The Thesis Committee for Sarbartha Banerjee
Certifies that this is the approved version of the following Thesis:

Characterization of Smartphone Governor Strategies and Making of a Workload Aware Governor

APPROVED BY
SUPERVISING COMMITTEE:

Lizy K. John, Supervisor

Mohit Tiwari

Characterization of Smartphone Governor Strategies and Making of a Workload Aware Governor

by

Sarbartha Banerjee

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2018

Characterization of Smartphone Governor Strategies and Making of a Workload Aware Governor

Sarbartha Banerjee, M.S.E

The University of Texas at Austin, 2018

Supervisor: Lizy K. John

This thesis presents the importance of workload characterization towards governing the operational voltage and frequency of a smartphone processor by running a series of workload on an ARM v8 processor. The idea of finishing a task as fast as possible to return to idle state(race-to-idle) versus the idea of choosing the correct frequency for time deltas(pace-to-idle) is studied in detail. Android governors either statically use a single frequency for the entire active time or determines the voltage and frequency dynamically based on the load average on the processor. Similar load averaging strategies are used for other blocks in SoC (System on Chip) like the GPU or the media processor. However, the different blocks of a SoC draw power from the same current source. Owing to lack of fine-grained workload characterization, the power is redirected to the not-so-important unit providing poor performance and energy efficiency. The behavior of different existing governors is explored by running on a variety of workload and analyze the optimal strategy for energy efficiency satisfying an acceptable user performance. Crucial traits of active user applications are inferred from scheduler to fine tune the optimal voltage and frequency across different blocks under constrained power source to build a system-wide governor.

Table of Contents

List of Tables

List of Figures

List of Illustrations

Chapter 1: Introduction

Energy efficiency and power consumption have become the major design criteria for modern smartphones in addition to desirable performance. This is primarily because battery technology development has been much slower as compared to processor development, the form factor of the phones limiting the battery capacity and the stringent thermal limit of the chip. To address this issue, all modern smartphones have multiple DVFS (Dynamic Voltage Frequency Scaling) modes to manage its energy resources. In typical DVFS, the frequency and the voltage of the processor is modified based on the performance requirement. The DVFS modes are also tuned based on the state of the battery. This technique aims to meet performance requirements of a task giving a jitter-free user experience. Modern System-on-chip(SoC) has additional accelerators like graphics processing unit(GPU), camera unit, multimedia unit, modem block, audio processor etc. in a single chip. Each of these components might have their own DVFS modes. Tuning the frequency of each of these units effectively not only saves power but also increases performance in certain scenarios of constrained maximum power and thermal limits. In smartphones, sometimes a single power source is shared among various components. Prioritizing performance to the active application during user interaction rather than spending it in other units which might be used in some other background applications can improve user experience.

The availability of a high number of DVFS choices and the availability of DVFS in multiple components in a SoC makes decision making even harder when an application intermittently uses multiple components. Along with dynamic power, static power has also become an important contributor to system power consumption in smaller technology nodes. Therefore, minimizing the chip active time can by finishing the task faster can improve the energy efficiency.

One simple DVFS policy runs the job on the target system at the maximum possible frequency and then throttle down to minimum or deep-sleep state as quickly as possible. This method is typically termed as *race-to-idle*. This method is simple, reduces latency and saves energy in certain use cases. The energy saving comes from the fact that the processing unit is active for the minimum amount of time and the leakage power is saved. Moreover, the DVFS driver is simple and refrains from taking complex decisions saving power wasted by kernel code in determining the DVFS mode. But, its validity and usefulness is yet to be conclusively established for smartphones as workloads are interactive in nature and incurs high IO latency waiting for user inputs. Keeping the processor at highest frequency for these intervals not only consume battery but also heats up the device bringing thermal throttling more frequently. More complex methods can optimally configure the processor into one of the multiple available power states (DVFS points) adapting to the overall load average of a resource. But in such cases, though dynamic power consumption is reduced, consumption of leakage power might be increased. Moreover, heterogeneous clusters in multicore processors and various other components like the GPU have independent DVFS points which make the optimal choice even more difficult.

Moreover, there are situations when the smartphone is running on low battery. Normally, the frequency of all the blocks are toned down to consume less energy. But the increasing leakage current raises the question whether it really increase the energy efficiency when an application is run on the system at lower frequency?

Thus, the need of understanding the workload while choosing the governor is becoming essential. At least if one can classify the workload and figure out the functional units that need to be used, it will greatly help in determining the governor to be used for the individual units. Also, most of the governors are designed for the CPU. But global decision of the various DVFS modes based on the workload improves the energy efficiency with more performance and less heat up of the smartphone system-on-chip.

Our study encompasses the analysis of various categories of governors for different kind of workloads to explain the optimal strategy in a mobile platform. The race-to-idle strategy works best for servers where the quality of service and latency of the request are important. Nevertheless, it is entirely different in a battery-operated device where user experience goes together with the energy consumed and the maximum power that can be delivered by the current source. It also depends on state of charge of the battery making the choice more complicated. It is seen that if a compute intensive workload is run on the device at a lower frequency on low battery, the power consumed by the device is low but the overall energy consumption is more with providing the user with a glitch performance. Hence, it may not be wise to always run the processor at lower frequency when the battery is low. Again, workload characterization and understanding its status in the run queue will help us take smarter governor decisions.

Designing a governor is an optimization problem. Some power-hungry governors are good in performance while some relaxed governors might be power saving. With the availability of multiple DVFS modes, finite DVFS switching time and workload detection, researchers are coming up with improved governors that predict the pattern of the workload [18] and choose the appropriate DVFS point. Moreover, care should be taken that the governor process itself doesn't slow down the system and try to be less invasive and power hungry as possible.

With the increase in the core power and hard power limits, the need for choosing the optimum DVFS mode has become more important. The TDP of mobile devices are often less than the maximum capability of the multi-core processor and hence detecting phase changes to re-balance the power budget is essential for maximum performance and better power efficiency.

In this work, exhaustive comparison is performed between the performance and energy efficiency across race-to-idle and pace-to-idle governor strategies and categorized scenarios where one scheme is superior to other which exemplifies the need for workload characterization for choosing the optimum DVFS point faster and utilize fine-grained idle opportunities during process run. Next, Linux scheduler is scavenged to identify process traits which will lead to detecting these scenarios which can be used by a governor to make smarter choices for choosing a better DVFS point. And finally, the winteractive governor is introduced. The winteractive governor is a workload aware enhancement of the widely used interactive governor which uses scheduler task state to determine the DVFS point. The conventional interactive governor relies on the load average to decide the DVFS mode but the winteractive governor looks at the state and priority of the user application along with the overall available power budget of the system to decide the DVFS point.

Chapter 2: Background

In this section, some of the existing governor strategies are explored for smartphone CPUs and GPUs. The mobile SoC comprises of multiple units and each of them run at a separate clock and voltage having their own governor. For instance, in a big-LITTLE mobile processor, each of the CPU clusters run on a separate voltage/frequency point while the graphics unit can run at a third voltage while sharing the same current source. One can also change the frequency of the BUS connecting the DDR.

With the demand of new aggressive power saving techniques, designers have added more voltage/frequency points to one unit and added governors for independent control of more and more units. Power can be saved if the required unit is enabled at the appropriate frequency. But switching the DVFS modes consumes energy and has non-zero latency. Hence, too much switching is also not desirable. In addition, every unit can also be separately put in the different idle power modes like clock gating, retention or deep sleep. All these low power modes have different wake up latency and leakage current consumed. But in this thesis, the main concern is the accurate determination of the active DVFS mode. Let us consider the different existing governors and try to draw inferences on when each of them gives the best performance and/or power efficiency.

2.1 RACE-TO-IDLE AND PACE-TO-IDLE SCHEMES

Before understanding the basic android governors available in the system, first let us elaborate on the different type of governor strategies. Once there is an active task in the run queue, the race-to-idle schemes boost up the resource to its maximum frequency irrespective of the merit of the process and sending the resource back to idle as fast as possible once the task is completed. A pace-to-idle scheme on the other hand interactively monitors the behavior of the process to fine tune the frequency based on the requirement of the resource. A lot of research is being conducted to compare these two types of governor strategies. Some works suggested pace-to-idle strategy is the better strategy [6] due to the intermittent CPU usage pattern of the workloads and is power efficient. But with the shrinking transistors, leakage current is becoming comparable with the dynamic current, race-to-idle schemes are becoming popular. Race-to-idle schemes not only gives better performance most of the times but also keeps the chip active for the minimum amount of time. After performing experiments on a wide variety of workloads, it is found that characterization of a workload identifying the unit it is going to use is very important. Race-to-idle strategy can then be applied on that critical resource to get the best performance. If the workload requires a lot of user input a pace-to-idle strategy works better as it can adapt itself to increase frequency when the performance is required saving energy in idle intervals. The race-to-idle strategy also brings in thermal throttling or performance reduction due to current clipping. Thus, in a power constrained system, race-to-idle might not give the best performance. Nevertheless, it is easy as task monitoring consumes useless power and energy and is also skeptical in boosting the frequency when it is urgently needed. Moreover, the pace-to-idle schemes apply hysteresis timer after the run queue is empty to grab potential opportunity of a refilling of the run queue which often wastes power in the maximum frequency delayed the return to idle mode.

With thinner smartphone form factors and powerful processors, the current source cannot support the power required to operate at maximum frequency for all the processors in a system. Pace-to-idle works good for multi-core tasks as it hits the power and thermal limit less frequently. But periodic monitoring and changing the frequency and voltage halts the processor refraining it from doing useful work. A lot of orthogonal research is being done to reduce the voltage and frequency switching time. Reduction of switching time will give governor designers more confidence in changing the frequency to suit the workload requirement. Several researchers have proposed elegant methods [13,14] to reduce the switch time. But still the DVFS switching time is high as it involves changing the voltage which is slow. So, characterization of workloads and identification of phases based on usage pattern can reduce the number of DVFS mode changes and will increase efficiency. The length of the phases can help identify the time duration when the system needs to be at a specific frequency. So, it is necessary not only to decide the correct frequency but also the sampling time of the governor.

Finally, several resources in a SoC uses the same power source. A good governor will not only focus on changing the CPU DVFS modes but will also consider optimizing the frequency of other units in the SoC to increase both performance and energy efficiency. Designing an overall governor for the entire SoC coupled with workload characterization is the best solution in a smartphone system where multiple components are sharing the same current source. The scenarios which push the current source near the limit makes designing good governors highly essential not only to distribute power across units but also to produce the best performance. Sometimes, in case of power crunch, the unit can be boosted by borrowing power from other resources [21] to deliver better performance.

Since, there is no clear answer if the race-to-idle is a better scheme or a pace-to-idle, our study will show that processor should perform workload characterization to understand race-to-idle opportunities for both performance improvement and energy efficiency in certain cases and at the same time choose intermediate frequencies to prevent hitting the power and the thermal limit and increase energy efficiency in most cases.

2.2 CPU Governors

A brief overview of the types of CPU governors present in the Linux kernel of an android smartphone today are explained:

2.2.1 Performance Governor:

This governor is a constant frequency governor which keeps the system in highest possible voltage and frequency irrespective of the workload. This is highly power hungry and latches itself to maximum frequency. This governor works best when a series of compute intensive job is run in the system. It also has zero latency when serving application launch which incurs minimum latency when the CPU is at its highest frequency. Moreover, it also keeps the bus to DDR at its peak frequency. It doesn't waste extra time and power in DVFS switching. But keeping the processor in this governor can cause thermal throttling and unnecessary running the system near the peak current of the supply. It is considered as a ‘race’ governor as it finishes the job as quickly as possible and goes to idle. It consumes more power than all the other governors but might not provide the best performance always due to chances of thermal throttling and in some applications which are purely accelerator dependent when the accelerator might not receive the entire power due to limited power budget.

2.2.2 Ondemand Governor:

The ondemand governor [2] switches the system in highest possible voltage and frequency whenever a job is scheduled and immediately ramps down gradually to lower frequency when there are no more pending active jobs in the run queue of the scheduler. This works well when there is a sequence of compute intensive jobs interspersed with long delays which matches with user input dependent workloads prevalent in smartphones. The immediate return to low frequency ensures that it spends minimum time in the performance mode and least energy usage. However, if the idle time between jobs is very low, this governor hops between frequencies repeatedly, making it a bad choice. This behavior is typical for games which need interleaved CPU and GPU interactions. Moreover, it performs poorly neither saving power nor improving performance in benchmarks that have jobs with equal compute and memory intensity. The memory fetch window will immediately cause the processor to change frequencies when it could have stayed in performance mode as the task state changes from RUNNING to INTERRUPTIBLE. This governor can be considered as a pace' governor which will adapt the frequency based on the workload requirement.

2.2.3 Interactive Governor:

The problem of undesirable switching by the ondemand governor is solved by the interactive governor. It works like the Ondemand governor with the exception that a hysteresis is added before ramping down the frequency. This is the most common governor used all over Linux kernels. It is also a pacing governor. The frequency range along with the hysteresis delay can be programmed to suit the workload requirement. Moreover, it doesn't immediately switch to the highest frequency but ramps up in steps from an intermediate frequency. The delays in each DVFS mode is programmable. One of the drawbacks of interactive governor is the delay to switch to the highest frequency. Though it performs well in interactive tasks which doesn't always demand the highest frequency and it checks for the load average to figure the best frequency but lags the performance governor during application launch. In Table 1, the number of DVFS switches are listed for some of the benchmarks:

Benchmarks	**Performance Governor**	**Interactive Governor**	**Ondemand Governor**	**Powersave Governor**
Antutu	18	842	3809	0
applaunch	0	1897	7463	0
audio	0	43	112	0
Dhrystone	0	8	12	0
Geekbench	0	229	887	0
homescreen	0	44	51	0
linpack	0	31	83	0
memcpy	0	10	12	0
Nenamark	0	1178	8383	0

Table 1: Number of DVFS switching for different governors.

It is noticeable that the interactive governor hysteresis filters out some of the DVFS switching as against the ondemand governor. One more interesting observation is that Antutu task thermally throttled the system and hence some DVFS mode toggling is observed. The thermal throttling happened when Antutu started running graphically intensive work when the CPU is at the highest frequency.

2.2.4 Powersave Governor:

Powersave governor is designed to save energy. This gives slow response but this governor is highly power efficient and is often used when battery is low or during thermal throttling. It also gives good performance when the application is using another component of the SoC like the GPU. But this governor might end up consuming more energy preventing the CPU to reach the idle state for a longer period. Though the powersave governor is more focused on increasing the power efficiency, that is always not the case. If this governor is active during a CPU intensive work, it not only degrades the user experience but in some cases, can end up consuming more energy by keeping the CPUs active for a longer period. It is also true that in some cases the powersave governor might give good performance. If the workload is not using the CPU and is IO intensive, like a game using GPU or a multimedia workload, it is better to bring the corresponding peripheral to turbo mode while keeping the CPU in powersave as it will not trigger the thermal throttler and provide adequate power budget to the GPU to perform as fast as possible. During this study, it is observed that only changing the CPU governor did not necessarily give the best result. This motivated us to explore the governors and frequencies of other components like the GPU, multimedia unit or the bus.

2.3 GPU Governors

Most of the chips have GPU as a proprietary unit, so the governors supported are specific to the hardware used in the experiment. Since our test setup had a Qualcomm Snapdragon processor, a few GPU governors are listed.

Most of the fancy governors are largely pacing governors whose performance lie between the performance and the powersave governors. Mentioned below are a few GPU governors which are custom build. Some of these are aggressive in switching DVFS modes while others are more relaxed.

2.3.1 MSM-Adreno-tz Governor:

The default GPU governor used by Qualcomm for their Adreno GPUs. It is based on the ondemand governor but is biased towards performance, hence it gives better performance in games but less battery life. But this governor snoops for the load average on the GPU. But it doesn’t go to the lower frequency even on idle.

2.3.2 Adreno Idler:

This is less aggressive than the msm-adreno-tz. It uses an idling-algorithm on top of the default governor. It tries to keep the GPU frequency near the idle frequency. Though it is more energy efficient but suffers from lags as it never reaches the highest frequency.

2.3.3 Performance Governor:

This governor keeps the GPU running at the max frequency. This is a governor to use for the best possible experience in games but is poor in energy efficiency.

2.3.4 Powersave Governor:

Like the CPU governor, this keeps the GPU running at the lowest possible frequency. This approach usually gives long battery life, but faces extreme lag in games. It can be used in cases when the application is not demanding high graphics.

2.4 DVFS Modes

Owing to the need to save power and to provide flexibility to choose the appropriate mode to perform a task, hardware designers provide several DVFS points. Our testing platform is a Dragonboard 410c [15] consisting of a Qualcomm Snapdragon 410 processor having Quad-core ARM A53 processor with all four cores running at the same voltage & frequency. The cores can be independently put into low power mode but they cannot be run at different frequency. This Snapdragon processor supports 5 different frequency points. The frequencies are 1.2GHz, 1.15GHz, 1.09Ghz, 998MHz and 800MHz.

These frequency points are chosen by selecting different PLL multiplier to the base crystal frequency of 19.2MHz. The governors should be designed to take advantage of these points to perform optimally and efficiently. Apart from that the DDR also has different frequencies available like 533, 400 or 200 MHz [22]. Either it can be scaled differently or in most of the existing governors it is scaled based on the CPU frequency. Similarly, the GPU has its own independent DVFS modes but shares the same power rail as the CPU and others.

Mobile applications mostly stress a single processing unit. So, choosing the optimum DVFS points for that specific block is necessary. Since the same power source is used to power multiple components of the chip, there are scenarios which can yield poor performance if the wrong unit is in high performance mode. Choosing wrong DVFS points for individual component may push the overall current drawn to be close to the power source limit which will adversely affect performance. For instance, if there are a lot of IO operation or if a multimedia application is running, keeping the CPU in performance mode will allocate a larger power budget from the current source and the multimedia will simply perform poorer due to lack of power budget for this unit. If the CPU is kept in powersave mode, enough power is not redirected to the CPU during compute intensive operation and it ends up consuming more energy. Same is also true is case of multi-core applications as often the power source cannot deliver the peak power of all the CPUs together and if the CPUs are kept at the maximum frequency, maximum power limit is reached and there is reduction in performance either by thermal throttling or by global power management which will uniformly allocate less power budget for all resources.

Some of our observations in Geekbench 3 memory bandwidth tests show the importance of the DVFS point selection. As against the memcpy vector, which only does memory operations, the Geekbench 3 memory bandwidth tests perform computations on the data brought from the memory. The performance governor shows drastic reduction in memory bandwidth as it reaches the peak current of the power source. Since the compute and memory operations are complimentary to each other, the interactive governor either increases the CPU frequency or does the memory operations using power budget effectively. The drastic difference in the bandwidths are shown in Figure 1.

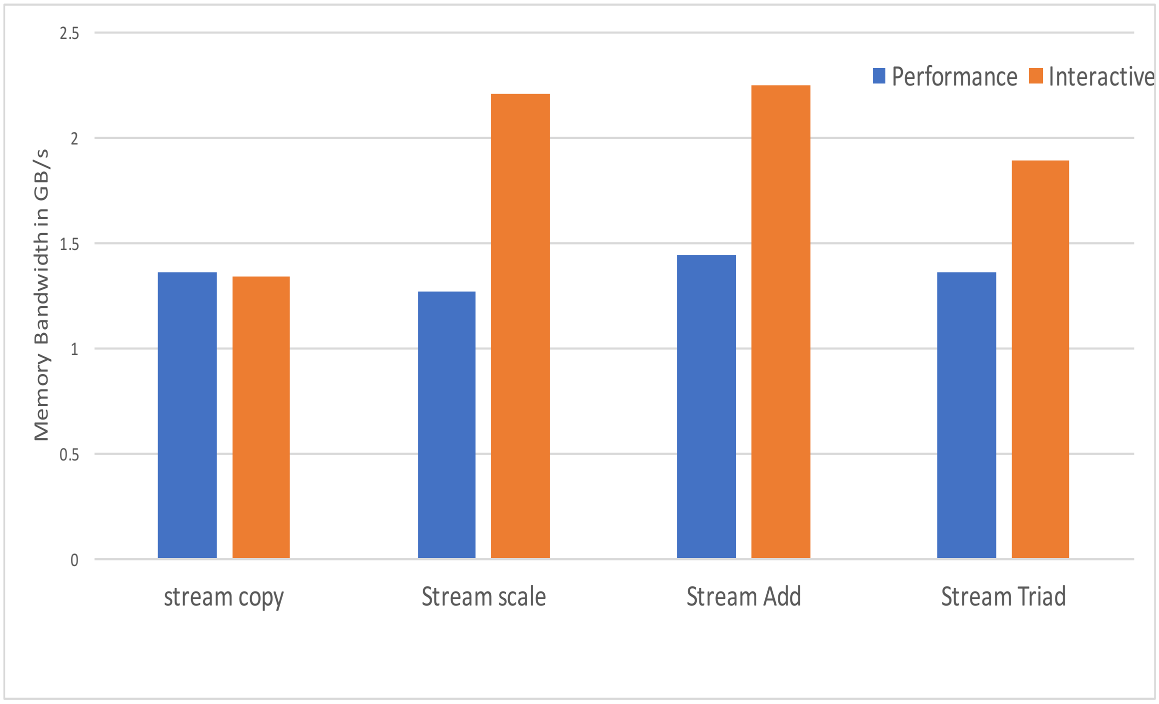

Figure 1: Geekbench 3 Memory Bandwidth comparison between performance and interactive governors

The reduction of memory bandwidth for stream scale, stream add and stream triad is due to reaching the power budget limit. There is no such operation in stream copy and the bandwidth for both the governors are the same.

Similar observation is found in comparing the single core and multi core scenarios. The multi-core result should be theoretically four times on a quad core processor in an unconstrained system. Practically there is some reduction due to common bus being used and bank conflicts.

But the power constrained result show a much less increase in memory bandwidth in multicore results with interactive governor as shown in Figure 2. This exemplifies the requirement of choosing the correct DVFS point in a multi-resource power constrained system.

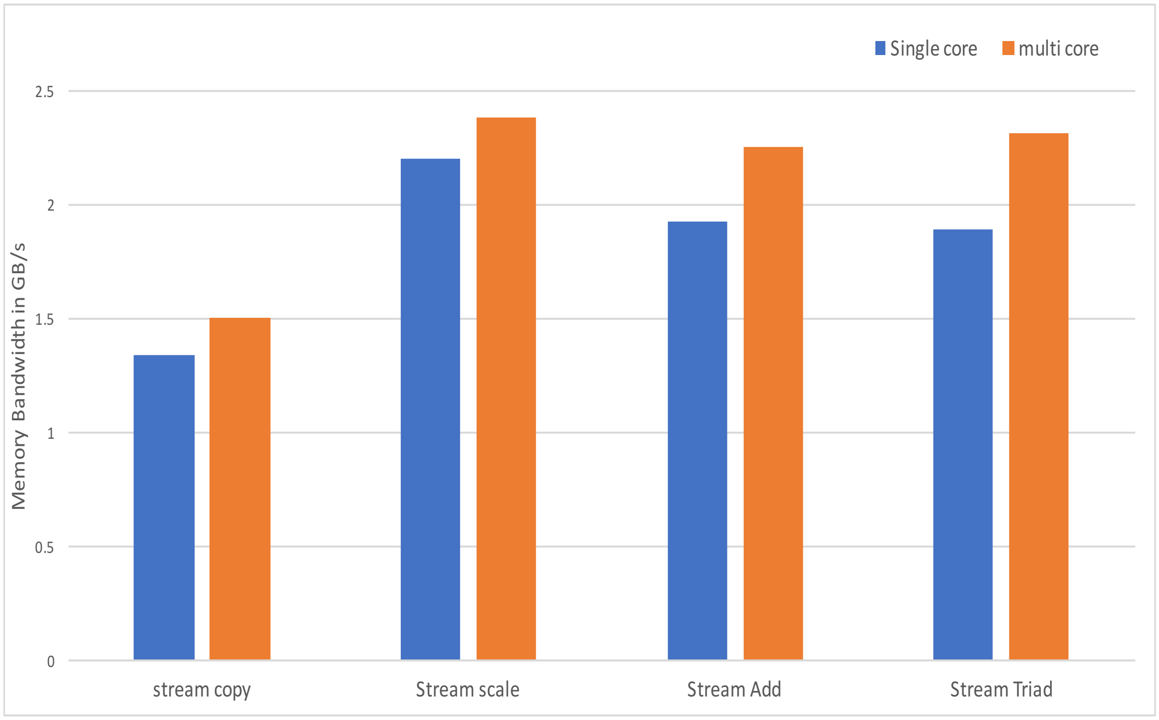

Figure 2: Geelbench 3 Memory bandwidth comparison for interactive governor for single and multi core

2.5 LINUX KERNEL AND ANDROID APPLICATION EXECUTION

This section describes the various components that are part of the android framework. Figure 3 shows the hierarchical view of the android system. The Four sections are described as the following:

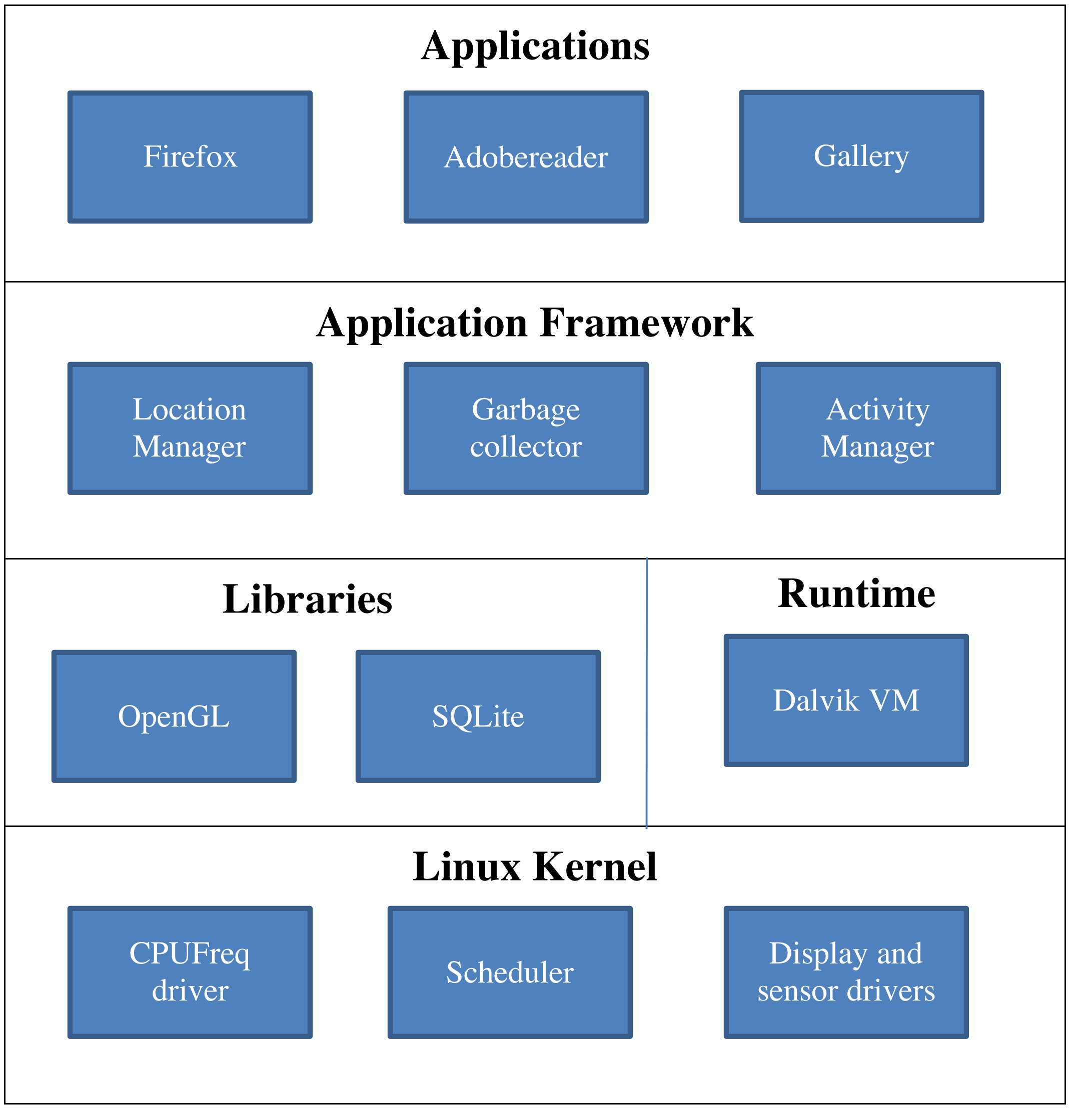

Figure 3: Hierarchical view of Android built on Linux Kernel

2.5.1 Linux Kernel:

On the base of every android device is a Linux kernel. Android 5.1.1 is used with Linux 3.10 kernel underneath. The kernel constitutes of the Linux task scheduler with all the different drivers which includes Bluetooth, camera, sensors etc. It also holds the CPUFreq driver which determines the DVFS modes. It holds programs for process and memory management, permissions and has access to the interrupts and the file system.

2.5.2 Libraries:

On top of the kernel, there resides the libraries. It handles all the GPU standard drivers like OpenGL, the media framework libraries for playing or recording audio/video. This layer also includes the different c/c++ libraries and database libraries like the SQLite and webkit for browsers.

2.5.3 Runtime:

Residing in the same layer is the android runtime(ART). Every application is run in its own sandbox on a virtual machine. Whenever, an application is launched, it runs on the dalvik virtual machine. The dalvik VM is a register based virtual machine that provides the necessary optimizations for running in a low memory environments. The applications running in the system are essentially java bytecodes. The dalvik compiler converts the bytecodes into dalvik executables which are optimized to run in smaller processors with limited memory. It takes advantage of the core features of Linux like multi-threading, process and memory management.

2.5.4 Application Framework:

This layer is built using Java and provides high level services and APIs (for example, notifications, sharing data, and so on) that are leveraged by the applications. The key services of the Android framework include: Activity Manager, Content Providers, Resource Manager, Location Manager, Notifications Manager, View System, and Telephony Manager.

2.5.5 User Applications:

On top of all the layers run the android platform stack which is comprised of native applications like calculator, browser, calendar and a host of third-party applications that are installable from play store. Launching applications launches a separate instance of virtual machine whose environment and system resource allocation is done by the zygote process discussed next.

2.5.6 The Zygote process

One of the important daemon running in the system is the Zygote and the Zygote64 processes. Whenever, the user launches an application, the Zygote or the Zygote64 processes are triggered to create the virtual machine and create a process with task_struct etc. Any user application will have Zygote as the parent process.

2.6 WORKLOAD BEHAVIOR FROM SCHEDULER

A nice way to identify the incoming workload can be detected from the scheduler and the run queue. The Linux scheduler has per process information that can help to determine certain traits of the workload.

Every active process in the scheduler has a task_struct that includes important information about the process. The behavior of the processes can be extracted from the task_struct. Some of the fields of this structure is mentioned below:

2.6.1 task->state:

The task state gives an idea if the task is currently running or queued in the run queue or it is interrupted and waiting for some IO or memory operation. The task->state also provides an idea if the process is stopped or is a parentless zombie. Task->state can provide crucial information to the governor about the run state of the process.

2.6.2 task->utime/stime:

Utime variable gives information about the amount of time the process has spent time in CPU executing instructions. Stime gives the amount of time the process was busy in performing system calls. Stime gives the amount of exception and traps executed and can also provide a measure of the amount of time the process was waiting for user input etc. During consecutive intervals, the ratio of utime to stime gives an idea of the CPU usage by the process. A running process doesn’t always ensure that the CPU is being used. When the running process has a high utime, it can be inferred that the process is performing a CPU intensive task.

2.6.3 task->rt_prio:

This gives the real-time priority of the process. This is useful in inferring the amount of time the task is really scheduled in the CPU. Sometimes the task is in ready state and the utime is high but the rt_prio is low. If this is the case, that means the process is not the primary contributor to the CPU usage.

2.6.4 task->comm:

A set of characters that provide the name of the task.

Monitoring the above fields of all the processes in the scheduler gives more fine-grained view of the different tasks running in the CPU which can help in taking the appropriate decision based on certain heuristics. For example, if a process has a high utime, a high rt_priority and is in running state, one can raise the CPU frequency immediately to the maximum without waiting for load average and then can adjust later. Again, if the process has a high stime and is in the running state with higher priority, the CPU can settle for an intermediate frequency to begin with and can then adjust itself based on load average.

But one of the major problems is that monitoring for all the processes in an android system to take a decision has diminishing returns and the amount of time/power spent by the governor process is prohibitive. One of the crucial aspect of governor design is that it must be non-invasive and should take a negligible time/power to execute. As the complexity of the governor decision algorithm is increased, the decision may be precise but it will lag and the opportunity will be lost as the window will be gone. Therefore, to make the decision time acceptable a subset of the entire process list should be used. The active user applications is used whose parent process is always the zygote or the zygote64. Thus, identifying the PID of the process and the parent PID is important to verify it as a user application. The below field of the task gives info of the process PID and its parent PID.

2.6.5 task->pid/ppid:

The task_struct is a list of fields for individual tasks. To know the process to which the task belongs, the task->pid field is being used. The ppid field gives the parent pid for the process. It provides information about the parent process as some tasks can belong to the child process.

2.7 LINARO WORKLOAD AUTOMATION

Workload automation(WA) is a framework for running benchmark and applications in real hardware devices. It is built on top of python and uses the android debug bridge (ADB) to interact with the hardware target device. With WA tool, one can not only mention the workload along with the specific governor but also modify the different tunables of the governor. Other capabilities include invoking the perf tool on top of the regular execution to collect performance counter data, instrument the benchmark or application and display the result for the execution phase only. Through the WA tool, a sequence of workloads is executed and different traces of automated user interactions on an application. It is very powerful and is used extensively across evaluation of workload behavior in this thesis.

Chapter 3: Experimental setup

Dragonboard 410c[15] is used for the analysis of energy consumption across various workloads and benchmarks. It contains a Qualcomm Snapdragon 410 consisting of Quad-core ARM Cortex A53 processors running Android 5.1.1. There are shunt registers provided on board[16] to check the incoming current to the processor. The reason of choice for this processor is its prevalence in value-tier market where a low-cost power supply is used giving and the processor must perform in a constrained power supply while giving the maximum possible performance. Below are some of the specifications of this processor are listed in Table 2.

CPU	4 x ARM Cortex A53 1.2GHz
CPU architecture	64 bit ARM V8 architecture
GPU	Qualcomm Adreno 306 400MHz
DSP	Qualcomm Hexagon DSP
Memory	1GB LPDDR3 533MHz
Storage	8GB eMMC
Video	1080p@30fps HD video playback

Table 2: Snapdragon 410 processor specifications

The points across the shunt resistor(R77) on the board are tapped and a INA219 current sensor is connected to measure the current. The output of the current sensor is sampled using an Arduino microcontroller to get the data. A block diagram of the setup is shown in figure 4.

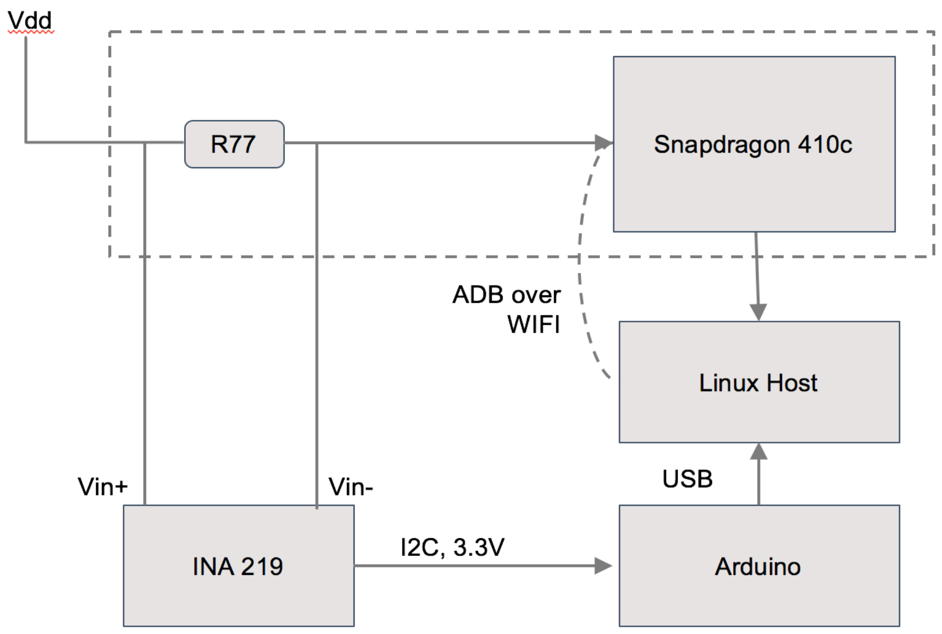

Figure 4: Block Diagram of experimental Setup

All the synchronization from the host side is done by the processing tool. The INA219 feeds current data to the microcontroller at 1KHz sampling frequency. The Arduino board averages the current values and send data via USB to the host every 500ms. This can be tuned to get the data at higher frequency. But 500ms seemed a reasonable reporting interval as some of the workloads take 200-300s to run. All the workloads start with a cache warmup phase and then the real application starts which is followed by a 5 second idle time.

The following parameters are tuned during the study of governor behavior:

- CPU governor
- Governor tuning
- DDR frequency
- GPU frequency

- Thermal throttler
- Hotplugging setting

The experiments are performed on four different governors which include *performance, interactive, ondemand and powersave governors*. The thermal throttling was switched on and relied only on the on-chip temperature sensor. The GPU frequency includes 400, 333 and 200 MHz.

The performance governor always ran at 1.2Ghz. The interactive governor is tuned to be power efficient. The intermediate frequency where the interactive governor immediately is not the highest frequency but an intermediate one at 994MHz. The processor runs at this frequency out of idle. it rechecks the load average after 80ms to decide to move up or stay in the same frequency or tone down. Moreover, it stays in the maximum frequency for 500ms for switching down to idle or lower frequency if the load average is lower than 85%.

Moreover, interactive governor has a feature to boost to the highest frequency on screen activity. This is disabled as it makes the interactive governor power hungry.

Chapter 4: Workload description

Some of the experiments for determining how each existing governor performs under different scenarios are mentioned in this section. The workloads are chosen to exhibit radically different type of behavior and is a mix of benchmarks and applications. While the benchmarks are run directly, the applications perform a sequence of operations automated by the workload automation tool. Table 3 and 4 lists the task performed by the benchmarks and the applications.

Benchmark	**Version**	**Description**
Antutu	5.3.0	Tests the CPU integer/floating point performance, memory performance, IO read/write and graphics performance.
Geekbench	3.4.1	Geekbench includes FP/integer performance, memory performance test. It also includes a mix of high MPKI vectors like nbody and lua and high IPC vectors like sgemm. It also includes memory tests.
BBench	3.0	bbench opens heavy preloaded webpage in the native browser with playing an audio in the background. It is a memory/IO intensive task.
Nenamark	2.4	OpenGL-ES 2.0 based graphics benchmark
hackbench	-	Runs a series of kernel tasks in the linux scheduler
ebizzy	-	ebizzy is a browser application with large working set and no locality. There are a lot of memory allocation with intermittent CPU intensive tasks.
linpack	1.2.9	Compute intensive using a lot of floating point instructions
caffeinemark	1.2.4	Series of tests measuring the speed of java programs. This is not memory or compute intensive but consists of a lot of syscalls.
dhrystone	-	Runs a tight loop having integer computations. highly compute intensive.

Table 3: Description of benchmark

Application	**Description**
applaunch	Launches either the calculator, browser or google Maps application when no other application is running in the system
multi_applaunch	Launches calculator, browser and maps application in a sequence on top of one another.
video	Playing a 720p video file in the native android video player.
audio	Plays an audio file in the native android audio player.
maps	Open google maps and perform a navigation task.
Adobereader	Scrolls, zooms and searches a word after opening a pdf file.
Facebook	Performs a series of tasks after logging in a facebook account including scrolling through the wall, like a friend's photo, post a status and comment on an existing post.
iozone	Performs a series of IO performance tasks
idle	keeps the processor in idle state to measure the leakage power.

Table 4: Description of Application

The group of benchmarks are chosen to cover the compute intensive, memory intensive and GPU intensive tasks. The applications are common user interactions that happen in a smartphone. Some of these benchmarks like the Antutu, Geekbench, Dhrystone and Nenamark directly reports the scores when run with different governors. For others, a fixed set of tasks is performed and the execution time is measured. Table 5 shows the performance variation across different governors when run on the given set of benchmarks and applications.

Workload	**metric**	**Governors**			
		performance	***interactive***	***ondemand***	***powersave***
Antutu	Score	19246	19038	19027	12201
Dhrystone	DMIPS	4053	4053	4052	2679
Linpack ST	Score (in MFLOPS)	176.864	175.75	171.79	63.61
Linpack MT	Score (in MFLOPS)	154.174	198.431	211.627	49.419
Geekbench SC	Score	472	488	455	285
Geekbench MC	Score	1407	1409	1375	741
Applaunch calculator	Launch time (s)	0.71	0.74	0.79	0.89
Applaunch Browser	Launch time (s)	1.007	1.02	1.07	1.46
BBench	Runtime(s)	190.9	184.17	187.2	246.24
Adobereader	Runtime(s)	77.14	79.14	79.95	103.61
facebook	Runtime(s)	138.02	141.08	141.79	144.6
ebizzy	Total records/sec	2017	2011	1757	472
Nenamark	Frames per second	35.6	35.2	34.9	37.4
Memcpy	Bandwidth (in MB/s)	3114	3060	2970	588

Table 5: Benchmark/Application performance for different benchmarks

For compute intensive applications, the performance governor gives the best performance with interactive closing in certain benchmarks. The interactive governor works better in multicore scenarios as the power budget limits the performance of the performance governor. Interactive applications like Facebook and Adobereader performs neck-to-neck in performance and interactive governors. Nenamark works best in powersave as the GPU governor and the DDR frequency is tweaked to provide maximum performance. It shows that Nenamark or some GPU intensive job may not depend on the CPU frequency at all. Though the above table is only the performance view, the power view is radically different. Even though the performance governor does well in maximum scenarios the power consumed by performance vs the interactive governor is very different. Another point worth noting is the quality-of-service for each case. For example, the series of operations done in Adobereader and Facebook which resulted in the scanty difference in runtime may be negligibly visible to the user. So, what matters is the amount of energy consumed and the peak power. If any application crosses the peak power, throttling will kick in and slow down the performance. The power budget is exceeded in the multicore variant of the Geekbench benchmark by the performance governor which caused it to perform poorer than the interactive governor.

Chapter 5: Benchmark and Workload Categorization

This section enlists a view into the different governor action based on the type of the workload. Looking at the performance table above doesn't give a clear view of the choice of governor in an energy constrained system. One needs to look at the energy efficiency. For the sake of clarity, multiple sub-sections are created to categorize various type of workloads:

5.1 CPU Intensive Single Core Workloads:

These are the workloads that are compute intensive and works best when the processors are at peak frequency. Race-to-idle scheme gives better performance and is often energy efficient as well. The pace-to-idle governors suffer from too many unnecessary frequency switches. Interactive governor works good if the workload is continuously CPU demanding. Let us consider certain compute intensive tasks.

5.1.1 Dhrystone:

Dhrystone is an example of continuously CPU demanding benchmark. Figure 5 shows how the current consumed while running Dhrystone is similar for interactive and performance which runs the CPU at 1.2GHz while it is much less in powersave owing to clipping the frequency at 800MHz. From the performance standpoint, dhrystone performs almost equally good for performance, interactive and ondemand governor with performance governor performing slightly better as it is in the maximum frequency from the beginning. The QoS of this benchmark is to achieve as high performance as possible. Profile power of all the benchmarks are plotted in Figure 5.

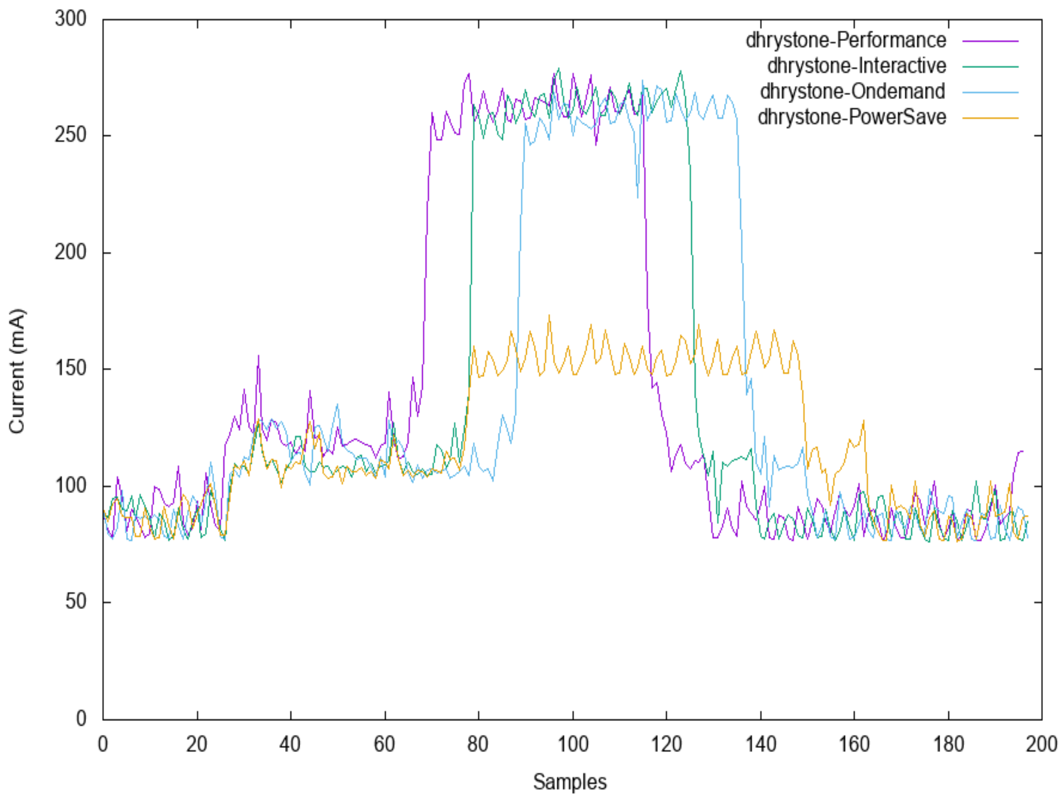

Figure 5: Current consumed by Dhrystone benchmark running with different governors

The energy consumed by the performance, interactive and the ondemand governors are the same as all of them operate constantly at the highest frequency with powersave governor consuming 86% of the energy. But the performance delivered is unacceptable. Race-to-idle strategy works best for dhrystone.

5.1.2 Applaunch:

Application launch is another case where the race-to-idle strategy works best. The QoS of this scenario is the fastest turn-around time after user click. Application launch depends on the size of the application. Three cases are considered: launch a light calculator application, a medium-heavy application of firefox browser and a very heavy application of google maps. In all the cases, the turn-around time is the best for the performance governor. This is one of the most important usecase in a smartphone application. Even if the overall energy consumption is high, one should ensure the fastest application launch as the launch time is very less.

First, the performance to the powersave governor energy consumption is compared for application launch. The power profile of launching the calculator is shown in Figure 6 and firefox application is shown in Figure 7. In the power profile, each of the spikes resemble an application launch. It is observed that the powersave governor takes considerable time and 13% more energy during applaunch of calculator while taking 27% more energy during the firefox launch. It provides crucial insights over the fact that powersave is not a good option when battery is low for launching application as it might end up consuming more energy from the battery.

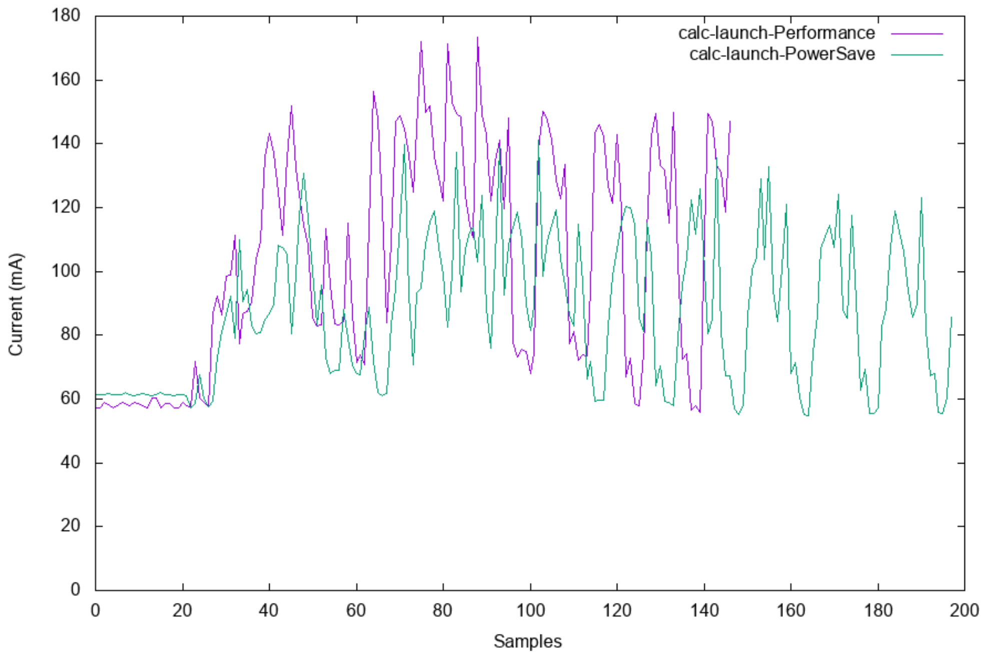

Figure 6: Current profile comparison between performance and powersave governors during calculator launch

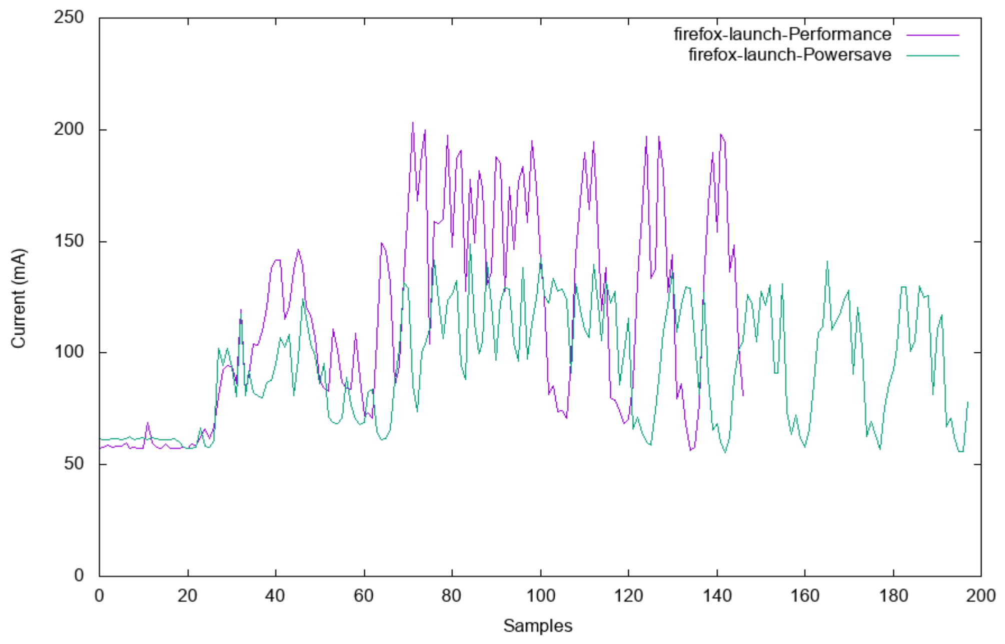

Figure 7: Current profile comparison between performance and powersave governors during firefox launch

Next, the performance and interactive governors are compared. Interactive governor consumes 4% less energy but provides delay of 4% in calculator and 1% in firefox. The main reason for the delay is the switch time to the maximum frequency. It is visible in the power profile as shown in figure 8. In the circled portion, it is seen the performance governor immediately works in the maximum frequency but the interactive governor lags to reach the maximum frequency with a lesser power slope. The application launch is performed three times to average out the result and the cache is warmed up in the initial portion of the figure which is not used to measure the launch latency.

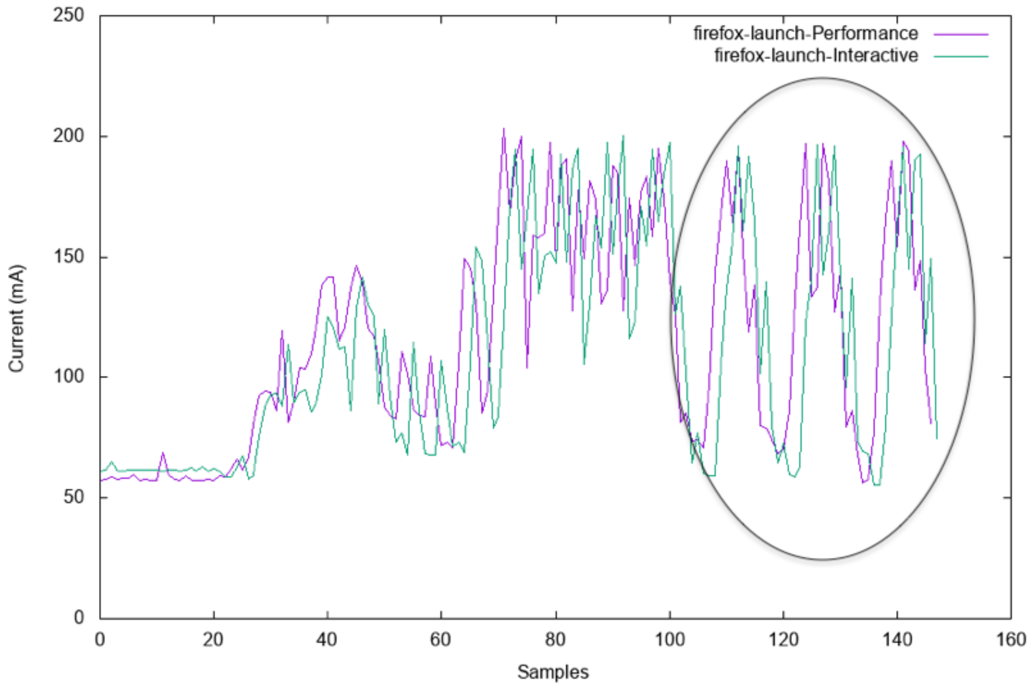

Figure 8: Current profile comparison between performance and interactive governors during firefox launch

Further analysis is performed on Antutu benchmark which is compute intensive and has a lot of tests using the integer and the floating-point units to the limit. Also, this benchmark provides more stress to single core performance. The Performance governor inherently works well for this benchmark especially in the phases where the integer and floating point operations are being performed. The current profile of the antutu benchmark as shown in Figure 9 illustrates that the pace-to-idle strategies like interactive and ondemand governors converge to performance mode during compute intensive tasks but after a delay and sometimes consuming more current due to DVFS switching. The powersave governor consumes much less current and gives unacceptable performance.

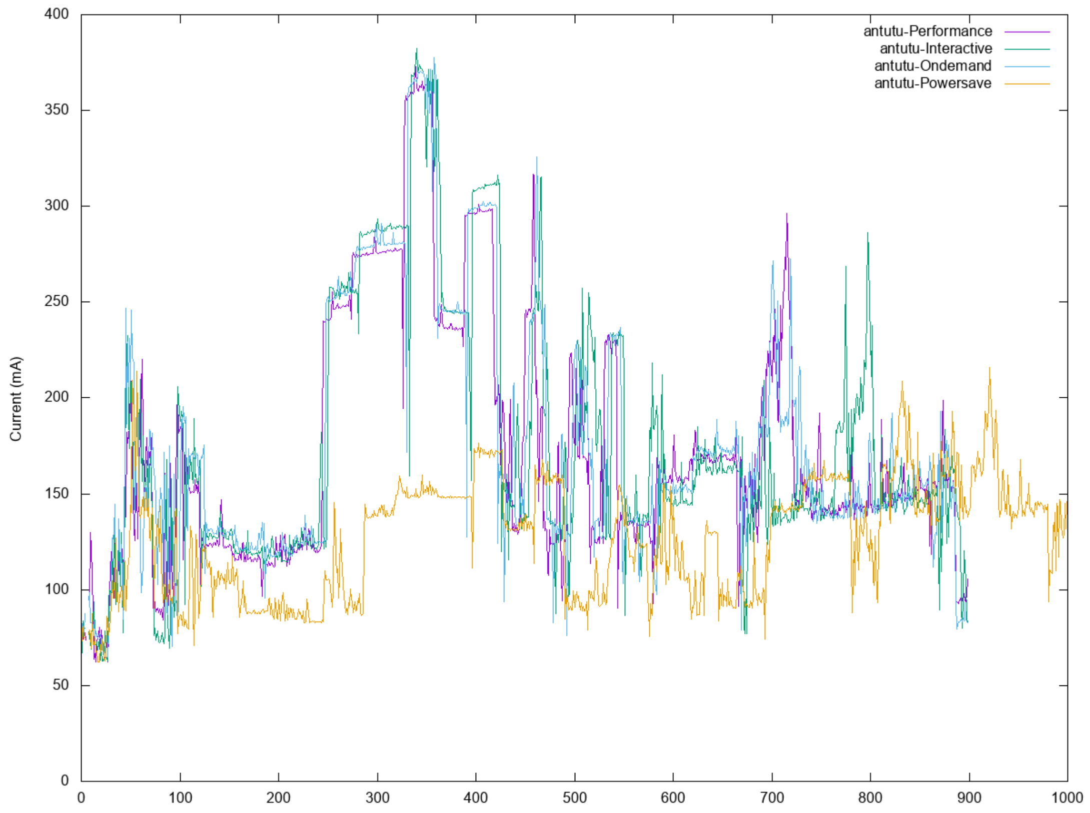

Figure 9: Current profile comparison between different governors during antutu benchmark run

Same observation is noticed in single thread linpack and the integer and floating point testcases of geekbench 3 which exemplifies the fact that in certain phases, the governor needs to be less conservative and should switch to the maximum frequency while in other cases, it should be more conservative. The application behavior, its state, priority in run queue and utime/stime can provide crucial application specific information that can help decide this heuristics on top of looking at the load average.

Figure 10 shows the energy comparison of the different benchmarks/applications which are compute intensive and favor the race-to-idle type of strategy. The scheduler utime value of these applications are high. These suggest that the workload characterization is important to understand and the frequency of the compute intensive phase of the application can be boosted up to the maximum frequency not only for performance but also for energy efficiency.

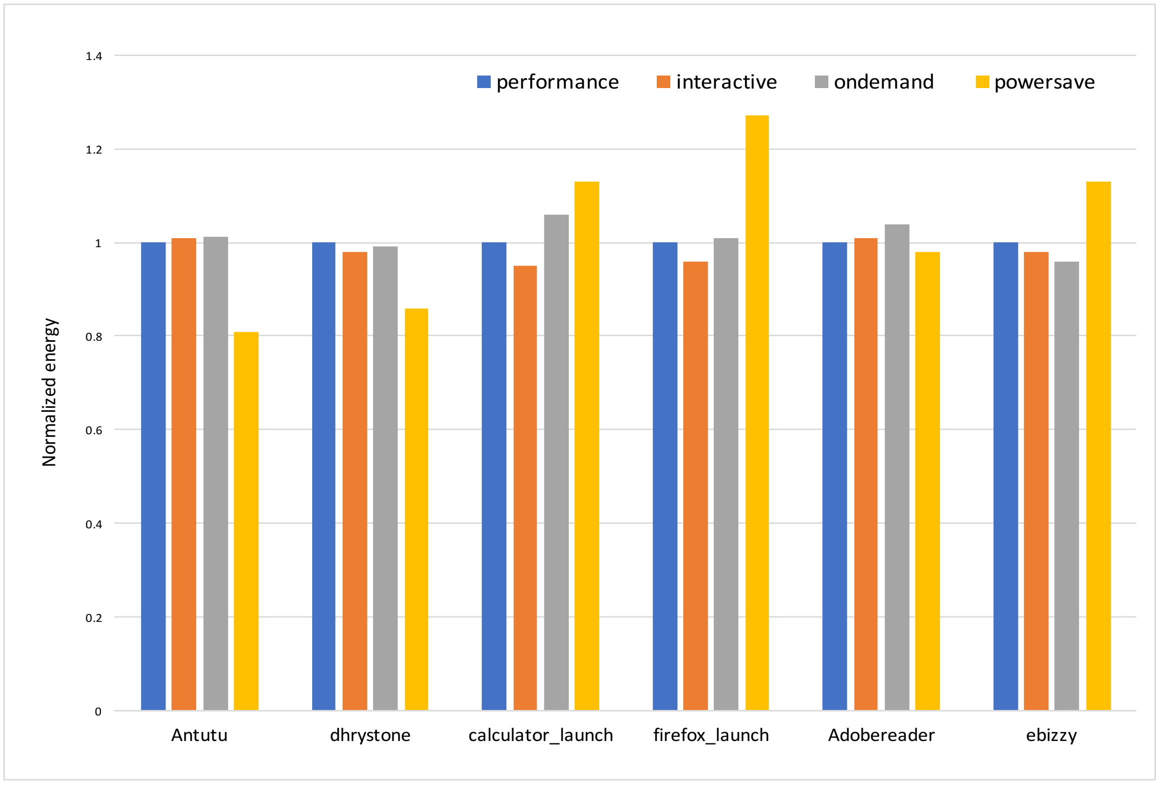

Figure 10: Normalized energy (wrt performance governor) Comparison of different governors for different benchmarks/applications

5.2 Multi-core Workloads:

A group of multicore workloads are evaluated which includes antutu multitask and runtime tests as a part of the user experience tests, the linpack multi-thread and the geekbench 3 multicore tests. In all these scenarios, it is observed that the interactive governor is the best in terms of both performance and power efficiency. It is surprising as to how the performance governor performs poorer than interactive in terms of performance due to the constrained power budget of a smartphone. The interactive tunes the frequency of each core separately based on the load average which also puts less pressure in the memory subsystem. On the other hand, the performance governor which runs all the CPU cores at the highest frequency ends up wasting more energy and the performance in throttled as the stressed resource doesn't get the lion's share of the power budget. The antutu multitask score of performance governor is 3370 while the geekbench score is 3397. Similar trend is seen in the linpack multi-thread score as shown above. The energy efficiency of these vectors is also high for interactive governor as it ramps the cpu frequency of each core differently based on the load average. Similar observation is seen in Figure 11 and 12 which compares the integer and the floating-point tests for Geekbench 3.

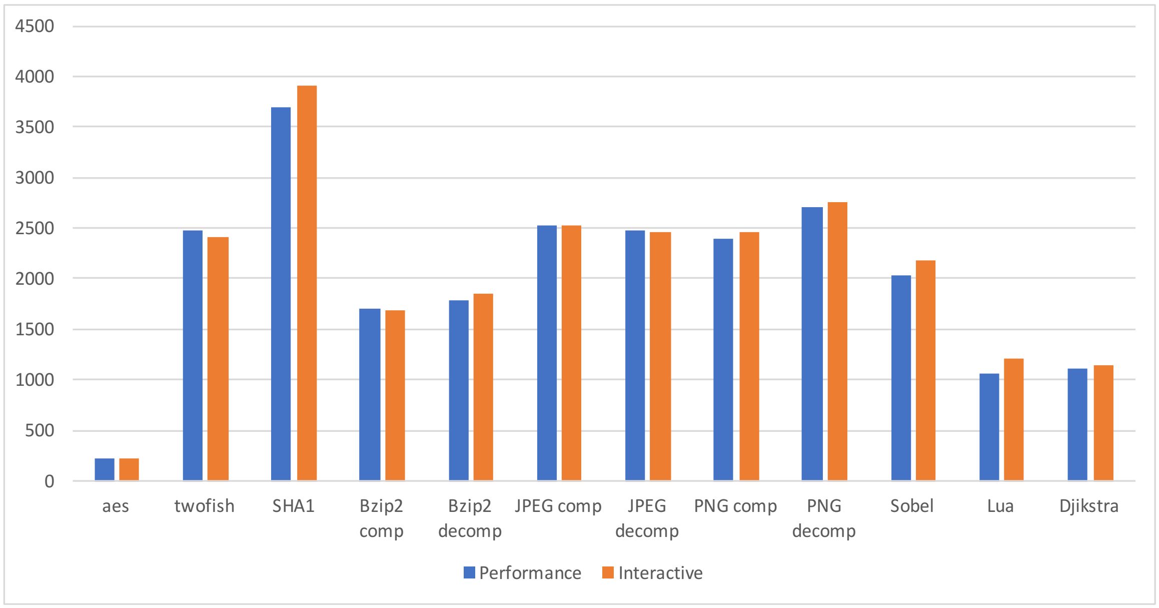

Figure 11: Multicore Geekbench 3 Integer benchmark comparison between performance and interactive governors

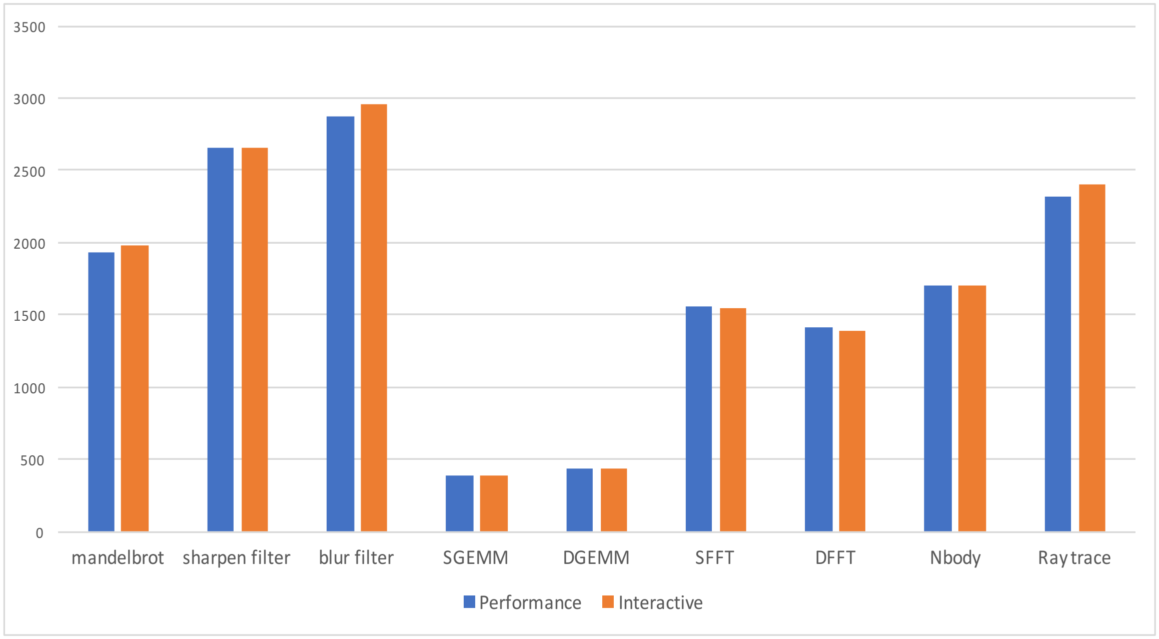

Figure 12: Multicore Geekbench 3 Floating Point benchmark comparison between performance and interactive governors

Most of the benchmarks where interactive works better are due to the power budget constraints. Note that the single core results show the opposite trends for integer and floating point performance. Thus, a pace-to-idle strategy is necessary for performance and energy efficiency for a multi-core application.

5.3 INTERMITTENT CPU WORKLOADS WITH IO OPERATIONS:

Some interactive applications are CPU semi-intensive while having IO operations like user input or loading non-cacheable memory or using system features like the gps. One such application is BBench which loads saved webpages and scrolls through them. Scrolling through webpages need not require the CPU to operate at the maximum frequency but some intermediate frequency while it can be sent to idle when waiting for user input. Interactive governor works best in these kinds of cases. Similar is the scrolling through the Adobereader and facebook applications. In Adobereader CPU is only needed on searching for a text. Below table shows the time in DVFS states for interactive governor. Since the CPU is prevented to reach the idle state, it will stay in 800MHz when it is waiting for user input.

DVFS mode (in MHz)	**BBench**	**Facebook**	**Adobereader**
1209	9085	5788	3625
1152	318	186	153
1094	233	165	125
994	2119	1360	800
800	6893	6714	3286

Table 6: Frequency residency of interactive governor for user applications

The interactive governor consumes 17% less energy for BBench while saving 2% for facebook and 13% for Adobereader. Thus, pace-to-idle strategy saves energy while delivering similar performance. The power saving will be even more if the CPU is put to low power state. But performance is more for the facebook and Adobereader as the applications need a sudden surge of CPU after each IO operation where the interactive governor takes time to ramp on. Figure 11 shows the current profile for BBench showing the uniformity of IO operation (browser scrolling). The performance governor has higher power as the green lines are dominated by the purple ones.

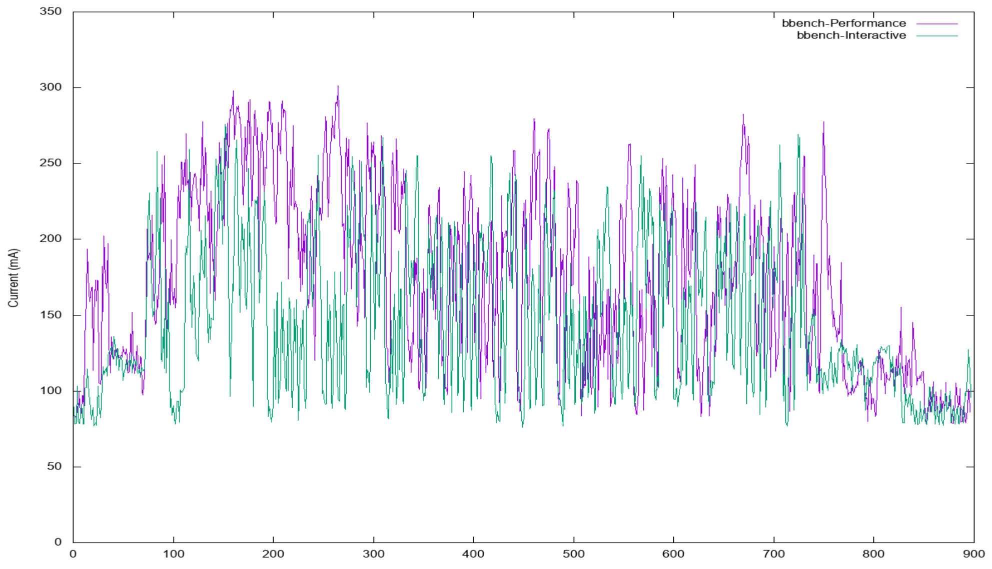

Figure 13: Current profile of BBench with performance and interactive governors

5.4 MEMORY INTENSIVE WORKLOADS:

In some of the memory intensive workloads, it is necessary to keep the bus frequency at the peak. Moreover, the CPU DVFS point should not change the performance but yields power savings if the CPU is kept in the idle state. The Memcpy vector is run and it is found that the interactive and the performance mode gave similar performance. The powersave governor gave poor results because the bus frequency was scaled based on the CPU frequency and the bus frequency while running the test is much smaller. Later, the bus frequency is tweaked to maximum for powersave and found comparable performance of bandwidth around 3GB/s.

5.5 NON-CPU WORKLOADS:

There are other applications like playing a video which requires the multimedia unit to be active. The CPU can stay in the powersave mode while providing power budget to the DDR and the multimedia unit to perform. Moreover, playing games require the GPU to be in higher performance mode to render better user experience. The CPU should not be kept in the high-performance mode unless advertisements and other demands. This will leave the device cooler and less prone to thermal throttling as video, audio and gaming run for much longer than benchmarks. So, energy efficiency is important not only for the thermal throttling but also for the battery life.

Nenamark is run on all the governors by keeping the GPU in adreno-msm-tz governor and found similar performance. However, while running the nenamark benchmark with powersave, the bus frequency is increased to 800MHz for fairness and changed the GPU governor to performance receiving better frame rate of 37.4fps as against 35fps for others. It proves that a system wide governor would help increase performance of the overall system. The current profile of all the governors with same bus frequency is shown below in Figure 12.

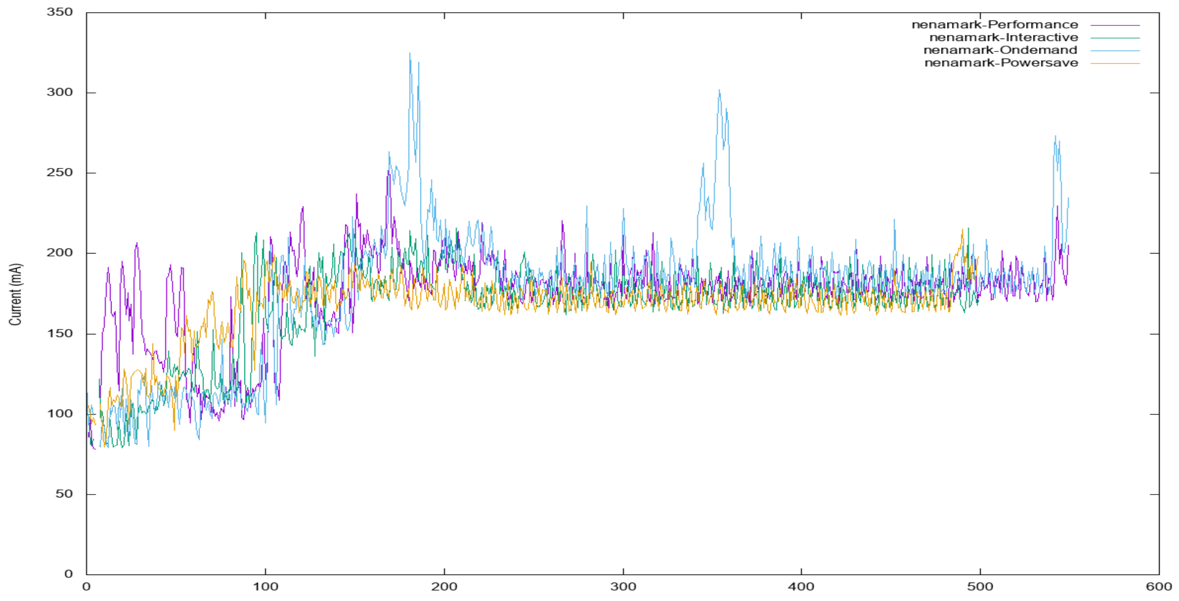

Figure 14: Current profile of Nenamark with different governors.

Similar trend is seen with video and audio as shown in the following table:

Application	**Performance**	**Interactive**	**ondemand**	**powersave**
Audio_30s_playback	1	0.915	0.842	0.841
Video_720p_20s	1	0.914	0.956	0.877

Table 7: Normalized Energy consumed in audio/video playback

The video power is optimistic as there is no display connected to our setup. Addition of display power will decrease the potential of saving. But at the same time, it can introduce a lesser power budget in which case, the power-hungry performance mode may start showing jitters in performance as is normally seen in the garbage collection window.

Chapter 6: Observations

After running various types of workloads on all the different kinds of governors, it is seen that choosing the correct governor in a battery-operated system-on-a-chip depends vastly on the workload and on the power consumption of the other resources. Governors should also minimize triggering of thermal throttling hardware to attain best efficiency and performance in addition to better chip life.

As mentioned in the introduction that a race-to-idle scheme works well for servers because more importance is given on the performance, this scheme is thought to be a poor fit for battery-operated devices not only in terms of energy consumption but is sometimes detrimental to the performance of the device. Nevertheless, the race-to-idle scheme performs better not only in performance but also it gives better energy efficiency in compute intensive single core applications and sometimes in multi-core applications as well. Moreover, there is provision of switching off each core into several idle low power states in a multi-core device. So, if a governor finishes the pending work in the minimum time and goes to idle, it might be energy efficient. This strategy will work even better with technology shrinking as the leakage power becomes comparable with the dynamic power and is especially useful in big cores. With more application-specific units are put in the SoC, having a global governor controlling the DVFS modes of every component based on workload characterization will be the desired solution. The race-to-idle scheme also makes sharing of power source easier as the units are active for the minimum amount of time. Last but not the least is the fact that race-to-idle schemes give better performance most of the time.

The pace-to-idle strategy also performs well in multiple scenarios where multiple resources are used together or in a sequential manner. For instance, the BBench workload loads a set of heavy webpages from the memory making it an IO intensive workload followed by the execution of contents in the webpages, which is compute-intensive. In these scenarios, the pace-to-idle strategies work best as all the units like the memory-bus and the CPUs are appropriately scaled whenever it is needed. The CPU frequency is lowered by the interactive governor when it is temporarily idle redirecting the power budget to other units like the memory bus which in turn can run at a higher frequency increasing the memory bandwidth. It also helps in thermal distribution as the cores get heated up when it is constantly at higher frequency reducing reliability and performance by engaging the thermal throttler. It is intuitive to lower the core frequency when only a portion of compute power is required. In interactive applications like facebook, pdf viewing and messaging, which constitutes a major portion of smartphone usage, pace-to-idle is a good solution.

If the frequencies are chosen appropriately, the pace-to-idle policy is a generalization of the race-to-idle policy. One of the major concerns of determining a good operating point is the input which is used to determine the DVFS operating point. The interactive governor uses CPU load average in determining the next frequency. Also, to filter out transient spikes in load averages, it first reaches an intermediate frequency and then boosts further based on re-observation of the load average. Applications like app_launch which need immediate boost suffers from this delay which introduces lag in performance. Moreover, the interactive governor is ignorant of the other CPUs and resources working in the system. Hence, it leads to choose an optimistic DVFS mode which could not be supported by the power source.

Chapter 7: The Winteractive Governor

Keeping the above observations in mind, the interactive governor is enhanced to shed its low response time and increase its visibility to the system level. The system level view is crucial in power constrained devices and should perform better in multi-core scenarios as is identifying the task state and other process traits from the scheduler to conceptually develop a workload aware governor named Winteractive governor.

The winteractive governor takes input from the process task_struct to take better governor decisions at CPU level. Moreover, it has a system level credit system that takes the final decision based on the power budget of the system. Monitoring the task_struct of all processes is expensive. Hence, the winteractive governor only looks at the user processes which are much fewer compared to the overall processes and services that run in an android system. The basic block diagram is shown in figure 15:

The winteractive governor is built on top of the interactive governor changing the target load parameter to quickly boost to the maximum frequency. It also can tune the hysteresis timer to quickly return to idle. A system-wide credit system exploits the intermediate frequency during multi-core applications or tasks that need both CPU and GPU. It understands process state and type to determine the frequency. When it cannot decide, it uses the load average to determine the frequency much like the interactive governor. It is a more sophisticated pace-to-idle mechanism that captures race-to-idle strategy faster giving more performance and saving energy in certain cases. The GPU is monitored by the load average based mechanism to know the GPU utilization. The governor maintains a app_list consisting of launched applications. The governor periodically scavenges the scheduler to ensure if the process has become dead or zombie from the task->state. A dead process is one which is killed either by the user or by process manager. A zombie process is one which is dead and is parentless. A process is added to the app_list when a zygote process is triggered which is sensed from the task->state of the zygote or the zygote64 process.

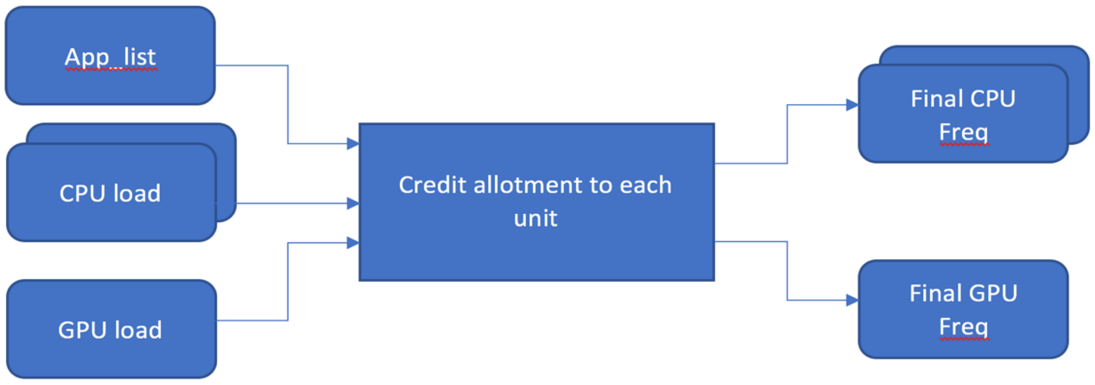

Figure 15: Block diagram of a system level multi-resource governor management

The first step performed in the GOVERNOR_INIT stage is to program the winteractive tunables same as the interactive governor. A search is performed on the entire task list to figure out the PID of the Zygote/Zygote64 process. The normal action at every interval is illustrated in Illustration 1:

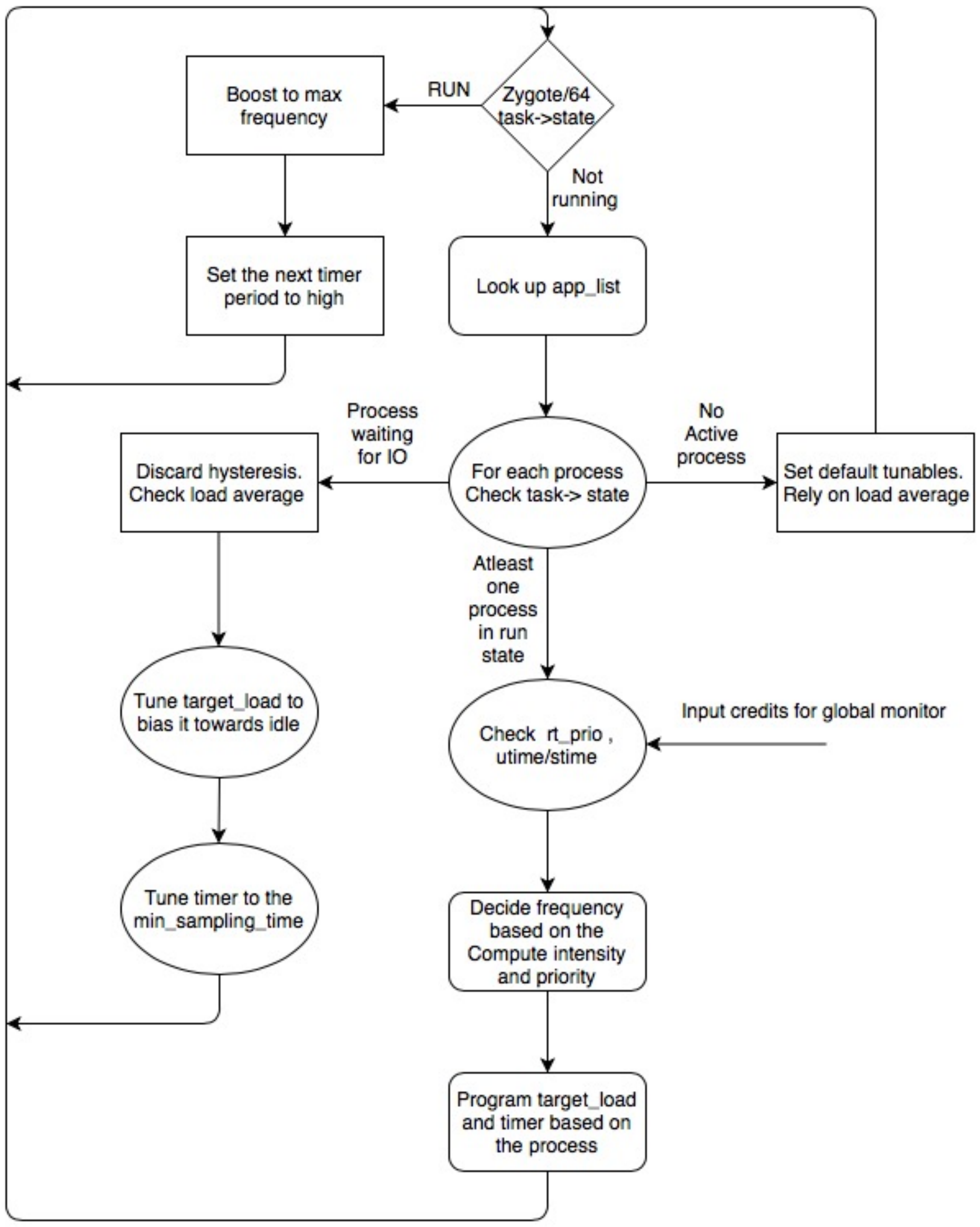
Zygote/64 task->state
RUN
Boost to max frequency
Set the next timer period to high
Not running
Look up app_list
For each process Check task-> state
Process waiting for IO
Discard hysteresis. Check load average
Tune target_load to bias it towards idle
Tune timer to the min_sampling_time
No Active process
Set default tunables. Rely on load average
Atleast one process in run state
Check rt_prio , utime/stime
Input credits for global monitor
Decide frequency based on the Compute intensity and priority
Program target_load and timer based on the process

Illustration 1: Action flowchart of Winteractive governor

Now let us explore on how the winteractive governor helps get better DVFS point for different scenarios:

7.1 COMPUTE INTENSIVE SINGLE CORE APPLICATIONS:

Whenever, a user application in the app_list is running (task->state = RUN), the utime of the corresponding process gives an idea of the compute intensity of the task. If the utime recorded from the past is high, the frequency is immediately boosted rather than looking at the load average. Moreover, if the utime is moderate and is comparable to stime, the normal interactive operation is performed in determining the DVFS mode. The rt_priority is another metric that gives crucial information here. If the process is present in the run queue but is waiting for IO or other services like the location services, the DVFS mode can be chosen to be the lowest frequency with minimum sampling time programmed to quickly re-adjust the frequency when the dependency is ready.

One of the major advantages of winteractive over interactive is the decreased latency of application launch. The Zygote or Zygote64 processes give early information of a process launch. Hence, if the zygote/64 is running, the winteractive simply boosts to the maximum frequency and sets the lowest sampling time. Setting the lowest sampling time relaxes the governor process allowing less kernel interference.

7.2 MULTI CORE APPLICATIONS:

The global power credit management system helps in determining a modest frequency not to run out of power budget. So, the CPUs don’t blindly go to the maximum frequency as they are aware of other resources running in the system. Moreover, based on the relative load average, preference can be given to specific cores. Moreover, the task->struct also provides information for CPU affinity which can be used to figure out the running cores. The credit based system will also return credits if an application is waiting in one processor, the frequency goes to idle immediately as against the hysteresis wait in interactive. If the CPU affinity of the process is high and credits are exhausted, the max_freq_hysteresis is set to near 0.

7.3 INTERMITTENT CPU WITH IO OPERATIONS:

The DVFS point can be immediately set to idle if the task state is not running. Moreover, in this kind of applications, the stime is very high. Typically, such applications are chatting or browser applications requiring user input. The task state is interruptible during which the processor can be sent to idle extracting more power savings than interactive. Moreover, as the utime of the process is only calculated when it is running, it helps determine the compute intensity of the process during running. Thus, when the process state is run, the utime can help trigger the appropriate frequency reducing lags to compute the load average and ramping in steps. Thus, short bursts of high performance and idle are expected in these applications ensuring no lag and maximum energy efficiency.

7.4 MEMORY INTENSIVE AND NON-CPU OPERATIONS:

The task struct provides information about the working set which could be potentially used to enhance the performance of memory intensive applications but is not currently performed in winteractive governor. For non-cpu applications, the CPUs rely on the remaining credit available and the load average. Most of the credit is given to the GPU or other active resource.

So, the winteractive governor retunes the interactive governor dynamically and get to maximum frequency faster when it is required and also extracts idle opportunity when the application is waiting for IO or other services.

Chapter 8: Conclusion and Future work

The different governor strategies are studied and their impact on performance and energy efficiency on different workloads. Analysis is performed on the existing race-to-idle and pace-to-idle strategies and figured out that race-to-idle strategies are good in compute intensive applications and when sudden surge of CPU activity is required. Overall, the pace-to-idle strategies work better in interactive applications and multi-threaded workloads due to power budget limitations. Moreover, on a broader perspective, a system wide governor can manage task distribution based on the activity of all the resource in the system sharing a common power source. Unique characteristics are observed which includes that the performance governor doesn't always provide the best performance and the powersave governor the best energy efficiency. The performance and energy efficiency depend on the nature of workload. Hence, workload characterization is crucial in deciding the frequency when the task is running.

Existing interactive governor depend on load average of a CPU and ramps to the maximum frequency in steps to provide an energy efficient performance. But in applications which are highly compute intensive or need a sudden surge of resource activity, the interactive governor takes time to ramp up to the maximum frequency wasting opportunity for achieving best performance. Moreover, once it reaches the maximum frequency and the load average comes down, it waits blindly in the maximum frequency for a hysteresis period before returning to lower frequency. The ondemand governor aggressively falls to lower frequency causing too many DVFS switches. But understanding the workload provides more information if the reduction in load average is temporary or permanent. For example, if the ongoing process is waiting for user input, the latency is huge and the processor can immediately ramp down frequency. While if it is performing some services for which a different kernel process is running, the wait is temporary and the processor can afford to stay in relatively higher frequency. The task state provides this crucial information of the state of the process. Moreover, the nature of the application can help to decide if the frequency should be ramped up in steps or directly to the highest frequency upon return from IO wait. The application launch is treated as a special case as it is short lived and to achieve the best performance, the race-to-idle strategy should be used. So, Zygote process run state serves as a signal of app launch and boosting the frequency immediately for jitter free experience.

Further work will need tuning the winteractive governor to take care of the memory intensive processes. The task_struct provides information of the pages touched and the working set size, the number of reads/writes etc. It can be used to further tune the frequency of the bus and aggressive idling of the CPU during memory operations. The thermal throttling interrupt can be taken by the credit system to lower the credits. But on top of it all, the governor should be simple as it is a kernel task itself and should not intrude too much in the system so that it consumes considerable performance and energy. Choosing the useful characteristics from the task_struct is crucial. Moreover, choosing the important processes are equeally important. There are hundreds of processes running in the android system at any given point of time. A lot of them are services and supporting library processes. The winteractive governor only chooses to monitor those processes whose parent is zygote or zygote64 stating that these processes are user applications. A further analysis needs to be performed to check the contribution of kernel processes. When none of these processes are in run state, the winteractive governor depend on the load average much like the interactive governor. Overall, characterization of a workload and wise current distribution to the critical components is imperative in designing a governor that not only gives the desirable performance but also is highly energy and thermally efficient.

Bibliography

[1] Weicker,R.P. (1984). Dhrystone, a synthetic systems programming benchmark. Communications of the ACM,27(10),1013-1030.

[2] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in Proceedings of the Linux Symposium, vol. 2, pp. 215–230, 2006.

[3] Gutierrez, Anthony, Ronald G. Dreslinski, Thomas F. Wenisch, Trevor Mudge, Ali Saidi, Chris Emmons, and Nigel Paver. "Full-system analysis and characterization of interactive smartphone applications." In Workload Characterization (IISWC), 2011 IEEE International Symposium on, pp. 81-90. IEEE, 2011.

[4] Russell, Rusty. "Hackbench: A new multiqueue scheduler benchmark." Message to Linux Kernel Mailinglist: http://www.lkml.org/archive/2001/12/11/19/index.html (2001).

[5] Dongarra, Jack J., Cleve B. Moler, James R. Bunch, and Gilbert W. Stewart. LINPACK users' guide. Society for Industrial and Applied Mathematics, 1979.

[6] Hoffmann, Henry. "Racing and pacing to idle: an evaluation of heuristics for energy-aware resource allocation." In Proceedings of the Workshop on Power-Aware Computing and Systems, p. 13. ACM, 2013.

[7] Albers, Susanne, and Antonios Antoniadis. "Race to idle: new algorithms for speed scaling with a sleep state." ACM Transactions on Algorithms (TALG)10, no. 2 (2014): 9.

[8] Albers, Susanne, and Antonios Antoniadis. "Race to idle: new algorithms for speed scaling with a sleep state." ACM Transactions on Algorithms (TALG)10, no. 2 (2014): 9.

[9] Rao, Karthik, Jun Wang, Sudhakar Yalamanchili, Yorai Wardi, and Ye Handong. "Application-Specific Performance-Aware Energy Optimization on Android Mobile Devices." In High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on, pp. 169-180. IEEE, 2017.

[10] Isci, Canturk, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget." In Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture, pp. 347-358. IEEE Computer Society, 2006.

[11] Huang, S., and W. Feng. "Energy-efficient cluster computing via accurate workload characterization." In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 68-75. IEEE Computer Society, 2009.

[12] Begum, Rizwana, David Werner, Mark Hempstead, Guru Prasad, and Geoffrey Challen. "Energy-performance trade-offs on energy-constrained devices with multi-component dvfs." In Workload Characterization (IISWC), 2015 IEEE International Symposium on, pp. 34-43. IEEE, 2015.

[13] Eyerman, Stijn, and Lieven Eeckhout. "Fine-grained DVFS using on-chip regulators." ACM Transactions on Architecture and Code Optimization (TACO) 8, no. 1 (2011): 1.

[14] Kim, Wonyoung, Meeta S. Gupta, Gu-Yeon Wei, and David Brooks. "System level analysis of fast, per-core DVFS using on-chip switching regulators." In High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on, pp. 123-134. IEEE, 2008.

[15] “Dragonboard 410c.” https://developer.qualcomm.com/hardware/dragonboard-410c

[16] “Measuring power consumption for Dragonboard 410c.” https://developer.qualcomm.com/download/db410c/power-measurement-appnote.pdf

[17] “Linaro workload automation”. https://media.readthedocs.org/pdf/workload-automation/latest/workload-automation.pdf

[18] “Android debug bridge”. https://developer.android.com/studio/command-line/adb.html.

[19] C. Isci, G. Contreras, and M. Martonosi, “Live, runtime phase monitoring and prediction on real systems with application to dynamic power management,” in Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchi- tecture, pp. 359–370, 2006.

[20] Su, Bo, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathouse, and Zhiying Wang. "PPEP: Online performance, power, and energy prediction framework and DVFS space exploration." In Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on, pp. 445-457. IEEE, 2014.

[21] Paul, Indrani, Srilatha Manne, Manish Arora, W. Lloyd Bircher, and Sudhakar Yalamanchili. "Cooperative boosting: needy versus greedy power management."

In ACM SIGARCH Computer Architecture News, vol. 41, no. 3, pp. 285-296. ACM, 2013.

[22] APQ8016E Clock Plan. https://developer.qualcomm.com/download/db410c/clock-plan-apq8016e.pdf

[23] A.Alimonda, S.Carta, A.Acquaviva, A.Pisano and L.Benini, “A feedback-based approach to dvfs in data-flow applications,” the IEEE Transactions on computer Aided Design of Integrated Circuits and Systems, vol 28. no. 11, pp 1691-1704, 2009.

[24] Y Zhu, VJ Reddi, “High-performance and energy-efficient mobile web browsing on big/little systems” HPCA 2013.