Precise Power Characterization of Modern Android Devices

Wei Lin
Carnegie Mellon University
Pittsburgh, PA
<weilin@andrew.cmu.edu>

Joshua A. Wise
Carnegie Mellon University
Pittsburgh, PA
<jwise@andrew.cmu.edu>

December 13, 2010

http://moroso.emarhavil.com/~joshua/743wiki/

Abstract

The authors perform a novel, *precise*, characterization of the power performance of modern Android cell phones. Finding the battery life on phones such as HTC's EVO 4G inadequate, the authors intend to answer the question: where is the power going, and what techniques can be used to make more of each joule? Existing tools (Intel's PowerTOP and the Android built in power meter) are demonstrated to be inadequate for this end, and an alternative mechanism of accurately measuring the power of each independent device on the system is shown. Additionally, the authors design easily-buildable hardware to perform the precise power measurement, as well as a series of analyses that can be performed given the recorded data to motivate a Dynamic Voltage/Frequency Scaling (DVFS) focused approach to performance.

1 Introduction

With society's increasing dependency upon mobile cellular devices to perform functions beyond simply making a telephone call, the cell phone itself has evolved to into an entirely new device–the smartphone. The desire for speed and more integrated functionality has led these devices to sport bigger and more vibrant screens, higher resolution cameras equipped with flash, and support for the hundreds of thousands of little applications (apps) that a user can dream of. To keep up with communication, the modern device also comes equipped with a wide assortment of radios including 802.11n ("Wi-Fi"), Bluetooth, WiMAX, and LTE – all in addition to the basic CDMA/GSM stack. However, all this comes at a price; what used to last a week detached from a wall outlet now barely survives the better half of a day. This phenomenon is especially apparent with the Android line of cell phones; on many of these devices, thirty hours of standby time is almost unheard of. It's true that battery density has failed to scale linearly with performance, but that doesn't explain the standby performance – where is all this power going?

In its current state, power consumption modeling for the Android platform is in its infancy. There exists the official application that ships with the operating system, and there also exist a few third party power instrumentation tools. The current state of some of these tools is detailed in section 2; in short, though, they simply paint an incomplete picture of the power consumption of an Android phone.

Given the inadequate nature of the existing tools, as a subgoal, we aim to more accurately model the power draw of the Android platform; in section 4, we describe a device to do just that. In doing so, we distinguish between the power required by the software stack (by waking the processor from idle via interrupts) and the devices integrated within the hardware stack. This allow the attribution of the poor battery life of the platform to a source that is disproportionally drawing power.

We analyze our results in section 6, and draw conclusions as to optimizations to the existing software stack that can take these results into consideration. Finally, we consider the limitations of our work in section 7, and conclude in section 8.

2 Prior work

2.1 PowerTOP

In the early days of energy-aware Linux optimizations, Intel developed a wakeup-analysis tool called PowerTOP [2]. In Linux kernel version 2.6.21, support was introduced for a "tickless kernel", in which the CPU is not to be regularly woken up; prior to that version, the kernel

Figure 1: An HTC EVO 4G in its standard configuration.

would cause the CPU to wake from a sleep state on every tick of the system timer. With the "tickless kernel", the system timer is *disabled* while the system is blocked on I/O (i.e., there are no processes ready to run). In early iterations of the tickless kernel, drivers were poorly optimized for a system in which wakeups cost power, and would perform inefficient tasks such as busy-waiting for hardware.

Intel's PowerTOP is a tool designed to monitor the causes of wakeups on a tickless system; by narrowing down the sources of wakeups, a user (or developer) can modify his system to avoid drivers and behaviors that result in poor power performance. PowerTOP, as a secondary function, can monitor various metrics of a CPU's sleep state ("P-states" and "C-states"); in the case of the Android platform, these apply less, as the ARM CPUs do not have this functionality.

PowerTOP gives a clearer picture of what is consuming power than a simple CPU graph, but no actual estimate of current power being consumed is available; the view that it provides is incomplete to fully diagnose a system with poor battery life. Similarly, although PowerTOP is capable of measuring wakeups per second, it provides no data as to the power impact on any given machine; we differentiate our work by giving a solid numerical base on which to optimize.

2.2 Android built-in tools

The built-in Android application [1] is by far the most ubiquitous power monitoring application. (See Figure 2.) Unfortunately, its utility is substantially limited. It shows graphs detailing devices and significant applications that have been run on the phone and its contribution to the total energy consumed since the last charge cycle in percentage form; however, it does not show any indication of the current power draw of the device or even a prediction of the amount of battery life remaining.

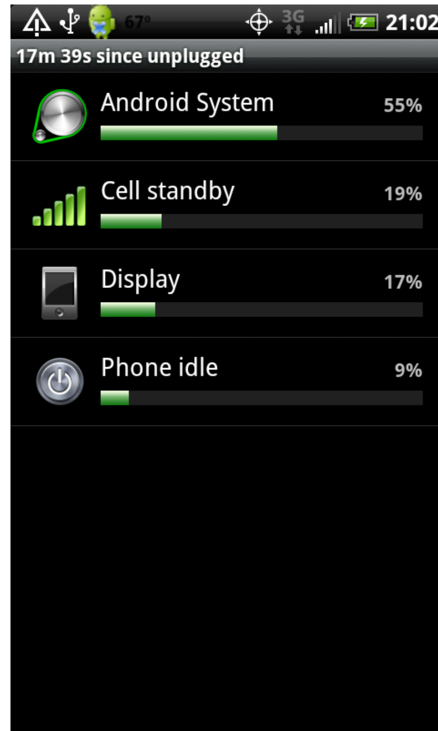

Figure 2: The Android platform's built in battery monitor.

The built-in application can be customized by vendors due to the open source nature of the platform to tweak the constantsfor each device, but there is evidence that the usage of this feature is imprecise at best. (Indeed, the data presented itself to be somewhat suspect; on the HTC Incredible, it showed a 4.3" AMOLED screen to be contributing only 7% to the total energy consumed.) We intend to build on the themes of this utility, but bring the mechanisms used to a higher degree of accuracy and calibration.

2.3 OverclockWidget

It is fitting to mention in passing the abundance of utilities for Android along the lines of OverclockWidget [6]. OverclockWidget is typical of this family of tools in that it allows the user to specify a minimum and maximum DVFS setting for the CPU (set using the `cpufreq` interface to the Linux kernel) for both the case in which the screen is on and the case in which the screen is off. OverclockWidget, however, does not provide guidance as to the optimal settings for any given device, or any given workload.

User communities commonly hold the belief that an optimal solution for increasing battery life on an Android

device is to turn the clock rate down when the display is off, and to enter a maximum-power DVFS setting when the display is on [7]. These claims are often given without a basis. We differentiate our work by providing numerical evidence for our conclusions, which we build to be used along with the capabilities that OverclockWidget can provide.

3 Novel approach

To measure the power draw of the device, a mechanism was built to analyze current flowing into the phone. This device involves using a lab supply to provide a "virtual battery" for the phone (to avoid having a physical battery buffer current spikes); current is measured and plotted versus time using a sense resistor and a small microcontroller (ATTINY) to log readings over time. A more in depth description of the hardware, as built, can be found in section 4.

With the hardware analyzer, the power draw of each salient component of the phone is isolated and measured; a more substantiative description of tests run is outlined in 5. The result of the data collection is a precise analysis of the power draw of each of the software-controllable components of the EVO 4G; specifically, the impact of the CPU and radios on battery life is condiered, and suggestions are provided for applications. This analysis should help pave a pathway for future research in runtime analysis of software-caused power usage above and beyond that provided by the existing Android stack.

4 Hardware design

The core work underlying this system is a *battery interposer* module that sits in between the target device and a benchtop power supply. The interposer serves three purposes: in brief, it *conditions* power to the device, it *measures* power consumption by the device, and it *emulates* a battery to allow the device to boot.

4.1 Power conditioning

Power to the phone is regulated and conditioned from the bench supply using a simple linear regulator – specifically, a TO-220 package LM317T. The input voltage to the board is specified as being approximately 7V; the board outputs 3.9V to the phone. (This provides a comfortable dropout range for the LM317T, and suitable headroom for the op-amps, which are incapable of swinging rail-to-rail.) A linear regulator was determined to be sufficient; the device's average case current consumption is around 300mA, which would result in the dissipation of 930mW in the voltage regulator.

As the current consumed by the device increases, the power regulator maxes out at dissipating 3.875W if the load is drawing 1.25A. This results in linearity errors in the measurement modules from heat dissipated; the regulator's heat sink has been measured at in excess of 85°C at such high power levels. The linearity errors at these power levels were judged to be an acceptable source of error, especially in the face of the design time speedup and simplicity improvements from using the LM317T.

4.2 Power measurement

Power consumption on the phone is measured indirectly by measuring the voltage across a 10 mΩ sense resistor that is in series with the target phone. The sense resistor is placed on the high side of the phone (i.e., in series with the $V+$ lead, as opposed to the $V-$ lead); this was done to avoid issues that may arise when grounding the phone via a USB lead.

A simple differential amplifier circuit built from an LM741 is used to amplify the small voltages present across the sense resistor; at 1.25A, the voltage drop across the sense is only 10.25 mV. A gain of around 50x - 100x was chosen more or less arbitrarily given the parts available. Some attention should be given to how the LM741 is powered; since the LM741 is not capable of swinging its outputs from rail to rail, the power rails must be comfortably above the 3.9V input (and comfortably above the 5V maximum differential output), and comfortably below ground. For that reason, the 7V input to the measurement board is used as the $V+$ input to the op-amp. To generate the negative voltage for the op-amp, a Mi-

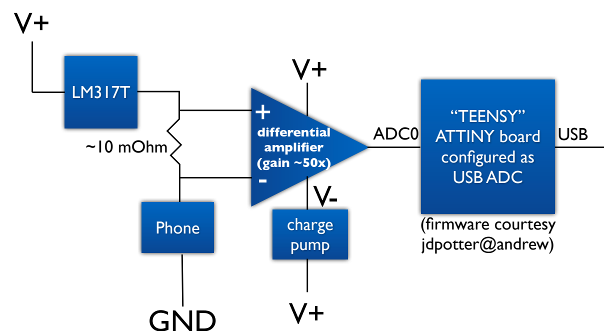

Figure 3: Block diagram for the hardware components.

crochip TC7662B integrated charge pump controller was employed to generate −7V given the input 7V. The rails on the TC7662B were filtered using a randomly selected electrolytic capacitor.

Once the small voltage has been amplified, it is then passed through an RC filter into a "TEENSY" board featuring an ATMEGA32U4 USB microcontroller. The microcontroller is loaded with firmware (courtesy Jacob Potter) that samples the ADC0 input at a rate of approximately 15Hz, and reports the raw readings back over the USB interface; since the input samples at 15Hz, the RC filter is tuned to have a time constant of approximately 0.2 seconds, which should avoid sample aliasing issues. The TEENSY obtains its own power from USB, and needs no regulation from the system.

4.3 Battery emulation

Since Android phones are capable of drawing more than the 2.5W specified by USB, the Android platform must always positively detect a battery's presence to buffer surges. On EVO 4G, the battery contains a small, undocumented, microcontroller attached to the two pins between $V+$ and $V-$ on the battery; presumably this microcontroller stores some state-of-charge information and can also read temperature from the battery. Without this microcontroller's presence, the platform will refuse to boot, and if the microcontroller is removed from a live system, the platform will immediately power down.

Since emulating this custom microcontroller is out of the scope of this project, the microcontroller from a real battery is used with a "piggy-back board" – a battery designed for the system is inserted into a board that connects through only the D_0, D_1, and $V-$ pins to the Android host. In this way, no power is supplied from the battery, but presence detection still functions.

4.4 Completed system

The prototype system was built on a piece of copper-clad perfboard ("donutboard"). In order to fit the assembled hardware into the phone's battery header, no solder could exist on the bottom of the board (indeed, nothing at all would be permitted to extend through the bottom of the board). To assemble the system, then, all components *and wires* must exist on the top side, which is the copper-clad side. The first prototype system can be seen in Figure 4.

4.5 Calibration

The ADC readings have been calibrated to real current and voltage values (and hence power readings) with the aid of an electronic load. The electronic load was set to a range of constant current values, starting at no current, and ending at 1.25A. The resulting ADC values (and voltage at the load) were graphed with respect to the delivered current; the results can be seen in Figure 5. Given a linear regression of these data, we derive the following equation for power in Watts as a function of ADC counts.

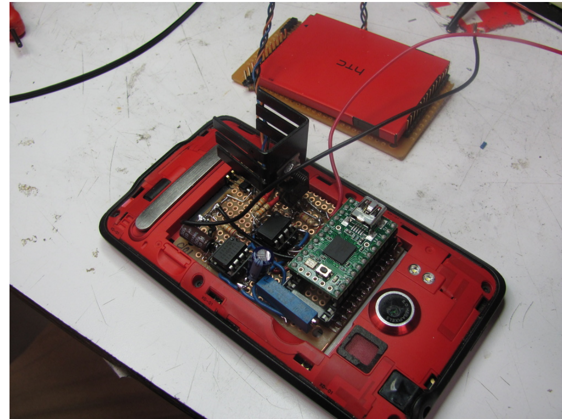

Figure 4: The first assembled prototype system installed in an HTC EVO 4G.

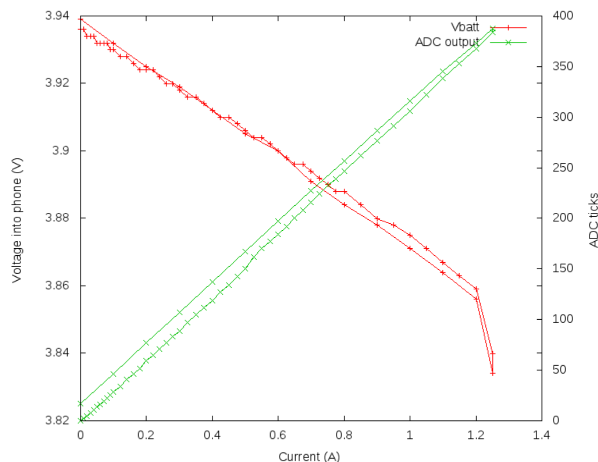

Figure 5: Calibration data from the ADC.

$$P(x) = (0.0032 \cdot x + 0.0081) \cdot (-0.0002 \cdot x + 3.936) \quad (1)$$

5 Data collection

In this section, we discuss the various components of the EVO 4G that were tested, and the mechanism by which

we tested them. We mention a mechanism to provide a "baseline" reading by which to compare all other data, a mechanism to characterize the radios, and a test suite that effectively characterizes the impact of the CPU on power consumption.

5.1 Baseline

The baseline configuration for the phone has the phone drawing as little power as possible; as few of the system peripherals as possible are enabled. Specifically, the CPU is set to the lowest DVFS setting (240MHz, with all system voltages determined by the kernel's default DVFS tables); the display's backlight is turned off; the CDMA radio is set to "airplane mode"; the WiMAX, 802.11n, and Bluetooth radios are disabled; and the system is set to attempt to draw no power from the USB. All running applications are "force-stopped" before the tests begin. This configuration provides an absolute minimum system that can be commanded by USB to run various benchmarks, and can be representative of a phone in standby, but awake.

The baseline configuration provides an important figure by which to judge the relative consumption of all other peripherals; it provides an *absolute* basis by which to determine the exact power draw of any component, but it also provides a *relative* basis by which we can judge improvements, in an analogue to Amdahl's Law.

5.2 Communication radios

The communications radios are analyzed in terms of a range of settings. Working from the hypothesis that the primary source of power draw in the radio is from being active at all, we measure the system's power consumption with each radio off; on, but with data dormant; and on, and with varying levels of activity. We test this on each of the three WAN networks – CDMA, WiMAX, and 802.11n.

To simulate various levels of upload and download activity, we use a traffic generator on both the device and a computer connected to the Carnegie Mellon network; each system sends UDP packets of a predetermined size, at a predetermined rate, to the other. By doing this, we can determine if the driving force behind radio power consumption is simple dormancy, packet rate, or data rate.

5.3 Application processor

Power consumption on the processor is analyzed by varying the running DVFS setting and stressing different subsystems on the CPU's applications processor (AP) core. For all discrete voltage/frequency steps the following benchmarks are performed:

- **Dhrystone[4]**: integer arithmetic stress test
- **LINPACK[3]**: floating point unit (FPU) stress test
- **STREAM[5]**: memory stress test
- power correlation with CPU wakeups per second from sleep

The application processor does not provide a continuously variable DVFS set; the available settings between 240MHz and 998MHz are quantized in increments of roughly 38MHz. The AP voltage cannot be varied independently of the frequency setting; the voltage is linearly scaled between 1.050V and 1.300V depending on the frequency.

We also analyze the hypothesis presented by [2] that power consumption has a strong correlation to application processor wakeups per second, and that batching up work to reduce wakeups can improve system power consumption. We designed a microbenchmark that simply schedules itself repeatedly at a predetermined interval; this microbenchmark is executed at wakeup intervals of 1Hz, 10Hz, 100Hz, and 1kHz at all DVFS settings.

6 Results

In this section, we briefly discuss our results, and the conclusions that can be drawn from them. We begin by discussing the system's baseline setting. We then proceed onto a discussion of the WiMAX, CDMA, and 802.11n radios; from there, we discuss the results of the CPU and memory system benchmarks.

6.1 Baseline

After setting the phone to the baseline configuration, the system draws 110mA, or approximately 430mW. This would cap the system's battery life, on a 1400mAh battery, at approximately 12.7 hours if the system were constantly awake; this falls short of the weeks provided by a classical "feature-phone". Especially with the consideration that the 110mA specified here is under no load whatsoever, this motivates the need to aggressively sleep the system in order to maintain a reasonable battery life.

6.2 Communcation Radios

With the system in the baseline configuration and the WiMAX radio (4G data) searching for a signal, the system

dissipates 685mW. Once the connection has been established, the system dissipates 500mW with the radio idling. Comparing this to the baseline results, there is a increase of 70mW.

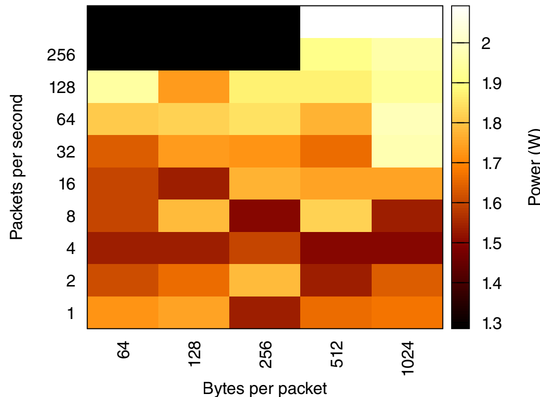

Figure 6: Power graphed versus WiMAX downloads.

The system power consumption is then measured with the radio receiving data. The results of the measurements can be observed in figure 6. Each power measurement represents the average power draw of the device over a short period of time receiving packets of a particular size at a particular frequency. Looking at the figure, there appears to be very little correlation between power consumption and data rate. The only trend that can be observed is that lower rates tend to require less power. This observation can be made independently of packet size and packet frequency. This is counter-intuititive as larger packets should keep the DSP active longer and thus should result in higher average power consumption. It is evident that this effect is negligible compared to the noise in the system.

The system power consumption during radio transmission can be observed in figure 7. Here there results are much more inline with expectations. It is interesting to note that power consumption of radio transmission starts at a power consumption level that is equal to the maximum power consumption of radio reception and goes as high as 7W. The conclusion from this is that radio transmission hurts independent of the speed in which data is being transmitted.

For the Wi-Fi radio, the results appear to be bimodal. When the radio is on, the device dissipates 730mW; an increase of 230mW over baseline. This was true for radio reception, broadcast and idle. Varying packet size and frequency had negligible impact on power consumption relative to the noise in the system.

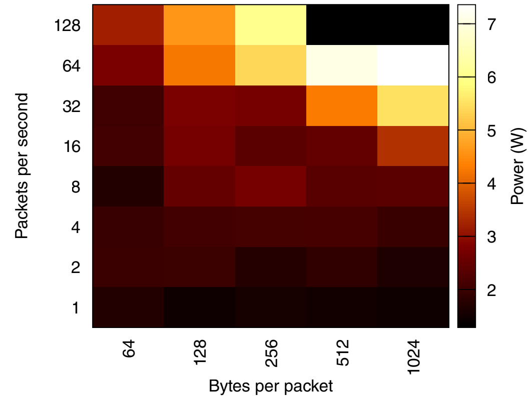

Figure 7: Power graphed versus WiMAX uploads.

The same conclusion can be drawn for the 3G radio. Varying the packet size and frequency of radio reception resulted in insignificant data. When the radio is on and receiving data or idle it draws 1.035W, while with the radio dormant, the system draws 509mW. Variations in transmit or receive appeared to change power consumption only by around 50mW.

Unfortunately for the 3G data radio, radio transmission could not be tested due to limitations of the current probe hardware; please see section 7 for more details.

6.3 Application processor

The experiments described previously on the relationship of the CPU's DVFS settings and performance per watt were carried out. We motivate these experiments primarily by the need to maximize the amount of work performed per joule expended. To analyze the hypothesis provided by many Android users that the clock should be turned down when the display is powered off, we analyze the results of each benchmark in terms of both performance per watt, *and performance per watt over idle* – that is, performance per additional watt consumed when the system is already idling at baseline.

If the hypothesis that the clock should be turned down when the display is off is true, we will see that the optimal performance per watt is with the clock at its minimum setting; performance per watt over idle is irrelevant in that case, since the system will not otherwise continue to idle after task completion. If the hypothesis that the clock should be maximized when the display is on is true, we will see that the optimal performance per watt over idle is with the clock at its maximum setting.

6.3.1 STREAM

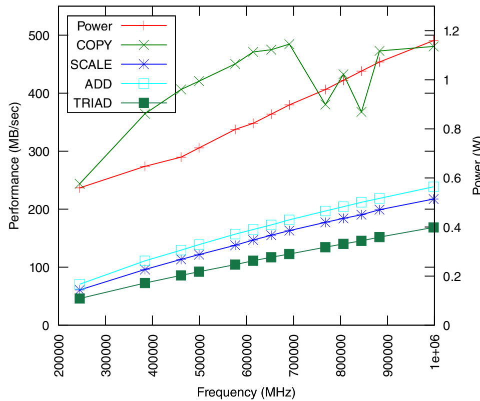

Figure 8: Each STREAM benchmark with respect to the power consumption.

We begin by observing the results of the STREAM benchmark. STREAM consists of four independent tests, called COPY, SCALE, ADD, and TRIAD; COPY stresses only the memory system, while the other tests also insert a dependency on the floating point unit. To elucidate the differences between these tests, figure 8 is provided. We also look to figure 9 for a view of STREAM performance *in aggregate*, with respect to the two performance-per-watt figures discussed above.

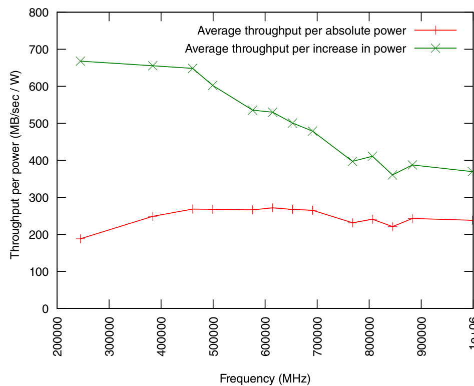

Figure 9: STREAM in aggregate with respect to power.

From these results, we find that on the STREAM test, the opposite of the hypothesis commonly stated is true; the ideal case when the system is attempting to get back to sleep as quickly as possible is to hit the highest DVFS setting, and when the system is awake due to user interaction, the DVFS setting should be lowered as much as possible while maintaining acceptable user interactivity.

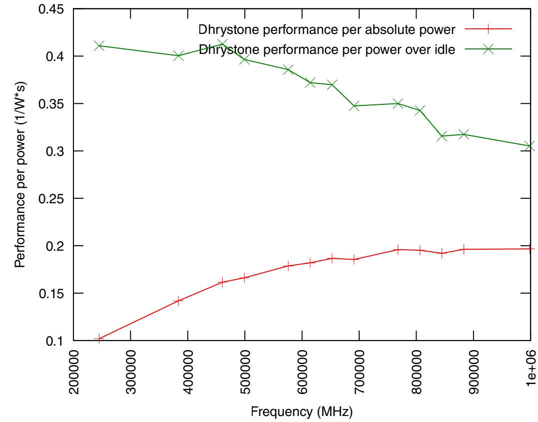

Figure 10: Dhrystone performance with respect to power.

6.3.2 Dhrystone

We also observe the results of the Dhrystone benchmark, which stresses the integer unit, and in part, the memory subsystem. Dhrystone was designed as a benchmark for early compilers, and has recently fallen out of favor for comparing systems; the basic premise within, however, is sufficient for comparing DVFS settings within one machine.

For results, we look to figure 10. We find much the similar result as we found in STREAM; the ideal case is the opposite from that which was hypothesized.

6.3.3 LINPACK

As our final benchmark, we look to LINPACK, which stresses the floating point unit on the system in a microbenchmark that solves a dense system of linear equations. It reports an estimation of millions of floating point operations per second (MFLOPS) that a computer is capable of. The LINPACK benchmark should correlate well with video game performance, or other numerical activities on a running system.

We find results for the LINPACK benchmark in figure 11. This is the third benchmark that find the same results.

6.3.4 CPU wakeups

We discussed previously the concept of optimizing for system wakeups. We ran a microbenchmark, as described in section 5, that repeatedly interrupts the processor and causes it to reschedule the running task, and we measure the change in power consumption at all DVFS settings. We represent these results in a heatmap, as shown in figure 12.

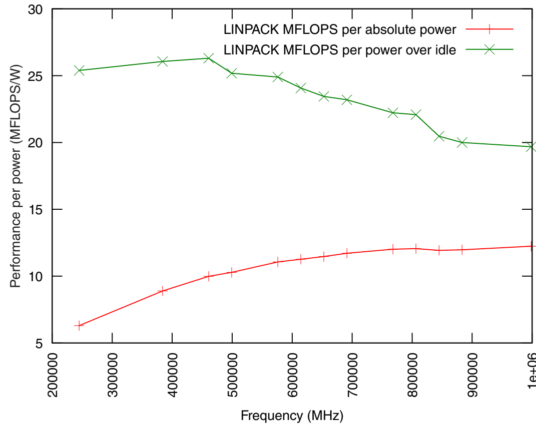

Figure 11: LINPACK performance with respect to power.

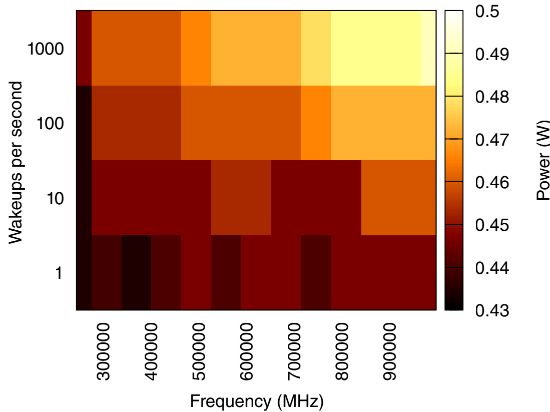

Figure 12: Power consumption with respect to wakeups per second and DVFS setting.

We find that the change in power consumption at idle from wakeups at 1Hz to wakeups at 1kHz is on the order of 6%. Compared to the nearly factor-of-three increase over idle when running STREAM, this change seems negligable. This calls into question the effectiveness of using PowerTOP on Android; by an analogy with Amdahl's Law, the space available for optimization is much greater in reducing CPU usage on sensitive mobile devices.

7 Future work

7.1 Graphics processor

A relatively large limitation of this work is that the graphics processor remains unprofiled. While we highly suspect that the graphics processor will follow the trend set by the general purpose processor, limitations in time ultimately resulted in this section being left unexplored and thus is a topic for future research.

7.2 Current probe

Two interesting misbehaviors of the current hardware were noted. First, when the system heats up as a result of heavy load, the ADC reading drifts higher; this problem is particularly notable after the system makes a sustained excursion above 0.75A or so and then drops back below 0.25A. When this occurs, the ADC may read an offset as high as 80mA above what it should. While this did not pose to be a problem during data gathering process, a more adequately cooled voltage regulator should yield more accurate results.

More serious of a misbehavior is the ability of the 3G data radio to interfere with the analog input stage. When the 3G radio is transmitting at high data rates, the output of the op-amp begins to swing negative instead of reading the true current through the sense; the ADC, then, reads zero. Attempts to shield the device proved to be futile due to the limited confines of the battery compartment of the Evo 4G. For future work, it may be wise to put the sense circuitry onto a separate daughter board for proper EMI shielding.

8 Conclusion

We hypothesized that an effective DVFS scaling strategy was different from that which was in common use on Android phones. Additionally, we suspected that the primary drivers of power consumption on modern Android phones were not those which were commonly claimed (i.e., the screen). We built a system to perform *live*, *precise* analysis on a *running* Android phone – the HTC EVO 4G – and ran benchmarks.

In this paper, we present evidence from the data that we collected that successfully validates our hypotheses. We have found that the optimal DVFS strategy on modern Android phones appears to be to scale the CPU down as much as is reasonable while the user is interacting with the phone, and to scale it up as far as is possible when the phone has no idle load. We found also that fine-grained threading is appropriate on an Android system, since CPU wakeups per second do not directly influence power consumption as severely as they do on modern *x86*-based desktop computers. We present an easily-replicatable mechanism by which application developers can measure the impact that their application has on a system's battery life, and by which system developers can

take live readings of the system's behavior as it responds to user input.

In conclusion, the system that we have designed effectively lays a groundwork for future Android developers to optimize power consumption and improve battery performance. As mobile devices move towards a dual-core, and eventually many-core environment, finding effective techniques for managing Dynamic Voltage and Frequency Scaling will become increasingly important; we predict that a symbiosis between hardware and software will prove both optimal and necessary in striking a balance providing both power and performance on mobile platforms.

References

[1] Android Open Source Project Contributors, *android.os.BatteryStats*, source from http://android.git.kernel.org/?p=platform/frameworks/base.git;a=blob;f=core/java/android/os/BatteryStats.java;hb=HEAD. Accessed October 29, 2010.

[2] Intel Corporation, *"LessWatts.org – Saving Power on Intel systems with Linux – PowerTOP,"* on the Internet at http://www.lesswatts.org/projects/powertop/. Accessed October 29, 2010.

[3] Burkardt, John, *"The LINPACK Benchmark,"* on the Internet at http://people.sc.fsu.edu/~jburkardt/c_src/linpack_bench/linpack_bench.html. Accessed October 29, 2010.

[4] Weicker, Reinhold P., *"Dhrystone Benchmark: Rationale for Version 2 and Measurement Rules,"*, SIGPLAN Notices 23,8, August 1988.

[5] McCalpin, John D., *"Memory Bandwidth and Machine Balance in Current High Performance Computers,"* IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995.

[6] Cui, Billy, *OverclockWidget (Need Root)*, available from http://www.androlib.com/android.application.org-freecoder-widgets-overclock-zww.aspx. Accessed December 13, 2010.

[7] Richkid, *Rooted & Overclock Widget = 12 hours 29 mins and counting (53%)* on the internet at http://www.htcevoforum.net/f6/rooted-overclock-widget-%3D-12-hours-29-mins-counting-53%25-98/. Accessed December 13, 2010.