0xd9
Corrupt JPEG data: 1 extraneous bytes before marker 0xf4
Corrupt JPEG data: 3 extraneous bytes before marker 0xd2
Corrupt JPEG data: 14 extraneous bytes before marker 0xf4
Corrupt JPEG data: 3 extraneous bytes before marker 0xd9
Corrupt JPEG data: 5 extraneous bytes before marker 0xd9
Time : 1.85765
Time : 0.119487
Corrupt JPEG data: 2 extraneous bytes before marker 0x4e
Corrupt JPEG data: 2 extraneous bytes before marker 0xda
```

(b) Pedestrian detection

FIGURE 5: Workloads for experiments.

architecture, consisting of four little cores (Cortex-A7) and four big cores (Cortex-A15). Each core supports the same ISA and equips L1, L2 cache, where the size of L2 cache for little and big core is 512 KB and 2 MB, respectively. Both the Raspberry Pi 2 and Pine 64 devices have homogeneous four cores, Cortex-A7 and Cortex-A53, respectively. Note that the multicore architecture used in Odroid XU3, also called Exynos 5422, is actually employed for commercial mobile devices including Samsung Galaxy S5 and Chromebook 2.

We use three workloads for experiments. The first one is a video recording, as shown in Figure 5(a). It is an I/O intensive workload, recording using a camera and displaying through LCD. The second one is the Sunspider test suite [36]. It is a CPU intensive workload that tests JavaScript performance including function calls, math, and recursion without rendering. The third one is a pedestrian detection using the Haar classifier [37].

4.2. Evaluation Results

4.2.1. Video Recording Workload Results. This section consists of two parts. In the first part, we explain evaluation results observed using the Odroid XU3 device that has heterogeneous 8 cores. The second part is the results of the Raspberry Pi2 and Pine 64 device that have homogeneous 4 cores. Note that, in Odroid XU3, we can measure the power consumption of each component individually using the measurement functionality already equipped in the device while, in Raspberry Pi2 and Pine 64, we only measure the overall power consumed by the device using the external power meter, as discussed in Section 3.3.

Figure 6 presents the power measurement results of the Odroid XU3 device when it is in an idle state. The results are reported by the online resource usage monitor discussed in Section 3.3. This measurement is conducted under the baseline configuration where all hardware components are powered on. The results reveal that big cores are the heaviest power consumer, using 0.929 watt, while little cores, GPU, and DRAM consume 0.155, 0.055, and 0.096 watt, respectively. Note that big cores consume quite large power even in

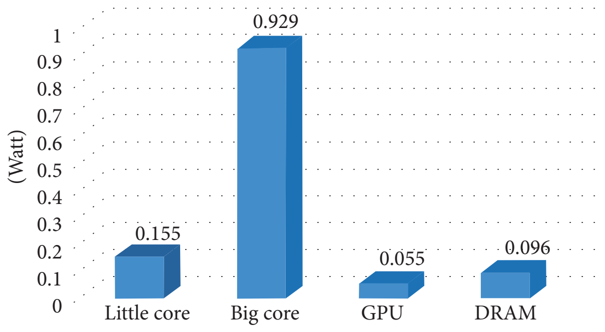


FIGURE 6: Power measurement results under the idle state on the Odroid XU3 device.

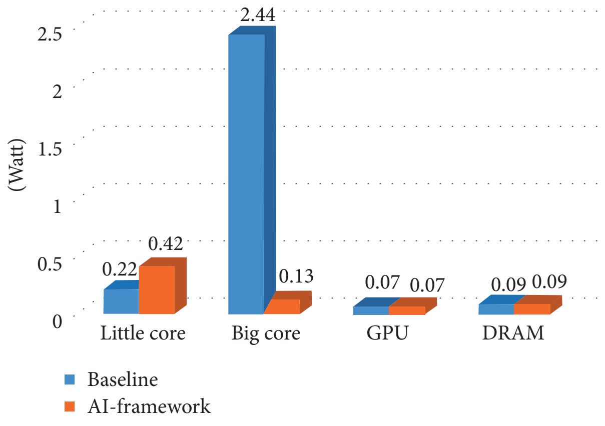


FIGURE 7: Power consumption comparison under the video recording workload on the Odroid XU 3 device.

an idle state, demonstrating the importance of the DPM and DVFS techniques.

To evaluate the power consumption reduced by our proposed framework, we execute the video recording workload under the two configurations, as shown in Figure 7. The first configuration is the baseline where any low-power

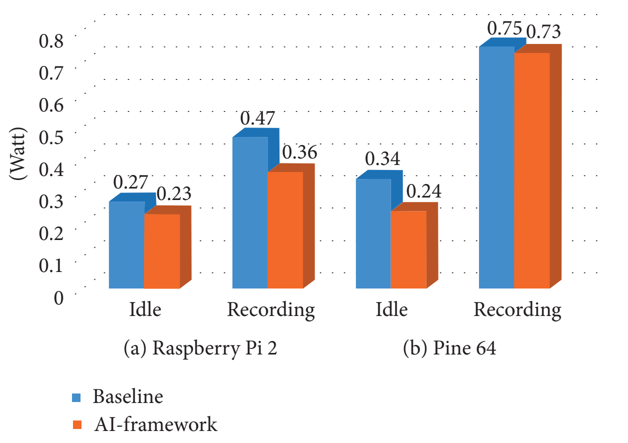


FIGURE 8: Power consumption comparison on the Raspberry Pi 2 and Pine 64 device.

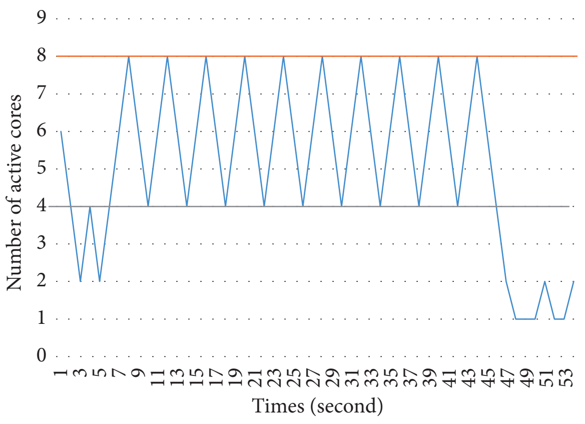


FIGURE 9: The number of active cores when we execute the Sunspider workload on the Odroid XU3 device.

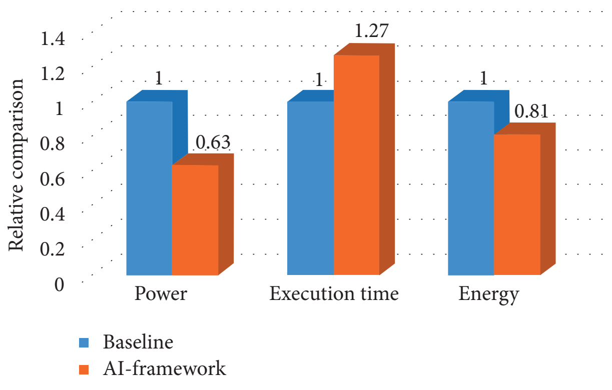


FIGURE 10: Power consumption, execution time, and energy saving comparison using the Sunspider workload on the Odroid XU 3 device.

technique is not applied. The second configuration is under our framework, labelled as AI-framework (Adaptive and Integrated framework) in the figure.

Since the video recording is not a CPU intensive workload, it uses only one core during most of its execution period. Hence, our framework turns on one little core, while turning off other cores. On the contrary, in the baseline, all cores are powered on and the workload mostly runs on one of the big cores. As a result, the power consumed by big cores becomes 2.44 watt in the baseline, while that consumed in the AI-framework is 0.13 watt due to offlining. The power consumed by little cores is 0.42 watt in the AI-framework, higher than that in the baseline. The overall power consumed by all cores in the baseline is 2.66 watt while that consumed in the AI-framework is 0.55 watt (79% reduction).

Figure 8 presents the power measurement results using Raspberry Pi 2 and Pine 64 under the idle and the video recording case. Note that these devices provide the frequency change functionality only, not supporting the dynamic power off functionality for an individual core. Hence, in this experiment, the AI-framework exploits the DVFS technique only.

The results show that our proposed AI-framework can reduce the power consumption by decreasing CPU frequency appropriately. For Raspberry Pi2, it reduces the power consumption from 0.27 to 0.23 watt under the idle state and from 0.47 to 0.36 watt when we run the video recording workload. For Pine 64, it reduces from 0.34 to 0.24 watt and from 0.75 to 0.73 watt, respectively. The reduction is relatively low for the video recording workload on the Pine 64 device. We conjecture that the camera module equipped in the Pine 64 device consumes a large portion of power consumption, leading to this small difference. We leave the component-level fine-grained power analysis as the future work.

4.2.2. Sunspider Workload Results. Figure 9 presents the number of active cores that are powered on during the execution period of the Sunspider workload in the AI-framework on the Odroid XU3 device. It shows that when the workload requires a large computing resource, the number of active cores increases up to the maximum cores. When the workload does not need that much computing resource, the active cores decrease down to the one core. It reveals that our framework indeed supports adaptability according to the workload characteristics.

Figure 10 presents the power consumption, execution time, and consumed energy when we run the Sunspider workload under the baseline and AI-framework. Note that the y-axis is the relative value. The power consumed in the baseline is 2.6 watt while that in the AI-framework is 1.6 watt (37% reduction).

However, the execution time of the workload in the baseline is 39 seconds while that in the AI-framework is 49 seconds (27% degradation). It shows the tradeoff of the low-power techniques, reducing power consumption at the expense of performance drop. As a net result, the AI-framework can achieve the 19% energy saving, reducing from 1.97 to 1.6 joule. For a CPU intensive workload, we can mitigate the performance drop by turning off cores conservatively, which will be further discussed in the next section.

We also run the Sunspider workload on the Pine 64 and Raspberry Pi 2 device. The results show that even though the

Table 2: Power consumption and detection latency on the Pine 64 and Raspberry Pi 2 device.

Device	Configuration	Power (watt)	Latency (ms)
Pine 64	AI-framework	2.47	122.7
	Baseline 1: max freq	3.18	121.6
	Baseline 2: min freq	2.07	275.0
Raspberry Pi2	AI-framework	1.34	239.1
	Baseline 1: max freq	1.41	230.5
	Baseline 2: min freq	1.28	338.1

Table 3: Power consumption and detection latency on the Odroid XU3 device (max utilization: 80%, min utilization: various).

Device	Configuration	Power (watt)	Latency (ms)
Odroid XU3	AI-framework: min util = 0%	5.90	251.7
	AI-framework: min util = 10%	5.74	264.8
	AI-framework: min util = 30%	4.38	267.4
	AI-framework: min util = 60%	3.13	425.2

AI-framework provides better energy efficiency as discussed in Figure 10, the improvement is small, ranging from 1% to 6%. Our analysis reveals that since the Sunspider workload is CPU intensive, requiring more than 4 cores on average, the AI-framework does not have enough chance to apply DVFS. Note that these two devices have 4 cores, as explained in Table 1.

4.2.3. Pedestrian Detection Workload Results. We measure the power consumption and average detection latency when we execute the pedestrian detection workload on the Pine 64 and Raspberry Pi2 device, presented in Table 2. Since we can utilize DVFS only in these devices, we conduct experiments under three configurations. In the baseline 1, we configure all cores to run at the maximum frequency (1152 MHz for Pine 64 and 900 MHz for Raspberry Pi2). In baseline 2, all cores are configured to run at the minimum frequency (480 MHz for Pine 64 and 200 MHz for Raspberry Pi2). On the contrary, in the AI-framework, the frequency of a core is changed adaptively based on the current CPU utilization and the min/max utilization threshold (20% and 80% in this experiment).

Experimental results show that the AI-framework balances well between the power consumption and performance. Baseline 1 provides the best performance at the cost of high power consumption. On the contrary, baseline 2 reduces power the most but gives a noticeable impact on performance. However, our framework can reduce the power consumption (22% reduction for Pine 64 and 5% reduction for Raspberry Pi2) while hardly affecting the performance of the workload.

Table 3 shows the results when we execute the pedestrian detection workload on the Odroid XU3 device. In this device, the AI-framework can utilize not only DVFS but also DPM. Therefore, we conduct experiments with four different *min_utilization* threshold values that trigger the policy manager in our framework as discussed in Section 3.4. When the threshold becomes smaller, the AI-framework tries to apply low-power techniques conservatively, while applying techniques aggressively as the threshold becomes larger.

When the min utilization threshold is set as 0%, the AI-framework tries to decrease computing resources when the current utilization is less than 0%. It means that the AI-framework does not apply DPM and DVFS, turning on all cores with maximum frequency, which provides the best performance and the worst power consumption in this device (baseline configuration). When the threshold is 10%, the AI-framework tries to decrease computing resources conservatively, obtaining relatively small power reduction (from 5.9 to 5.74 watt in this case). On the contrary, when the threshold is 60%, it tries aggressively, yielding better power reduction at the cost of latency. These results reveal the tradeoff between power reduction and performance. By setting the threshold appropriately (30% in this case), the AI-framework can reduce the power consumption without considerable performance degradation.

5. Conclusion

In this paper, we design a new low-power framework for multicore mobile devices. It integrates both DPM and DVFS techniques and applies them adaptively according to the workload characteristics and device features. Real implementation based experiments show that the proposed framework balances well between the power consumption and performance, resulting in the energy saving.

We will extend our work into the two directions. First, we investigate the performance drop, especially for a CPU intensive workload observed in Figure 10, using hardware-level performance monitoring unit supported by processors. We conjecture that workload-aware fine-grained power management can alleviate the drop while maintaining the power reduction benefit. The second direction is developing a what-if engine that can predict how an alteration of frequency or number of active cores influences energy efficiency in advance using our framework.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The present research was conducted by the research fund of Dankook University (BK21 Plus) in 2014 and by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-R0992-16-1012) supervised by the IITP and by the Center for Integrated Smart Sensors funded by the Ministry of Science, ICT & Future Planning as Global Frontier Project (CISS-2-3).

References

[1] N. D. Lanez, S. Bhattacharyaz, P. Georgievy, C. Forlivesiz, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the ACM International Workshop on Internet of Things towards Applications (IoT-App '15)*, November 2015.

[2] J. S. Kim, D. H. Yeom, and Y. H. Joo, "Fast and robust algorithm of tracking multiple moving objects for intelligent video surveillance systems," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 3, pp. 1165–1170, 2011.

[3] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2, 2004.

[4] C. Baun, "Mobile clusters of single board computers: an option for providing resources to student projects and researchers," *SpringerPlus*, vol. 5, no. 1, article 360, 2016.

[5] T. Guan, Y. Wang, L. Duan, and R. Ji, "On-device mobile landmark recognition using binarized descriptor with multi-feature fusion," *ACM Transactions on Intelligent Systems and Technology*, vol. 7, no. 1, article 12, 2015.

[6] M. Zhu and K. Shen, "Energy discounted computing on multicore smartphones," in *Proceedings of the USENIX Annual Technical Conference (ATC '16)*, Denver, Colo, USA, June 2016.

[7] A. Carroll and G. Heiser, "Unifying DVFS and offlining in mobile multicores," in *Proceedings of the 20th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS '14)*, pp. 287–296, April 2014.

[8] Y. Tawara, A. Idehara, and H. Yamamoto, "DVFS and power-off controls on a multicore operating system," in *Proceedings of the 10th International Forum on Embedded MPSoC and Multicore (MPSoC '10)*, Gifu, Japan, June 2010.

[9] J. M. Kim, Y. G. Kim, and S. W. Chung, "Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 286–292, 2015.

[10] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 3s, article 111, 2014.

[11] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, 2000.

[12] M. E. Salehi, M. Samadi, M. Najibi, A. Afzali-Kusha, M. Pedram, and S. M. Fakhraie, "Dynamic voltage and frequency scheduling for embedded processors considering power/performance tradeoffs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1931–1935, 2011.

[13] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C.-J. Wu, "A study of mobile device utilization," in *Proceedings of the 15th IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '15)*, pp. 225–234, March 2015.

[14] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao, "Towards better CPU power management on multicore smartphones," in *Proceedings of the ACM Workshop on Power-Aware Computing and Systems (HotPower '13)*, Farmington, Pa, USA, November 2013.

[15] W. Seo, D. Im, J. Choi, and J. Huh, "Big or little: a study of mobile interactive applications on an asymmetric multi-core platform," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC '15)*, pp. 1–11, IEEE, Atlanta, Ga, USA, October 2015.

[16] Z. Mwaikambo, A. Raj, R. Russell, J. Schopp, and S. Vaddagiri, "Linux kernel CPU hotplug support," in *Proceedings of the OLS*, July 2004.

[17] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Ottawa Linux Symposium (OLS '06)*, Ottawa, Canada, July 2006.

[18] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the USENIX Conference on USENIX Annual Technical Conference (USENIXATC '10)*, ACM, Boston, Mass, USA, 2010.

[19] A. Carroll and G. Heiser, "Mobile multicores: use them or waste them," in *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower '13)*, November 2013.

[20] Q. Zhu, M. Zhu, B. Wu, X. Shen, K. Shen, and Z. Wang, "Software engagement with sleeping CPUs," in *Proceedings of the 15th Workshop on Hot Topics in Operating Systems (HotOS '15)*, Kartause Ittingen, Switzerland, March 2015.

[21] W. Song, N. Sung, B.-G. Chun, and J. Kim, "Reducing energy consumption of smartphones using user-perceived response time analysis," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (HotMobile '14)*, ACM, February 2014.

[22] J. Wamhoff, S. Diestelhorst, C. Fetzer, P. Marlier, P. Felber, and D. Dice, "The TURBO diaries: application-controlled frequency scaling explained," in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '14)*, June 2014.

[23] M. Chiesi, L. Vanzolini, C. Mucci, E. Franchi Scarselli, and R. Guerrieri, "Power-aware job scheduling on heterogeneous multicore architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 868–877, 2015.

[24] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: a survey," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 1, article 7, 2016.

[25] S. Li and F. Broekaert, "Low-power scheduling with DVFS for common RTOS on multicore platforms," in *Proceedings of the 3rd Embedded Operating Systems Workshop (EWiLi '13)*, Toulouse, France, August 2013.

[26] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich, "Energy management in mobile devices with the Cinder operating system," in *Proceedings of the 6th ACM EuroSys Conference on Computer Systems (EuroSys '11)*, pp. 139–152, April 2011.

[27] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser, "Koala: a platform for OS-level power management," in *Proceedings of the EuroSys*, March-April 2009.

[28] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power containers: an OS facility for fine-grained power and energy management on multicore servers," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*, pp. 65–76, ACM, March 2013.

[29] Y. Kwon, S. Lee, H. Yi et al., "Mantis: efficient predictions of execution time, energy usage, memory usage and network usage on smart mobile devices," *IEEE Transactions on Mobile Computing*, vol. 14, no. 10, pp. 2059–2072, 2015.

[30] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery: analyzing mobile browser energy consumption," in *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*, pp. 41–50, ACM, Lyon, France, April 2012.

[31] D. H. Bui, Y. Liu, H. Kim, I. Shin, and F. Zhao, "Rethinking energy-performance trade-off in mobile web page loading," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*, pp. 14–26, ACM, Paris, France, September 2015.

[32] http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127.

[33] https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[34] https://www.pine64.org/.

[35] CPU hotplug Support in Linux Kernel, https://lwn.net/Articles/537570/.

[36] J. Resig, "JavaScript Performance Rundown," http://ejohn.org/blog/javascript-performance-rundown/.

[37] P. I. Wilson and J. Fernandez, "Facial feature detection using HAAR classifiers," *Journal of Computing Sciences in Colleges*, vol. 21, no. 4, pp. 127–133, 2006.