An Environment for Automated Power Measurements on Mobile Computing Platforms

Mladen Milosevic, Armen Dzhagaryan, Emil Jovanov, Aleksandar Milenković

Electrical and Computer Engineering
University of Alabama in Huntsville
301 Sparkman Dr., Huntsville, AL 35899
{mladen.milosevic, aad0002, jovanoe, milenka}@uah.edu

ABSTRACT

Mobile computing devices such as smartphones, tablet computers, and e-readers have become the dominant personal computing platforms. Energy efficiency is a prime design requirement for mobile device manufacturers and smart application developers alike. Runtime power measurements on mobile platforms provide insights that can eventually lead to more energy-efficient operation. In this paper we describe mPowerProfile - an environment for automated power measurements of programs running on a mobile development platform. We discuss mPowerProfile's main functions and its utilization in several example studies based on the Pandaboard and Raspberry Pi platforms.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques.

General Terms

Measurement, Performance, Experimentation.

Keywords

Energy-efficiency, Power Profiling.

1. INTRODUCTION

Energy efficiency is a prime design requirement for mobile device manufacturers and smart application developers alike. It is driven by several key factors, including (i) limited energy capacity of batteries, (ii) cost considerations favoring less expensive packaging, and (iii) user convenience favoring lightweight designs with small form factors that operate for long periods without battery recharges.

A number of recent research studies has focused on power profiling and power estimation of mobile computing platforms. Carroll and Heiser quantified energy consumption of each component in a mobile device by performing rigorous tests and then simulating a number of usage scenarios on mobile devices [2]. Bircher and John used processor performance counters and system-specific models to estimate consumption of CPU, memory, disk and I/O [1]. Pathak et al used system call tracing and known observations of the system to generate models that can perform run-time power estimation with fine grained measurements and with low error [3, 8, 9].

Runtime power measurements on real mobile platforms are important for studies that target power optimizations or studies that aim at developing analytical models for energy estimation based on parameters derived from real platforms. Whereas several prior studies focused on capturing power traces on smartphones [2] and wireless sensor network platforms [5], they relied on manual control and post-processing to synchronize power traces with events in profiled programs. Developing an environment for automated power measurements saves time and effort and allows for accurate and fast profiling of running programs on mobile platforms.

In this paper we introduce an environment for automated power measurements of mobile computing platforms. The environment relies on minimally invasive instrumentation of a mobile platform using a shunt resistor on the power line and an inexpensive data acquisition system (DAQ) for sampling the voltage at the shunt resistor. The sampled voltage is directly proportional to the current drawn by the platform, which in turn can be used to determine power and total energy consumed. To provide automated capturing of power traces of programs running on the mobile platform, we developed a custom program called mPowerProfile. This program runs on a development workstation and interfaces both the mobile platform through a serial link and the DAQ through a USB link, thus allowing a user to configure the measurement setup and to synchronize the program execution on the mobile platform with runtime measurement on the shunt resistor. The current samples are logged into a file for further processing and analysis, and the energy consumed is calculated and logged.

The rest of the paper is organized as follows. Section 2 describes our experimental setup and gives an example for energy calculation. Section 3 describes main functions of the mPowerProfile program and its use in capturing power traces. Section 4 gives a short description of two development platforms used in testing, Pandaboard and Raspberry Pi. Section 5 describes how mPowerProfile can be utilized in evaluating energy efficiency in several example studies. Section 5.1 focuses on energy efficiency of uncompressed and compressed data transfers from/to a mobile platform. Section 5.2 examines the impact of frequency scaling on energy efficiency of data transfers. Section 5.3 focuses on power profiling of a video player program and playing a video in a web browser from YouTube. Finally, Section 6 gives concluding remarks.

ACMSE'13, April 4-6, 201, Savannah, GA, USA.

2. EXPERIMENTAL SETUP

Figure 1 illustrates experimental setup for capturing power traces and measuring the energy consumed during program execution on a system under test, typically a mobile development platform. The mobile platform is connected to a power supply (V_{SUPPLY}) via a low-resistance shunt resistor (R = 0.1 Ω). The voltage over the shunt resistor is directly proportional to the current drawn by the mobile platform (V_{SHUNT} = R*I). The voltage is sampled using a data acquisition system (DAQ) connected to a development workstation. The current, I, can be calculated from the voltage samples from the shunt resistor as I = V_{SHUNT}/R.

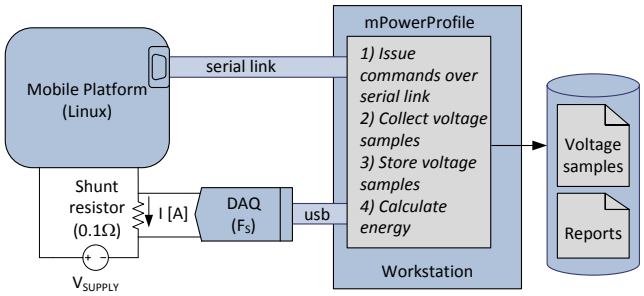


Figure 1. Experimental setup.

Figure 2(a) shows the measured current drawn by Pandaboard [7] before, during, and after compression of an input file using the gzip utility [11] with lowest compression level (-1). The compression command is issued at t=4 seconds and the compression takes about 8.5 seconds. The sampling is continued for 4 seconds after the completion of the compression task. Figure 2(a) shows the current drawn by Pandaboard during this period as it is used in our energy calculations. Figure 2(b) shows the filtered current signal, provided here only to enable easier visual inspection by a human of the changes in the current drawn during program execution.

The platform with all unnecessary services turned off draws 0.565 amperes when idling (I_{idle}=0.565 A). The start of the compression is marked with a steep increase in the current, which remains high throughout the compression and goes down to the idle current level once the compression has completed. The number of samples during the execution of a program is n = T*F_S, where T is the execution time for the given program and F_S is the sampling frequency. The total energy consumed (ET) is calculated as a function of the measured current samples I_j as follows:

$$ET = \sum_{j=1}^{n} I_j \cdot V_{PLATFORM,j} \cdot \Delta t \qquad (1)$$

where, Δt=1/F_S, and $V_{PLATFORM,j}$ = V_{SUPPLY} − I_j*R. Note that the calculation can be simplified by assuming $V_{PLATFORM}$ to be constant because the voltage drop over the shunt resistor is negligible. In addition to ET, we also calculate the energy overhead of the executing program alone, EO, which excludes the energy needed to run the platform when idle. This energy overhead is calculated as:

$$EO = ET - I_{idle} \cdot V_{PLARTFORM,idle} \cdot T \qquad (2)$$

where $V_{PLATFORM,idle}$ = V_{SUPPLY} − I_{idle}*R.

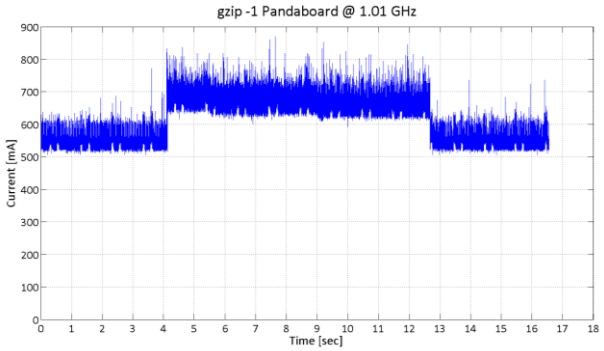
The accuracy of the energy estimation increases with increasing sampling frequency. The maximum sampling frequency supported by the DAQ in our setup is 200,000 samples per second (200 Ksps). For a processor core running at 1 GHz we can sample the voltage every 5,000 CPU clock cycles. We experimented with different sampling frequencies in the range of 10 Ksps to 200 Ksps and evaluated their impact on the energy calculations. We found that the energy calculated using 20 Ksps is within 1% of the energy calculated using 200 Ksps, so for our experiments we use a sampling frequency of 20 Ksps.

3. mPowerProfile

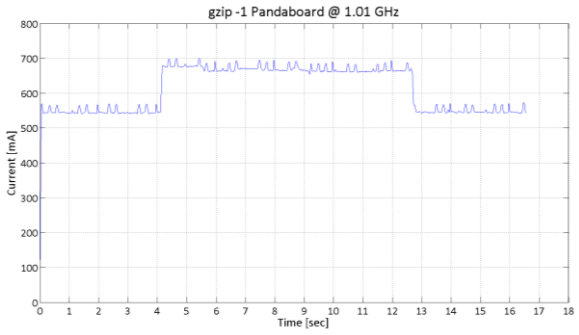
mPowerProfile is a software tool for automated capturing of power traces and evaluating energy-efficiency of programs running on mobile computing platforms. mPowerProfile runs on a development workstation and it controls both the system under test (via a serial link terminal) and the DAQ (via a USB port).

Figure 3 shows the mPowerProfile's GUI. It can operate in one of the two modes – manually controlled measurements and automated measurements. Regardless of the mode, the user first configures the DAQ channel parameters, including device and channel name, minimum and maximum voltages, wiring configuration (differential or single-ended), the number of channels (multiple channels can be sampled simultaneously), the sampling frequency, and a scaling parameter (all logged samples are multiplied by this parameter).

In the manual mode, the user selects the format (text or binary) and location of the output files where samples are to be recorded, and starts sampling by activating the Start button. Similarly, the capturing of samples is stopped by activating the Stop button.



(a)



(b)

Figure 2. Current drawn by Pandaboard during execution of the gzip utility.

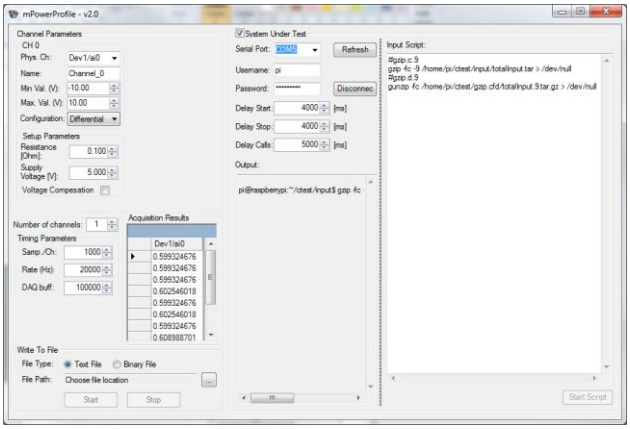


Figure 3. mPowerProfile graphical user interface.

In the automated mode, the user can capture power traces for a number of programs automatically. The user first connects to the platform under test via a serial link by specifying serial port, login parameters, and delay parameters (Start, Stop, and Call delays). The user also prepares a script in the script window by entering the shell commands for running applications which needs to be profiled for power. Each shell command is preceded by the number sign ('#') followed by the file name where the samples are to be stored. When the user activates the Start Script button, mPowerProfile takes the control and executes the commands from the script window. It starts capturing and logging samples from the DAQ immediately and waits for the Start delay to expire (e.g., 4000 ms) before issuing the first command over the serial link that will run an application of interest. mPowerProfile continues sampling during the application execution as well as during the period of time determined by the Stop delay after the application is completed. The collection of samples is then terminated and the log file is closed. The samples collected during the quite period before the application is launched can be used to determine the platform's idle current, I_{idle}. The total energy consumed, ET, and the energy overhead, EO, are calculated as shown in (1) and (2). mPowerProfile delays the processing of the next command for the amount of time specified in the Call delay parameter before repeating the previous steps for the next command. The script can include as many shell commands as needed. This way, a number of measurements can easily be taken with minimum effort from the user.

4. DEVELOPMENT PLATFORMS

4.1 Pandaboard

Pandaboard (Figure 4) is designed by Texas Instruments to support software development for smartphones and other mobile devices [7]. It features a Texas Instruments system-on-a-chip (SoC) OMAP4430 [6] with 1 GB of low-power DDR2 SDRAM. The OMAP4430 SoC includes a dual-core ARM Cortex-A9 MPCore processor, a 3D graphics accelerator, an image signal processor, and a rich set of standard peripherals (timers, communication interfaces, and a memory controller). A number of commercial mobile devices, such as Amazon Kindle Fire, BlackBerry Playbook, Motorola Droid RAZR, Samsung Galaxy Tab and Galaxy S II, are based on this chipset. Pandaboard also features an onboard 10/100 Ethernet port, a wireless interface (802.11n and Bluetooth), DVI and HDMI video interfaces, an audio interface, and two USB ports. Unfortunately, it does not support mobile broadband Internet access. The platform can run

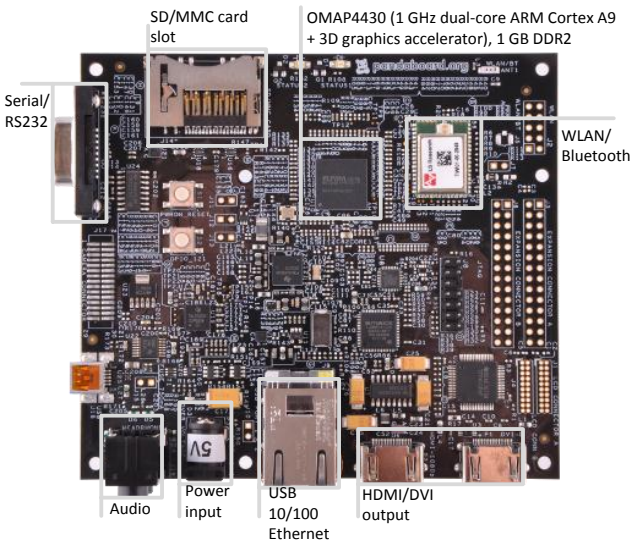


Figure 4. Pandaboard.

mobile open-source operating systems that are based on Linux, including Ubuntu, Android, and Tizen. In our experiments, we use an Ubuntu distribution provided by Linaro, a non-profit organization that works on consolidating and optimizing open-source code for the ARM architecture [4].

4.2 Raspberry Pi

Raspberry Pi (Figure 5) is a credit-card size computer that is developed in the UK by the Raspberry Pi Foundation to be a readily affordable platform for schools and aspiring young students [10]. Raspberry Pi Model B represents a lower-end device and it features Broadcom BCM2835 SoC, which contains an ARM1176JZFS running at 700 MHz, a Videocore 4 GPU, and 512MB of RAM. Model B also includes an onboard 10/100 Ethernet port, GPIO pins, RCA and HDMI video interface, an audio interface, two USB ports and SD card slot. Raspberry Pi has a large developer's community with projects ranging from entertainment centers to dedicated computers for photography, home automation, medical and robotic applications.

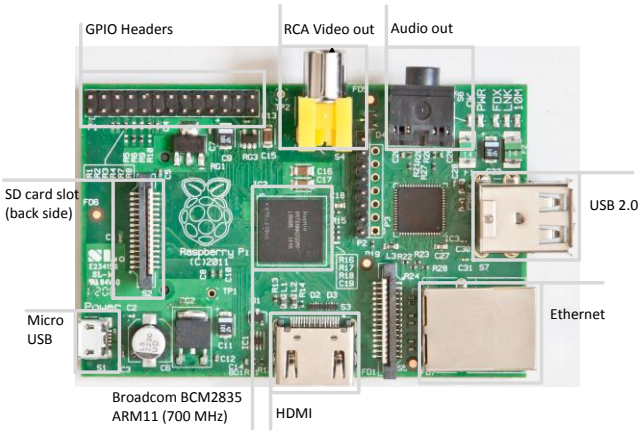


Figure 5. Raspberry Pi.

5. CASE STUDIES

5.1 Estimating Energy of Data Transfer with and without Compression

Minimizing storage capacity requirements and energy costs of data communication is of great interest for mobile platforms because of their limited storage and energy resources. Data compression utilities are thus critical in helping achieve energy-efficient data communication, reducing communication latencies, and making effective use of available storage.

In this case study we demonstrate the use of mPowerProfile in evaluating energy efficiency of uncompressed and compressed data transfers from Pandaboard to a remote server and vice versa, from the remote server to Pandaboard over a wireless LAN interface. This experiment involves measuring the energy of the uncompressed transfers as well as the energy of the transfers that involve compression and decompression tasks. For illustration purposes we consider two common compression/decompression utilities xz [12] and gzip [11] that are performed on Pandaboard while communicating with the remote server. The utilities support multiple compression levels (0 to 6 for xz and 1 to 9 for gzip), with higher levels producing smaller files at the cost of increased compute time. As an input file we use a single archive file (tar) of ~64MB that includes a text, an executable, an image, a file with comma-separated values from a wearable health monitor, and a source code.

For the compression tasks, the raw input file is read from the Pandaboard's tmpfs, compressed, streamed to the remote server over a secure channel, and the output is redirected to the null device of the remote server. By reading from the tmpfs on Pandaboard (local file system in the memory) and writing into /dev/null on the remote server we eliminate the impact of the latencies caused by reading from an SD card or by writing to hard disks on the remote server. Figure 6, line 2 shows a Linux command that carries out the compression task using xz with -4. For the decompression tasks, the compressed files are retrieved from the temporary file system of the remote server through a secure channel, decompressed on Pandaboard, and the output file is redirected to the null device of Pandaboard. Figure 6, line 4 shows a Linux command that carries out the decompression task using xz with -4. The communication between input, compression/decompression, and output operations is carried out through Linux pipes. The energies are measured for completing the entire tasks (transfers with compression and decompression). For the uncompressed upload (UUP), the raw input file is read from the local tmpfs and streamed to the remote server over the secure channel (Figure 6, line 6). For the uncompressed download (UDW), the raw input file is read from the remote server's tmpfs and streamed to the null device on Pandaboard (Figure 6, line 8).

Figure 7 shows the total energy (ET) and energy overhead (EO) in Joules for the uncompressed transfer (ET.UUP and EO.UUP) and for the transfers with compression (ET.C and EO.C) from Pandaboard to the remote server. The results indicate that the transfers with xz compression are not energy efficient – xz with -0 is the only combination that requires less energy than the uncompressed transfer. In all other cases, the compressed transfers require more energy than the uncompressed transfer. On the other side, gzip proves to be energy efficient for all compression levels except with -8 and -9. The most energy efficient combination is gzip with -1 requiring ~68 Joules, whereas the uncompressed transfer requires ~161 Joules. These conclusions hold regardless of the metric considered, the total energy or the energy overhead. These results indicate (a) gzip is indeed useful in reducing energy

```
1.  #xz_4_c.Isamples.txt (compression)
2.  xz -kfc 4 /run/shm/test/input/totalInput.tar |
    ssh armend@xeon-server "cat > /dev/null"
3.  #xz_4_d.Isamples.txt (decompression)
4.  ssh armend@xeon-server "cat
    /run/shm/xz.cfd/totalInput.4.tar.xz" | xz -kfdc
    > /dev/null
5.  #UUP_c.Isamples.txt (raw upload)
6.  cat /run/shm/test/input/totalInput.tar | ssh
    armend@xeon-server "cat > /dev/null"
7.  #UDW_d.Isamples.txt (raw download)
8.  ssh armend@xeon-server "cat
    /run/shm/input/totalInput.tar" | cat >
    /dev/null
```

Figure 6. Examples of Linux commands for compressed and uncompressed data transfers.

consumed for file uploads over WLAN, and (b) the common practice of using the default gzip (with -6) is not the most energy efficient way of uploading data from a mobile platform and thus low levels should be used instead.

Figure 8 shows the total energy (ET) and the energy overhead (EO) in Joules for the uncompressed transfer from the remote server (ET.UDW and EO.UDW) and for the transfers with decompression (ET.D and EO.D) on Pandaboard. Downloading compressed files that are streamed into decompression utilities on Pandboard proves to saves energy relatively to the uncompressed download for both xz and gzip with all compression levels. xz with -4 and -5 requires only ~42 Joules instead of ~158 needed for

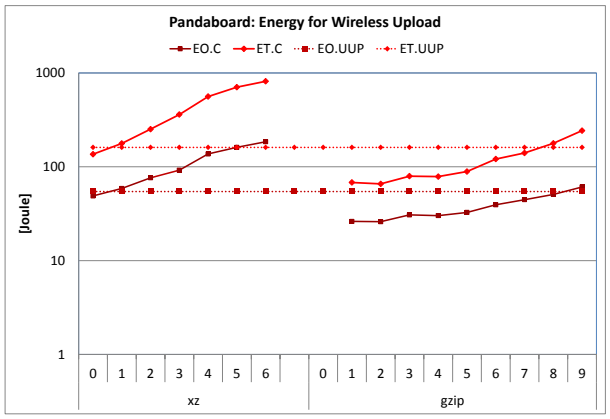


Figure 7. Pandaboard: Energy for upload over WLAN.

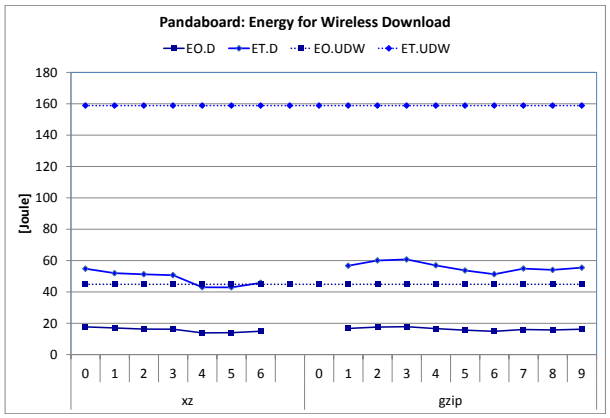


Figure 8. Pandaboard: Energy for download over WLAN.

the download of the uncompressed input file.

We repeat the same experiment for Raspberry Pi, this time using an Ethernet interface for communication to the remote server instead of WLAN. Figure 9 shows the energies for the uncompressed and compressed uploads. The results indicate that neither of the compression utilities saves the energy and that the the uncompressed file upload is the most energy efficient. Raspberry Pi's slower and weaker processor, smaller memory, and faster communication channel than in Pandaboard make the compressed uploads less energy efficient than the uncompressed ones. Figure 10 shows the energies for the uncompressed and compressed downloads. The gzip utility provides the energy savings over the uncompressed file downloads, whereas xz decompression tasks prove to be energy-wise inferior to the uncompressed downloads.

5.2 Estimating Impact of Frequency Scaling on Energy Consumed

Modern mobile platforms support dynamic frequency scaling in order to preserve energy or amount of heat generated by the processor chip. A Linux kernel infrastructure cpufreq allows for automatic scaling of the frequency up or down depending on the system load or manually scaling from userspace programs. For example, Pandaboard based on OMAP4430 chip supports the following clock frequencies: 300 MHz, 600 MHz, 800 MHz, and 1010 MHz.

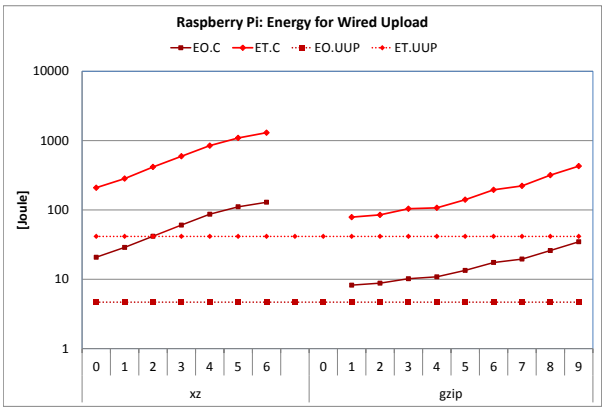


Figure 9. Raspberry Pi: Energy for uploads over Ethernet.

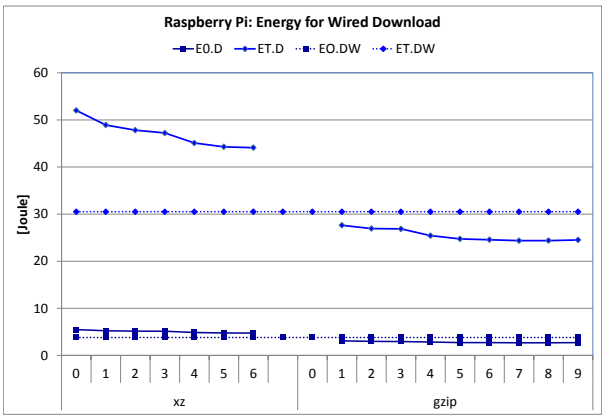


Figure 10. Raspberry Pi: Energy for downloads over Ethernet.

Figure 11 shows the energies on Pandaboard for raw and compressed uploads when the processor is running at 300 MHz. By comparing the results with the ones from Figure 7 we can observe that lowering the clock does not significantly impact the energy overhead for the uncompressed transfer (50 Joules vs. 54 Joules) or for the compressed transfers with low compression levels. However, it does increase the total energy because the compression tasks take more time to complete and relatively high idle current dominates the total energy. Similar conclusions can be drawn for the download transfers shown in Figure 12. It should be noted that the energy overhead for gzip is 60% lower when running at 300 MHz instead of 1010 MHz. In addition, the uncompressed download is more energy efficient when running at 300 MHz, e.g., EO.UDW(300 MHz) = 25.6 Joules, ET.UDW(300 MHz) = 132.86 Joules, EO.UDW(1.01GHz) = 44.87 Joules, and ET.UDW(1.01GHz) = 158.86. This indicates that uncompressed transfers that are not critical for user experience should utilize low clock frequencies.

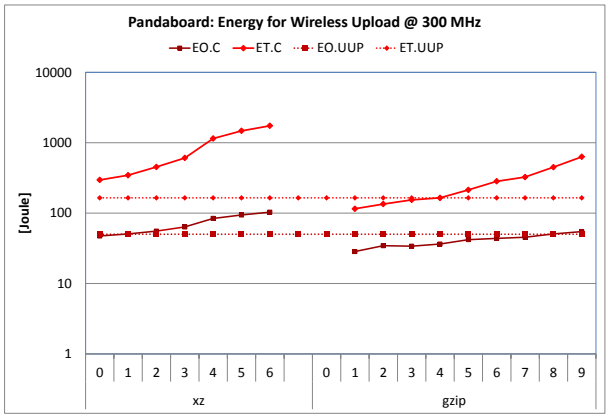


Figure 11. Pandaboard: Energy for uploads over WLAN at 300 MHz processor clock.

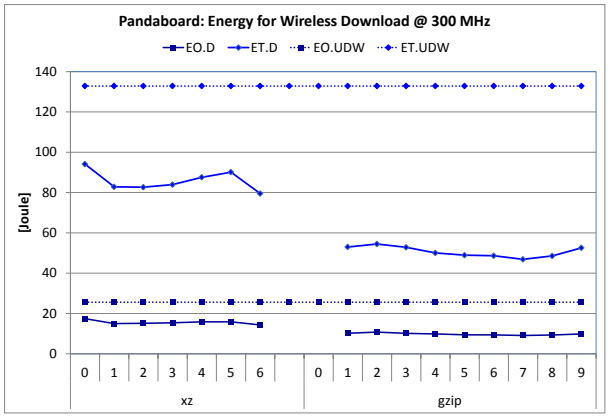


Figure 12. Pandaboard: Energy for downloads over WLAN @300 MHz processor clock.

5.3 Power Profiling of Video Playing Tasks

The proposed environment allows for concurrent capturing power traces from multiple shunt resistors. In this example, we instrumente a Pandaboard ES platform that runs at 1.2 GHz. In addition to the shunt resistor on the power supply line for the platform, we placed a shunt resistor on the power distribution line

for the processor core only. This way we can concurently record the energy consumed by the entire platform and the energy consumed by the processor core only.

Figure 13 shows the current drawn by Pandaboard ES and the processor core alone while playing a video clip from the SD card using mplayer program. The clip plays for about 12 seconds, and ET=49.8 Joules, and EO=12.2 Joules, whereas the ET(CPU)=9.8 Joules, and EO(CPU)=5.5 Joules. The results clearly indicate that the processor is responsible for only a part of the total energy overhead, and that the graphics accelator is likely reponsible for the rest of the energy consumed.

Figure 14 shows the current drawn by Pandaboard ES and the processor core while playing a Youtube video clip (overe the Etherenet) from the Firefox web browser. We can see that the current drawn by the platform is practically identical to the current drawn by the processor core. This indicates that the processor is heavily tasked when playing YouTube video.

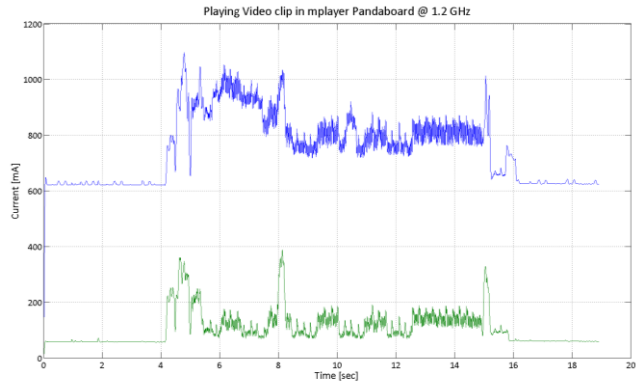


Figure 13. Pandaboard: Current drawn by the platform (blue) and processor core (green) for playing a video clip in mplayer.

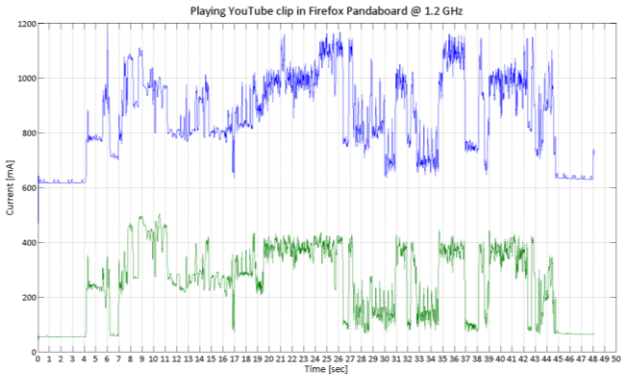


Figure 14. Pandaboard: Current drawn by the platform (blue) and processor core (green) for playing a Youtube video clip in Firefox.

6. CONCLUSIONS

This paper introduces an environment for automated power measurements of programs running on mobile computing platforms. The proposed environment relies on minimal instrumentation of the mobile computing platform and the mPowerProfile program that supports synchronized collection of power traces and automated calculation of the total energy and the energy overhead for running programs. The environment capabilities are demonstrated on several case studies.

7. ACKNOWLEDGMENTS

This work has been supported in part by National Science Foundation grants CNS-1205439 and CNS-1217470.

8. REFERENCES

[1] Bircher, W.L. and John, L.K. 2012. Complete System Power Estimation Using Processor Performance Events. *IEEE Transactions on Computers*. 61, 4 (Apr. 2012), 563 – 577.

[2] Carroll, A. and Heiser, G. 2010. An analysis of power consumption in a smartphone. *Proceedings of the 2010 USENIX conference on USENIX annual technical conference* (Berkeley, CA, USA, 2010), 21–21.

[3] Li, T. and John, L.K. 2003. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.* 31, 1 (Jun. 2003), 160–171.

[4] Linaro: open source software for ARM SoCs: *http://www.linaro.org/*. Accessed: 2012-05-28.

[5] Milenkovic, A. et al. 2005. An environment for runtime power monitoring of wireless sensor network platforms. *System Theory, 2005. SSST'05. Proceedings of the Thirty-Seventh Southeastern Symposium on* (2005), 406–410.

[6] OMAP™ 4 Platform - OMAP4430/OMAP4460: *http://www.ti.com/omap4430*. Accessed: 2012-06-02.

[7] Pandaboard: *http://pandaboard.org/*. Accessed: 2012-05-28.

[8] Pathak, A. et al. 2011. Fine-grained power modeling for smartphones using system call tracing. *Proceedings of the sixth conference on Computer systems* (New York, NY, USA, 2011), 153–168.

[9] Pathak, A. et al. 2012. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof. *Proceedings of the 7th ACM european conference on Computer Systems* (New York, NY, USA, 2012), 29–42.

[10] Raspberry Pi: *http://www.raspberrypi.org/*. Accessed: 2013-02-05.

[11] The gzip home page: *http://www.gzip.org/*. Accessed: 2012-05-25.

[12] XZ Utils: *http://tukaani.org/xz/*. Accessed: 2012-05-25.