PowerSpy: Fine-Grained Software Energy Profiling for Mobile Devices *

Kutty S Banerjee Emmanuel Agu
Computer Science Dept, Worcester Polytechnic Institute
100 Institute Road, Worcester MA 01609,USA

Abstract

Battery power capacity has shown very little growth, especially when compared with the exponential growths of CPU power, memory and disk space. Hence, battery power is frequently the most constraining resource on a mobile device. As a foundation for optimizing application energy usage on mobile devices, it is increasingly important to profile system-wide energy usage in order to accurately determine where the energy is going?. *Previous work on profiling energy usage has either required external hardware multimeters, provided coarse grain results or required modifications to the operating system or/and profiled application. We present PowerSpy, which tracks and reports the battery energy consumed by the different threads of a monitored application, the operating system, other applications in a multi-threaded environment along with I/O devices. Using PowerSpy, we are able to measure the power consumption of five diverse applications including a web browser, VRML graphics browser, compiler and video player, all without requiring modification to the application's source code.*

1. Introduction

Mobile computing has become increasingly popular as laptops, Personal Digital Assistants (PDAs) and cellular phones have experienced exponential annual growth in CPU power, available memory and disk storage. Users can check email, surf the web, edit electronic documents and playback streamed multimedia while on the move. However, even as most computing resources have experienced phenomenal growth, the capacity of batteries on mobile devices has shown very little growth in the past thirty years. Consequently, available battery power is frequently the most constraining resource on mobile devices.

As a foundation for avoiding energy-draining operations and optimizing system-wide energy usage on mobile devices, it is increasingly important to track system-wide energy usage in order to accurately determine *how the energy is being used*. Specifically, as part of our MADGRAF prototype for mobile 3D graphics [1], we wanted to develop a software module that could monitor battery energy usage in real time and inform intelligent power-conserving decisions, without requiring the instrumentation or modification of application source code. Previous work on energy profiling has either required external multimeters, offline post-processing, provided results that were too coarse or required modifications to the operating system. Generally, sampling approaches has been used for system energy profiling, in which either an external multimeter or operating system calls are used periodically to sample energy usage over time.

The goal of our research is to monitor accurately thread level and I/O device level power consumption. When an application is monitored, its energy consumption is noted at finite intervals of time. However, this measured energy is not just consumed by the threads of our monitored application but also includes the energy consumed by the hard disk, network card display, other I/O devices, and potentially the threads of other applications in a multithreaded environment. Our profiling approach involves two passes. First, battery power usage is accurately sampled at fine intervals while also tracking all system activity. We then filter (subtract out) the energy consumed by other applications, I/O devices and other *noise*, leaving us with an accurate estimate of energy usage by our monitored application.

Using PowerSpy, using only their executables, we profile five diverse commercial-strength applications including a web browser, VRML graphics browser, compiler and video player. Our results show that the network interface consumed the most power for networked applications, followed by disk I/O operations and then CPU.

2 Previous Work

The continous power sampling method for profiling application energy usage was adopted by PowerScope [4]. An external hardware multimeter with an in-built clock is used to sample the monitored computer. At each sampling time, the CPU status of the monitored computer is taken. This

*Funding was provided in part by the NSF grant number 0303592

0-7803-9305-8/05/$20.00 ©2005 IEEE

status consists of the current value of the *Program Counter* and the *Process ID- PID* along with interrupt handling details. At the same time, the multimeter records the instantaneous current, voltage values. Later, during a energy profiling post-process stage, the CPU values are associated with the multimeter readings for each sampled interval in order to reconstruct the power consumption details of the monitored computer.

Another approach to measuring I/O power consumption is to pre-determine the power consumed by each I/O device when it is in a given fixed power state[5]. Thus, if the power consumption of the hard disk is known in all its states, and the power consumption in transitions between states is predetermined, given the state of the I/O device, the I/O power consumption can be derived as in [5].

Energy Estimation is also performed by formulating fine level building blocks of which an application is composed [6]. One such way is to measure the power consumed by different procedures within the Operating System. Once we know which of these components a given application uses, we can then estimate its power consumption.

3. PowerSpy

In this section, we present the two-pass profiling approach used our energy profiling tool, PowerSpy. The functionality is presented in two stages, namely, *(1) Event Tracking* and *(2) Analysis*. Further, the Analysis stage has 2 separate passes which we shall expound on.

3.1. Event Tracking

In this stage, the application to be profiled for energy consumption is run. Simultaneously we profile this application for CPU Time, I/O Activity and Energy Consumption.

CPU Time: From the time that the application is started to the time that it ends, the thread IDs of all threads created by the application are kept in an 'in-core' database maintained by PowerSpy. The CPU Profiler records in the *cpu.log* file, the thread id of the thread being run at the end of each context switch along with the time stamp.

I/O Activity: Simultaneously, all I/O requests made to the attached devices are also recorded along with their time stamps and the request specifics in an *io.log* file. An I/O request in this context is an asynchronous call made to an I/O device. We assume that the device starts servicing the request from the time that it receives this I/O call.

Energy Consumption: The energy consumed by the system is also profiled in a separate *energy.log* file. This file contains a chronological listing of *time vs energy consumed.*

Thus, at the end of the Event Tracking stage, we have tracked and stored in three files,all CPU events, I/O events

Power Requirement	+5VDC (±5%)	+5VDC (±5%)	+5VDC (±5%)	+5VDC (±5%)	+5VDC (±5%)
Dissipation (typical)					
Startup (max peak)	4.7 W	4.7 W	4.7 W	4.7W	4.7W
Seek (average)	2.3 W	2.3 W	2.3 W	2.3W	2.3W
Read (average)	2.1 W	2.1 W	2.0 W	2.0W	2.0W
Write (average)	2.2 W	2.2 W	2.1 W	2.1W	2.1W
Performance idle (average)	1.85 W	1.85 W	1.85 W	1.85W	1.85W
Active idle (average)	0.95 W	0.95 W	0.85W	0.85W	0.85W
Low power idle (average)	0.65 W	0.65 W	0.65W	0.65W	0.65W
Standby (average)	0.25 W	0.25 W	0.25 W	0.25W	0.25W
Sleep	0.1 W	0.1 W	0.1 W	0.1W	0.1W
Power consumption efficiency (watts/GB)	0.008	0.011	0.016	0.022	0.033

Figure 1. Sample Hardware Spec Sheet

and system energy usage over time while the application was running.

3.2. Analysis Stage

In this stage, we process the data acquired at the end of the *event tracking stage* in order to recreate a snapshot of the system during the time that the application being monitored was being executed, so that we can understand the power consumption by the different parts of the system. The Analysis stage takes as input the cpu.log, io.log, energy.log event tracking files as well as a fourth file called spec.log. The spec.log as shown in figure 3.2, file contains the estimated energy required by various devices to perform specific tasks, as deduced from the specification sheets provided by the device's manufacturer. A typical spec.log entry is the energy consumed by a disk to read 1K bytes of data. The Analysis Stage is carried out in two separate passes: the I/O filtering process and the CPU thread-level accounting.

3.2.1 Pass One - I/O Filtering Process

This pass takes as input the io.log, energy.log and spec.log and modifies the energy.log file. In this pass, we filter out (subtract) the estimated energy consumed by each I/O device that ran in a given time interval, from the total measured energy consumed in that interval. The remaining energy is attributed to CPU threads.

Let us consider for example, that the energy consumed in a given time interval is *'A'* units. And that in this interval threads *a,b,c* were run (note that we carefully choose time interval *'A'* as the smallest time interval for which we notice a change in remaining battery energy). Additionally, I/O devices *e,f,g* were operating in this same interval. However, from spec.log, we know that devices e,f and g when performing certain tasks consume finite amount of energy, say P_e, P_f, P_g respectively. We can deduce the power consumed by threads a, b and c for their cpu-centric operations.

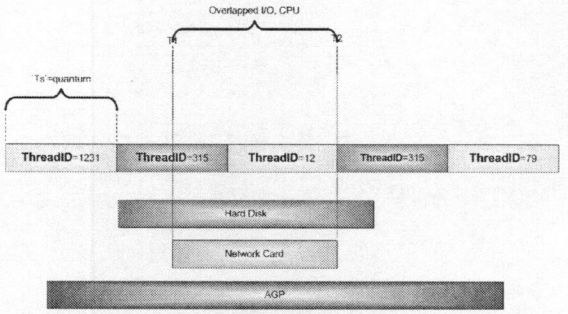

Figure 2. Overlapped CPU and I/O Operations

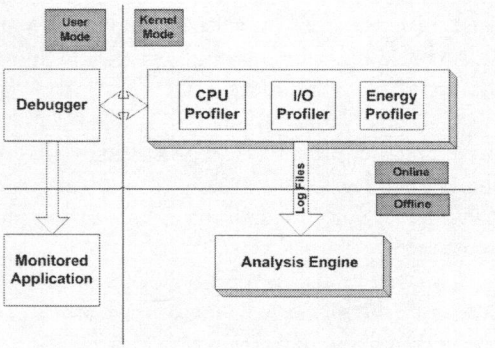

Figure 3. PowerSpy System Architecture

Mathematically, if P_a is the amount of energy consumed by thread *'a'* and (likewise for b,c,d,e,f,g), then

$$P_a + P_b + P_c = A - P_e - P_f - P_g = P_{threads} \quad (1)$$

As a concrete example of P_e, consider that the I/O request is a read request to the hard disk for a block of size 2KB. The specification sheets tell us that the energy estimate for a 1KB read operation is *'k'* units. Therefore $P_e = 2 * k$.

Figure 2 shows an example of overlapped CPU and I/O operations. Thus, at the end of this pass, we have been able to separate the I/O energy consumption from the CPU energy consumption and also get a list of the energy consumed by individual I/O devices.

3.2.2 Pass Two - CPU Thread Level Accounting

In this pass, we will isolate the energy consumed by different threads individually. This pass takes as input the energy.log and cpu.log files from the energy tracking stage. It parses the energy.log file for two consecutive energy, time intervals. It obtains a list of all thread ids from cpu.log that were run in this time range. For example, if in a time range threads with IDs 1211, 2032 and 5101 were run 5, 6 and 2 times respectively. Also the energy consumption in this time range was 50 units. Therefore, the energy consumed by the thread with id 1211 is

$$P_{1211} = 50 * (5/5 + 6 + 2) \quad (2)$$

The underlying assumption behind equation 2 being that CPU level power consumption is directly proportional to the time (or number of cycles) spent in running the thread.

4. PowerSpy System Architecture

In this section we describe our implementation of PowerSpy on the Windows operating system. Windows was chosen due to its overwhelming popularity on multiple mobile and ubiquitious computing devices. PowerSpy is targeted to work for Windows 2000, XP and Windows Server 2003. The components are as shown in figure 3.

As shown in the figure, some of the components execute in the user mode whereas the others execute in the kernel mode. Similarly, some components are operated online whereas the Analysis Engine currently operates offline but will operate online in future.

4.1. Debugger

The Debugger is responsible for initiating the application program that is to be monitored. It is responsible for keeping track of the different threads that the application spawns out. It maintains an in-core database of these threads and communicates them to the CPU Profiler. The use of a debugger process in tracking thread activity is important since under the Win32 platform, a debugger process that initiates another process has complete access over the debuggee's memory address space. Also, each time that the application spawns out a thread, the debugger is notified.

4.2. CPU Profiler

The CPU Profiler keeps track of the thread that was being run by the Operating System, each time a context switch occurs. If the thread being run matches any of the threads in the in-core database maintained by the Debugger, then it adds a flag and outputs the same to the "cpu.log" file.

The CPU Profiler communicates with the kernel mode components as shown in figure 4. The PowerSpy driver is a Windows kernel mode legacy driver [3]. Its purpose is to install a Deferred Procedure Call (DPC), which is a routine that monitors the Windows Thread Scheduler also called as *The Dispatcher Object* and is part of the Windows Executive [2]. It is important to note that since DPC objects run at the same privilege as the Windows Dispatcher, it is not controlled or scheduled by the Dispatcher and as such is in a position to monitor the activity of the Dispatcher. The DPC

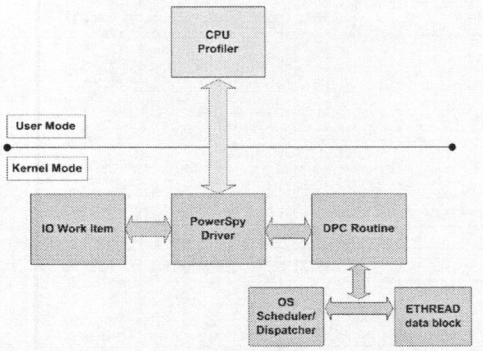

Figure 4. CPU Profiler Functional Diagram

routine is executed once every 10 milliseconds which is the normal quantum of all threads in Windows 2000. However, a context switch may occur before this time period because of thread level preemption. The "EThread" data block is an undocumented data structure maintained for each running thread by the Windows Scheduler [2].

Thus, the DPC routine gets a snapshot of the Windows scheduler. It also queues an I/O work item, which executes at a lower privilege than the DPC (since it is bad practice for long operations to be performed at exalted privileges). The I/O Work Item stores the system timestamp, and the thread ID which is the thread ID monitored by the DPC at the instance that the DPC was executed. In other words, the DPC captures the information but does not write it to the "cpu.log" file since file operations may take a long time. Rather it passes this operation to the I/O Work Item which writes to the "cpu.log" file. Figure 5 shows the sampling process. $T_1 to T_6$ are the sampled time intervals where the current thread being run is stored. Also the remnant battery power is stored.

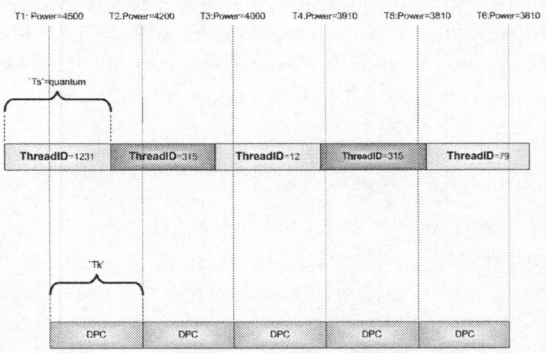

Figure 5. CPU Profiler Timeline

4.3. Energy Profiler

The energy profiler captures the amount of energy lost by the system during the time that the application is being monitored, and can be implemented either in hardware or software. As part of PowerSpy package, we provide a software solution. However, hardware measurements can easily be used instead with our existing framework.

In the hardware solutions, a digital multimeter is used to measure the instantaneous power drawn by the entire system at a high sampling frequency. PowerScope uses a hardware solution. These power values are then recorded along with their time stamps. In the software solution, we use system calls to measure the amount of battery energy remaining at a sampling frequency and record the same along with the time stamps. We now briefly discuss the pros and cons of hardware and software energy profiling.

The advantage of the hardware method is that since the energy sampling is done independent of the system being monitored, the sampling frequency can be very high and is independent of any OS activity. Also, the instantaneous power values are more accurate than their software counterparts. The software version is limited by the sampling rate at which the operating system gets updates from the actual battery, and suffers from the drawback that at very high sampling rates, the battery device may not be able to report changes in the remnant power accurately. In other words, if we sample the battery at 10 milliseconds, then considering 10 such sampled values, it may happen that the remnant battery is the same across all 10 values. However, in reality, it may be the case that the remaining battery energy has changed but is not being updated at that frequency.

For our purposes, since we wanted to develop a software module that could be conveniently integrated into our MADGRAF system [1], in which clients are mobile. In such a scenario,the deciding factor was the disadvantage that the hardware method involves external equipment and is not easily portable.

PowerSpy uses the software solution for querying battery levels. The I/O Work Item regularly queries the battery device using IOCTL_QUERY_BATTERY_STATUS ioctl. Thus, the I/O Work Item generates an IRP with the above ioctl and issues it to the battery class device driver. The battery device driver returns the remaining battery power in milli Watt Hour units.

4.4. I/O Profiler

The I/O profiler keeps track of all I/O requests send to the various devices connected to the system. Under the Windows Operating System, devices are sent asynchronous messages called I/O Request Packets (IRPs) [3]. So, we need a way of capturing all IRPs sent forth to the different devices. In order to do this technically there are two options. One is to write a upper filter driver [3] for all devices. However, writing a filter driver for all devices connected to the system is not a very scalable or feasible solution. The

Thread ID	Energy Consumed (mWh)
1329	15
133	12
291	1
9031	1
1100	1
3002	1

Device	Energy Consumed(mWh)
\Device \tcp	51
\Device\DR0	22

Figure 6. Outlook Express Energy Profile

Thread ID	Energy Consumed (mWh)
1122	9
3651	6
782	11
297	31
202	12
203	5

Device	Energy Consumed (mWH)
\Device\DR0	55
\Device \tcp	0

Figure 7. Microsoft Visual Studio results

other solution is to read undocumented structures inside the OS kernel such as reading the ETHREAD data structure in the "ntoskrnl.exe" file.(contains the xp kernel)

We have made use of a third party tool, *IRP Tracker Utility* [7] which writes all IRPs issued by the system along with their time stamps and status in "io.log" file. The time stamp used by IRP Tracker is in the hour:minute:seconds:milliseconds format. However, the time stamp used by the CPU Profiler and the Energy Profiler is in the form of a 64 bit value representing the number of 100-nanosecond intervals since January 1, 1601 . In order to convert the IRP Tracker's time format to that of the PowerSpy profilers' format, we used the Win32 system call SystemTimeToFileTime.

5. Results

In this section we present the results of experiments using PowerSpy to profile the power usage of a wide range of applications including the Outlook Express Windows mail reader, Mozilla web browser, Visual Studio IDE compiler, VRML browser and Microsoft Media player. All profiled applications were available as unmodified executables.

The laptop used for the test is a Dell Inspiron 8500 with Hitachi disk and WaveLan Wireless Card. The only devices that we profile are the disk and the wireless network card . However, the principle of I/O profiling and filtering using specification sheets can also be extended to other devices.

Also, the disk and the wireless cards have been brought into a known power state, i.e., the D0 state This was done by doing a dummy read of both the devices from user mode ensuring that the devices are in a maximum power consuming state.

5.1. Outlook Express

PowerSpy was used to profile the execution of the Outlook Express mail reader and the results are as shown in figure 6. For the above experiment, Outlook Express was started and mails were checked with the *Send and Receive* facility for a single POP3 mail account. The individual thread level power consumption *does not include the power consumed by the Operating System internal operations.*

Referring to figure 6, the power consumed by the I/O device '\Device\tcp' is higher than that consumed by any of the individual threads (power consumed by the threads as shown in figure 6 is purely due to CPU operations). The device '\Device\DR0' represents I/O access to the disk, the power consumed by which is also substantial. Thus, a major chunk of the power consumed by Outlook Express is due to network and disk access.

5.2. Microsoft Visual Studio

The power profiling performed on 'Microsoft Visual Studio' revealed its power consumption details as shown in figure 7. At the time of profiling, a sample VC++ project was opened, rebuilt and closed. As can be seen, the power consumed by the hard disk namely, \Device\DR0 is somewhat high indicating that the application tends to be more aggressive in its use of the disk, probably due to extensive the file read/write, swapping and disk writes that are typical of the compilation of projects with a large number of files.

5.3. Mozilla

Mozilla web browser was tested for its power consumption and the results are described in figure 8. A sample site at opengl.org was opened and refreshed and the power profile results were tabulated. As is evident from the fig 8, the power consumed by the tcp device is somewhat high. The disk I/O power consumption is comparatively low.

5.4. Media Player

The results of Windows Media Player profiled for power is given in figure 9. The experiment carried out with Windows Player was playing an online movie trailer. The URL

Thread ID	Energy Consumed (mWh)
123	121
124	32
125	42
313	51
417	1
1299	1

Device	Energy Consumed (mWh)
\Device\DR0	152
\Device\tcp	430

Figure 8. Mozilla web browser energy profile

Thread ID	Energy Consumed (mWh)
2379	402
3412	98
786	22
352	1
1102	1
1471	1
1472	1
1481	2
1296	2
1926	1
1106	2
1291	1

Device	Energy Consumed (mWh)
\Device\DR0	165
\Device\tcp	850

Figure 9. Windows Media Player results

of the site with the trailer was entered into Media Player and the movie started. It is very apparent, that the bulk of operation is carried out by the network device. The power consumption details reveal the same that is, \Device\tcp has a higher share in the power consumption list.

5.5. OpenVRML Browser

An open source VRML browser was used to test the power consumed by VRML files of various sizes. Figure 10 provides details about the power consumption by the browser when opening a VRML file with 10k vertices. The complexity of rendering a given VRML scene file was roughly proportional to the number of vertices in the scene description. The power consumed by the individual thread is relatively high. The initial loading and parsing of the VRML file is CPU intensive and that explains the high power consumption of a CPU work oriented thread. We also profiled 1000-vertex and 100K-vertex VRML files and found that in all cases, the amount of power consumed by

Thread ID	Energy Consumed (mWh)
1321	52
Device	**Power Consumed (mWh)**
\Device\DR0	42

Figure 10. VRML Browser (10,000 vertices)

the individual CPU work oriented thread is far higher than that consumed by any of the I/O devices. As the size of the VRML file gets larger, the amount of power consumed by the CPU thread in parsing and loading the file increases.

6 Conclusion

We have presented PowerSpy, a software tool for fine-grained power profiling on the Windows operating system. We present promising results of the power consumption of five diverse applications.

Our profiled applications consumed the most power on networked data transmission, where applicable. The Outlook Express mail reader, Media player and Mozilla web browser were all in this category. Applications such as Visual Studio that did not have significant networking activity expended power on disk I/O. Finally, the VRML browser performed many CPU operations to render a single graphics scene, which was reflected in its energy usage.

We hope that this tool encourages further work into understanding the nature and techniques for optimizing power use on mobile devices.

References

[1] E Agu, K Banerjee, S Nilekar, O Rekutin, D Kramer, *A Middleware Architecture for Mobile 3D Graphics*, in Proc. IEEE MDC, June 2005. (to appear).

[2] M Russinovich, D Solomon, *Inside Microsoft Windows 2000,* 3rd Edition, Microsoft Press, 2000.

[3] J Lozano, A Baker, *The Windows 2000 Device Driver Book* 2nd Edition, Prentice Hall, 2000

[4] J Flinn, M Satyanaranan, *PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications.* Second IEEE WMCSA 1999, New Orleans, LA.

[5] J R. Lorch, A J Smith, *Apple Macintosh's Energy Consumption.* IEEE Micro, vol. 18(6) 1998.

[6] T Li, L K John, *Run-time Modelling and Estimation of Operating System Power Consumption.* ACM SIGMETRICS '03, June 10-14, 2003, San Diego, CA.

[7] *IRP Tracker Utility, V1.3.* Open Systems Resources.