MULTIMEDIA-SPECIFIC CPU GOVERNOR FOR ANDROID DEVICES

By

NATHAN JOSEPH PAULSON

A Thesis Submitted to The Honors College

In Partial Fulfillment of the Bachelors degree

With Honors in

Computer Science

THE UNIVERSITY OF ARIZONA

M A Y 2 0 1 3

Approved By:

Dr. Chris Gniady
Department of Computer Science

The University of Arizona Electronic Theses and Dissertations
Reproduction and Distribution Rights Form

The UA Campus Repository supports the dissemination and preservation of scholarship produced by University of Arizona faculty, researchers, and students. The University Library, in collaboration with the Honors College, has established a collection in the UA Campus Repository to share, archive, and preserve undergraduate Honors theses.

Theses that are submitted to the UA Campus Repository are available for public view. Submission of your thesis to the Repository provides an opportunity for you to showcase your work to graduate schools and future employers. It also allows for your work to be accessed by others in your discipline, enabling you to contribute to the knowledge base in your field. Your signature on this consent form will determine whether your thesis is included in the repository.

Name (Last, First, Middle)
Paulson, Nathan, Joseph

Degree title (eg BA, BS, BSE, BSB, BFA):
BS

Honors area (eg Molecular and Cellular Biology, English, Studio Art):
Computer Science

Date thesis submitted to Honors College:
05-02-2013

Title of Honors thesis:
MULTIMEDIA-SPECIFIC CPU GOVERNOR FOR ANDROID DEVICES

The University of Arizona Library Release Agreement

I hereby grant to the University of Arizona Library the nonexclusive worldwide right to reproduce and distribute my dissertation or thesis and abstract (herein, the "licensed materials"), in whole or in part, in any and all media of distribution and in any format in existence now or developed in the future. I represent and warrant to the University of Arizona that the licensed materials are my original work, that I am the sole owner of all rights in and to the licensed materials, and that none of the licensed materials infringe or violate the rights of others. I further represent that I have obtained all necessary rights to permit the University of Arizona Library to reproduce and distribute any nonpublic third party software necessary to access, display, run or print my dissertation or thesis. I acknowledge that University of Arizona Library may elect not to distribute my dissertation or thesis in digital format if, in its reasonable judgment, it believes all such rights have not been secured.

☒ Yes, make my thesis available in the UA Campus Repository!

Student signature: Nat Paulson Date: 05-01-2013

Thesis advisor signature: [signature] Date: 05.02.2013

☐ No, do not release my thesis to the UA Campus Repository.

Student signature: _____ Date: _____

Last updated: 04/01/13

Abstract

With the massive increase in smartphone usage in recent years has come a new focus on multimedia applications. Because of this increase and the large amount of energy used by these applications, there has become a need for multimedia-specific DVFS for mobile devices. This paper examines the potential for energy savings in multimedia applications on Android devices. It also discusses the viability of a CPU governor to provide these savings through buffer monitoring, including different methods used in an attempt to implement this.

Introduction

The prevalence of smartphones in daily activity has increased greatly since their inception into modern culture in recent years. Due to increases in technology and the sheer number of smartphones, there has been an increased focus on applications that are not communication-oriented. For example, the top applications in the Google Play Store upon writing this paper are almost exclusively social media, gaming and multimedia applications. The study by Böhmer et. al [1] reinforces this shift quantitatively through a large scale study of application usage on mobile devices. In this study, only 49.5% of application usage was related to communication. It becomes quite clear through this number that the focus of mobile devices has shifted greatly from their original intent.

Along with this realization is the fact that applications focused on gaming or multimedia utilize more fully the hardware on the mobile devices. Proportional to this increased utilization is the massive increase in energy usage brought about by multimedia applications on mobile devices. With this increase comes the need to compensate for the heightened energy usage. This is accentuated by the fact that many of these applications have practical use away from home

where battery life becomes essential. For instance, popular music applications are often used during work commutes or in vehicles.

Further, mobile television and mobile video are becoming increasingly popular on devices as well. In the study by Miyauchi [8], it was found that the vast majority of mobile video viewing was done during commutes and much less occurred at home. Mobile video is a type of device usage that requires a high amount of power. Because of this and the increasing popularity of applications dealing with mobile video and multimedia, it is necessary to develop solutions more specific to these high energy applications.

In an attempt to deal with this problem, a CPU governor designed specifically for multimedia applications and minimizing energy usage with them could be utilized. To tailor a CPU governor to multimedia applications, it would have to deal mainly with the playing of audio and video. There are a variety of ways to go about this. The first way it was attempted in this paper was to directly monitor the buffers on the system and adjust the CPU based on the amount of information needed to be processed by the hardware. The next approach was to instrument the Java libraries responsible for dealing with multimedia on Android devices and then view the size of the data buffers being passed.

Background

The Android platform is an open-source project owned by Google. It includes an operating system based on Linux and since its release has grown rapidly, now owning a large portion of the global market share of smartphones. The architecture of the Android system has a Linux kernel at the base which handles the drivers that operate the hardware. Above that are the libraries that are needed by the Android system for functionality, including the media libraries

responsible for dealing with multimedia on Android devices. The ease of development on the Android system due to it being open-source and having a familiar kernel as well as its prevalence in the smartphone market made it the perfect target system for this study.

An important concept in energy savings is dynamic voltage and frequency scaling (DVFS). In DVFS, a processor's voltage and frequency are scaled up or down to fixed intervals based on the current load of the processor and system. This can be changed often to reflect the current needs of the system. Since this allows the system to only run at high frequencies when necessary, it allows for energy savings since a device only needs to run at maximum frequency and voltage during demanding loads.

The mechanism by which DVFS is implemented is often through the use of a CPU governor. A CPU governor is what actually checks the load of the system and changes the frequency of the processor (CPU) to a determined level. On Android devices, power management and DVFS is largely completed by the Linux kernel. This means that the Linux governors are what are also used on the Android system. The default governor on Android devices is the OnDemand governor. This governor works by having a threshold for the CPU load. If this threshold is exceeded, the governor will immediately set the CPU frequency to the maximum possible. When the load decreases, the governor will step down the frequency intervals until it reaches the minimum possible frequency. This type of governor is used as the default because it guarantees high performance by always moving to the maximum frequency. It has poor energy savings for the same reason, though. Because most applications do not require the maximum frequency to run fluidly, defaulting to this wastes energy since a lower frequency could be used that would result in lower energy usage. Other types of governors that will be mentioned in this paper are the Performance governor and the Powersave governor. The

Performance governor is one that sets the CPU frequency to the maximum possible frequency interval and locks it there. This governor does not ever take into account the CPU load and does not ever change frequency. It provides the maximum performance possible since it is always at the maximum frequency. Because of this, it also provides the worst energy savings. The final governor mentioned here is the Powersave governor. This is the opposite of the Performance governor in that it locks the CPU at the lowest possible frequency. It provides poor performance but the best energy savings.

Related Work

Because of the importance of energy savings in mobile devices, much research has been done to reduce energy usage. One of the critical components in energy savings is the CPU. In the related work on conserving energy in the CPU, Dynamic Voltage and Frequency Scaling (DVFS) is commonly employed.

DVFS schemes use different metrics in order to determine how to scale the CPU. Choi et al. [1, 2] divided the CPU workload into on-chip and off-chip workloads. The on-chip workload is the number of clock cycles used to perform CPU instructions and the off-chip workload is the number of cycles used to perform external memory accesses. These workloads are then used to determine the optimal voltage and frequency.

Liang et al. [5] implements a user-level CPU frequency governor named AD-DVFS. The governor uses the PMU hardware counters to determine the memory access rate. This rate is run through a prediction equation by the governor to determine the critical speed and set the CPU frequency accordingly.

Keller et al. [3] implements a different type of CPU frequency governor based on performance events. The frequency in this governor is determined by the number of floating

point operations per second. This is determined by a performance monitor which the governor then uses to determine and set the correct frequency.

Schöne et al. [7] uses a CPU governor which adds performance counters to the processor cores in order to measure the number of executed instructions and the number of last level cache misses during a certain length of time. From this, the instructions executed per miss is calculated which determines how memory-bound the workload is. The frequency is then adjusted based on this number.

Data

Table I: Streaming music energy usage

	Google Play Music	iHeartRadio	Spotify
Max	0.145233	0.193647	0.147913
Default	0.08123	0.124543	0.092545
Min	0.078492	0.109622	0.081001
Max/min	1.850290475	1.766497601	1.82606388
Default/min	1.034882536	1.136113189	1.14251676

Table II: Local music energy usage

	Apollo	MX Player	Rocket Music Player
Max	0.130685	0.126997	0.136055
Default	0.056214	0.058567	0.056182
Min	0.040601	0.046551	0.045662
Max/min	3.21876308	2.72812614	2.97961106
Default/min	1.38454718	1.2581255	1.23038851

Table III: Streaming HTML5 energy usage

	Firefox	Google Chrome	Opera Mobile
Max	0.393435	0.4362	0.307821
Default	0.360695	0.416429	0.286483
Min	0.329331	0.395993	0.278937
Max/min	1.194649152	1.101534623	1.10355026
Default/min	1.095235493	1.051606973	1.0270527

Table IV: Streaming video energy usage

	Youtube	Youtube HD	Netflix	Dailymotion
Max	0.270138	0.297592	0.303094	0.335087
Default	0.225364	0.239051	0.254382	0.271314
Min	0.200949	0.231247	0.248976	0.266665
Max/min	1.34431124	1.28690102	1.21736232	1.2565841
Default/min	1.12149849	1.03374746	1.02171294	1.01743386

Table V: Local video (1024) energy usage

	MX Player	VPlayer	RockPlayer 2
Max	0.274046	0.349434	0.363705
Default	0.266654	0.319174	0.339996
Min	0.24364	0.302377	0.309736
Max/min	1.124798884	1.155623609	1.17424194

Default/min	1.094459038	1.05554986	1.0976961

Table VI: Local video (1280) energy usage

	MX Player	VPlayer	RockPlayer 2
Max	0.332453	0.432225	0.420623
Default	0.307486	0.409085	0.407806
Min	0.256853	0.362335	0.339305
Max/min	1.29433178	1.1928878	1.23966048
Default/min	1.19712832	1.12902425	1.20188621

Table VII: Local video (1920) energy usage

	MX Player	VPlayer	RockPlayer 2
Max	0.412746	0.529411	0.427015
Default	0.388226	0.502795	0.407803
Min	0.219353	0.366118	0.360083
Max/min	1.88165195	1.44601194	1.18587937
Default/min	1.76986866	1.37331407	1.132525

Table VIII: List of media libraries by application

	Android/media/AudioTrack	Android/media/MediaPlayer
Apollo		X
Chrome		X
Dailymotion		X
Google Play Music		X
Firefox		
iHeartRadio	X	X
MX Player		X

Netflix	X	X
Opera Mobile		X
Rocket Player		X
RockPlayer 2		X
Spotify	X	X
VPlayer	X	
Youtube		X

Discussion

Tables I through VII show the energy usage of each of the different programs studied. These programs consist of music and video players some of which play local audio and video files and some of which stream the audio or video being played. In particular, the audio applications measured while using local audio files were the Apollo music player that comes with CyanogenMod, MX Player and Rocket Player. The audio applications measured while streaming audio files were Google Play Music, iHeartRadio and Spotify. The browsers that were used were measured while streaming HTML5 and included Google's Chrome browser, Mozilla Firefox and Opera Mobile. Three different applications were measured while streaming video: Youtube, Netflix and Dailymotion. Youtube was also measured while streaming the same video in high definition. This is referred to above as Youtube HD. Different video applications were used to measure local video: MX Player, VPlayer and RockPlayer 2. All of these applications were measured while playing the videos in three different resolutions.

Each number in the graph is in amps and is the measured current consumption of the phone over a three minute period while playing either audio or video depending on the application. The phone used in all experiments was a rooted Samsung Galaxy SII. It had a modified battery that was able to be hooked up to both a signal generator to keep a constant voltage and also to a computer to read how much current it was consuming. Since the time and voltage were kept constant in all measurements, the current consumed is equivalent to the

amount of energy being used by the phone during the application usage over those three minutes. This is because energy usage is equal to the product of current, voltage and time. For audio, the measurements were taken with the screen off. While measuring the energy usage in video applications, though, the screen had to be left on. The screen was left off for audio so that the power consumption used by the screen did not affect the measurement since the screen does not need to be used while simply listening to audio. Considering the low energy consumption of the audio applications and the high energy usage of modern smartphone screens, leaving the screen on would have greatly influenced the results and would have been misleading in the amount of energy used specifically by the CPU.

The Max row in each table refers to the test being performed with the Performance governor running. That is, the CPU is being run at the maximum frequency for the entirety of the measurement. The maximum frequency on the phone being used (Galaxy SII) is 1512 MHz. The Min row in each of the tables refers to a test that is performed with the Powersave governor running. This means that the CPU is being run at the minimum frequency for the whole measurement. While using this governor, it is likely that the performance of the application will be below the standard expected by smartphone consumers. Regardless, it is an important baseline because it represents the minimum energy usage possible while using each application. On the phone used for each measurement the minimum possible frequency is 192 MHz. The Default row refers to the default CPU governor on Android devices, which is the OnDemand governor. The Max/min row describes the ratio between the energy usage using the Performance governor and the energy usage using the Powersave governor. Likewise, the Default/min row is the ratio between the energy usage using the default OnDemand governor and the energy usage using the Powersave governor.

If we consider the difference between the Max/min value and the Default/min value, the smaller the difference, the more time the OnDemand governor is spending at maximum frequency and therefore the greater the load on the CPU. In other words, the closer the Default/min value is to the Max/min value, the more OnDemand is acting like the Performance governor in that the maximum frequency is required at all times. In the cases where this behavior occurs, a custom governor would not be beneficial for energy savings. This is because OnDemand has very small overhead in that it has a very simple trigger for ramping up the frequency of the CPU based only on load. If the overhead of the governor was increased like it would have to in a custom governor specific to multimedia as suggested above, it would decrease performance without decreasing energy savings which is undesirable. Through the data, it is clear that while some of the more intense video playing has a difference in these two values approaching zero, there are none that actually reach it. This indicates that there is no application that must run the CPU at its maximum frequency for the duration of the three minute period. Along with this, the fact that jumping back and forth between low and high frequency when the trigger of the OnDemand governor is met is very energy inefficient shows that even when the difference is quite low, there are still possibilities for energy savings. This is because it would be more efficient to use a governor that could determine a more precise estimate of the frequency necessary to deal with the current load based on something like the amount of data in a specific buffer. These energy savings would be accentuated in a case where the difference between Max/min and Default/min is large such as when only audio is played. This is because a large difference in these two values indicates that the OnDemand governor is not often at its maximum frequency. This means that the applications require very little of the CPU and jumping to the maximum frequency when load increases slightly becomes a large waste of energy when it

would only need to step up slightly to handle the increase in these situations. Here is the prime example of where a multimedia-specific CPU governor would provide large energy savings increases for an application. This is reinforced by the fact that during these tests, the audio applications worked with only small interruptions using the Powersave governor. Again here, monitoring buffers to determine the exact interval to increase the CPU frequency to would be much more efficient, especially since the steps would likely be small since the load increases do not warrant moving to the maximum frequency.

We attempted to create this multimedia-specific governor in two different ways. Both methods were focused around the monitoring of buffers in order to determine when and how to ramp the CPU frequency. The first method attempted was to directly read the hardware buffer information in the corresponding device in /proc on the system. This was successful for the sound card and the audio buffers could be directly read to determine when the CPU needed to be ramped and by how much. By monitoring the traffic on the card so directly, this governor would allow for quite exact manipulations of the CPU frequency to deal with different increases in the audio buffer size. The problem with this attempt was that it did not work for video applications. This is because the hardware buffers for the video card are not accessible like the hardware buffers for the audio buffers. They are not accessible because they are specific to the video card which are manufacturer specific in each phone and which are not part of the open-source system that Android provides. Therefore, even if it was possible to monitor this buffer, it would likely not be portable to other devices with different video cards and so another method was sought out for the governor.

The second attempt still dealt with the buffers relevant to multimedia but attempted to access them in a different manner. This time, we instead attempted to instrument the Java

libraries that were used by the Android applications to manage the multimedia. First, the specific libraries used by each application had to be found since multiple are available in the Android libraries. In order to make this method viable, it was necessary to have a small number of libraries that were used by most of the applications so that the governor did not have to track too large a number of calls. Fortunately, this was the case and in fact only two media libraries control almost all of the applications used in this study. To find what libraries each application implemented, adktool was used. This tool allowed the decompilation of the applications so that something very close to the source code could be viewed. Through a combination of grep commands and searching through the source code itself, it was possible to find the media libraries used by all of the applications used. As can be seen in Table VIII, twelve of the fourteen applications use the Android/media/MediaPlayer library which can control both audio and video playback on an Android device. Three of these twelve also use the Android/media/AudioTrack library which is more specific to audio playback. One of the fourteen applications (VPlayer) uses only the AudioTrack library and one of the applications (Firefox) uses neither library. Since VPlayer also plays video, it must use its own libraries to manage video playback. Likewise, Firefox must have custom libraries that handle audio and video plays in the browser. After determining the libraries in use, it was necessary to determine where these library calls hooked into the native code in order to track the actual buffer objects being passed down. Once this was found, the buffers could be monitored in this way for both audio and video. Upon implementing it, though, this method proved to have far too much overhead to be viable. This caused major performance issues with even audio only applications since the switches in CPU frequency were not coming fast enough after an increase in load on the CPU. This is unacceptable in a governor for multimedia since quality must be maintained along with energy savings.

It is clear through the data that a multimedia-specific governor could provide large increases in energy savings for these types of applications. The attempts at implementing a multimedia-specific governor provide insight into the necessities of such a governor. First, the monitoring of buffers in order to determine more specifically when and how to scale the CPU remains a promising idea. This must be done in a manner that allows for the efficient monitoring of both audio and video buffers and also in a way that does not create too much overhead to allow the CPU to respond to increases in load in a timely manner.

References

[1] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. 2011. Falling asleep with Angry Birds, Facebook and Kindle: a large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (MobileHCI '11). ACM, New York, NY, USA, 47-56.

[2] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. 2004. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the 2004 international symposium on Low power electronics and design* (ISLPED '04). ACM, New York, NY, USA, 174-179.

[3] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. 2004. Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times. In *Proceedings of the conference on Design, automation and test in Europe - Volume 1* (DATE '04), Vol. 1. IEEE Computer Society, Washington, DC, USA, 10004.

[4] V. Keller and R. Gruber. 2010. One joule per gflop for blas2 now! In *ICNAAM 2010*, 1321–1324.

[5] Haklin Kimm, Sung Shin, and Chang Oan Sung. 2007. Evaluation of interval-based dynamic voltage scaling algorithms on mobile Linux system. In *Proceedings of the 2007 ACM symposium on Applied computing* (SAC '07). ACM, New York, NY, USA, 1141-1145.

[6] Wen-Yew Liang and Po-Ting Lai. 2010. Design and Implementation of a Critical Speed-Based DVFS Mechanism for the Android Operating System. In *Embedded and Multimedia Computing (EMC), 2010 5th International Conference*, 1-6.

[7] Alexander Maxiaguine, Samarjit Chakraborty, and Lothar Thiele. 2005. DVS for buffer-constrained architectures with predictable QoS-energy tradeoffs. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (CODES+ISSS '05). ACM, New York, NY, USA, 111-116.

[8] Koji Miyauchi, Taro Sugahara, and Hiromi Oda. 2009. Relax or study? A qualitative user study on the usage of live mobile TV and mobile video. *Comput. Entertain.* 7, 3, Article 43 (September 2009), 20 pages.

[9] Robert Schöne and Daniel Hackenberg. 2011. On-line analysis of hardware performance events for workload characterization and processor frequency scaling decisions. In *Proceedings of the second joint WOSP/SIPEW international conference on Performance engineering* (ICPE '11). ACM, New York, NY, USA, 481-486.

[10] Wanghong Yuan and Klara Nahrstedt. 2004. Practical voltage scaling for mobile multimedia devices. In *Proceedings of the 12th annual ACM international conference on Multimedia* (MULTIMEDIA '04). ACM, New York, NY, USA, 924-931.

[11] Wanghong Yuan and Klara Nahrstedt. 2006. Energy-efficient CPU scheduling for multimedia applications. *ACM Trans. Comput. Syst.* 24, 3 (August 2006), 292-331.