Proceedings of the Linux Symposium

Volume One

July 19th–22nd, 2006
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

Review Committee

Jeff Garzik, *Red Hat Software*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat Software*
Ben LaHaise, *Intel Corporation*
Matt Mackall, *Selenic Consulting*
Patrick Mochel, *Intel Corporation*
C. Craig Ross, *Linux Symposium*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
David M. Fellows, *Fellows and Carr, Inc.*
Kyle McMartin

Linux Laptop Battery Life

Measurement Tools, Techniques, and Results

Len Brown   Konstantin A. Karasyov

Vladimir P. Lebedev   Alexey Y. Starikovskiy

Intel Open Source Technology Center

{len.brown,konstantin.a.karasyov}@intel.com
{vladimir.lebedev,alexey.y.starikovskiy}@intel.com

Randy P. Stanley
Intel Mobile Platforms Group
randy.p.stanley@intel.com

Abstract

Battery life is a valuable metric for improving Linux laptop power management.

Battery life measurements require repeatable workloads. While BAPCo® MobileMark® 2005 is widely used in the industry, it runs only on Windows® XP. Intel's Open Source Technology Center has developed a battery life measurement tool-kit for making reliable battery life measurements on Linux® with no special lab equipment necessary.

This paper describes this Linux battery life measurement tool-kit, and some of the techniques for measuring Linux laptop power consumption and battery life.

This paper also includes example measurement results showing how selected system configurations differ.

1   Introduction

First we examine common industry practice for measuring and reporting laptop battery life.

Next we examine the methods available on ACPI-enabled Linux systems to measure the battery capacity and battery life.

Then we describe the implementation of a battery life measurement toolkit for Linux.

Finally, we present example measurement results applying this toolkit to high-volume hardware, and suggest some areas for further work.

1.1   State of the Industry

Laptop vendors routinely quote MobileMark® battery measurement results when introducing new systems. The authors believe that this

is not only the most widely employed industry measurement, but that MobileMark also reflects best known industry practice. So we will focus this section on MobileMark and ignore what we consider lesser measurement programs.

1.2 Evolution of MobileMark®

In 1995, BAPCo®, the Business Applications Performance Corporation, introduced a battery-life (BL) workload to support application based power evaluation. The first incarnation, SYSmark BL, was a Windows® 3.1 based workload which utilized office applications to implement a repeatable workload to produce a "battery run down" time. Contrary to performance benchmarks which executed a stream of commands, this workload included delays which were intended to represents real user interaction, much like a player piano represent real tempos. Because the system is required to deplete its own battery, a master system and physical interface was required as well as the slave system under test. In late 1996 the workload was re-written to support Windows95 adapted to 32-bit applications.

When Windows® 98 introduced ACPI support, BAPCo overhauled the workload to shed the cumbersome and expensive hardware interface. SYSmark98 BL became the first software only BL workload. (No small feat as the system was now required to resurrect itself and report BL without adding additional overhead.) Additionally a more advanced user delay model was introduced and an attempt was made to understand the power performance trade-off within mobile systems by citing the number of loops completed during the life of the battery. Although well intended, this qualification provided only gross level insight into the power performance balance of mobile systems.

In 2002, BAPCo released MobileMark 2002 [MM02] which modernized the workload and adopted a response based performance qualifier which provided considerably more insight into the power performance balance attained by modern power management schemes. Additionally they attempted to better define a more level playing field by providing a more rigorous set of system setting requirements and recommendations, and strongly recommending a light meter to calibrate the LCD panel brightness setting to a common value. Additionally, they introduced a "Reader" module to complement the Office productivity module. Reader provided a time metric for an optimal BL usage model to define a realistic upper bound while executing a real and practical use.

In 2005 BAPCo's MobileMark 2005 [MM05] added to the MobileMark 2002 BL "suite" by introducing new DVD and Wireless browsing modules as well as making slight changes to increase robustness and hold the work/time constant for all machines. Today these modules help us to better understand the system balance of power and performance. Multiple results also form contour of solutions reflective of the respective user and usage models.

1.3 Learning from MobileMark® 2005

While MobileMark is not available for Linux, it illustrates some of the best industry practices for real use power analysis that Linux measurements should also employ.

1.3.1 Multiple Workloads

Mobile systems are subject to different user[1] and usage models,[2] each with its own battery

---

[1]Different users type, think and operate the system differently.

[2]Usage models refers to application choices and content.

life considerations. To independently measure different usage models, MobileMark 2005 provides 4 workloads:

1. Office productivity 2002SE

   This workload is the second edition of MobileMark 2002 Office productivity. Various office productivity tools are used to open and modify office documents.

   Think time is injected between various operations to reflect that real users need to look at the screen and react before issuing additional input.

   The response time of selected operations is recorded (not including delays) to be able to qualify the battery life results and differentiate the performance level available while attaining that battery life.

2. Reader 2002SE

   This workload is a second edition of MobileMark 2002 Reader. Here, a web browser reads a book from local files, opening a new page every 2 minutes. This workload is almost completely idle time, and can be considered an upper bound, which no "realistic" activity can possibly exceed.

3. DVD Playback 2005

   InterVideo® WinDVD® plays a reference DVD movie repeatedly until the battery dies. WinDVD monitors that the standard frame rate, so that the harness can abort the test if the work level is not sustained. In practice, modern machines have ample capacity to play DVDs, and frames are rarely dropped.

4. Wireless browsing 2005

   Here the system under test loads a web page every 15 seconds until the battery dies. The web pages are an average of

150 KB. This workload is not specific to wireless networks, however, and in theory could be run over wired connections.

1.3.2 Condition the Battery

In line with manufacturer's recommendations, BAPCo documentation recommends conditioning the battery before measurement. This entails simply running the battery from full charge until full discharge at least once.

For popular laptop batteries today, conditioning tends to minimize memory effects, extend the battery life, and increase the consistency of measurements.

MobileMark recommends conditioning the battery before taking measurements.

1.3.3 Run the Battery until fully Discharged

Although conditioning tends to improve the accuracy of the internal battery capacity instrumentation, this information is not universally accurate or reliable before or after conditioning.

MobileMark does not trust the battery instrumentation, and disables the battery low-capacity warnings. It measures battery life by running on battery power until the battery is fully discharged and the system crashes.

1.3.4 Qualify Battery Life with Performance

In addition to the battery life (in minutes) MobileMark Office productivity results always report response time.

This makes it easy to tell the difference between a battery life result for a low performance system and a similar result for a high performance system that employs superior power management.

There is no performance component reported for the other workloads, however, as the user experience for those workloads is relatively insensitive to performance.

1.3.5 Constant Work/Time

The MobileMark Office productivity workload was calibrated to a minimal machine that completed one workload iteration in about 90 minutes. If a faster machine completes the workload iteration in less time, the system idles until the next activity cycle starts at 90-minutes.

2 Measurement Methods

Here we take a closer look at the methods available to observe and measure battery life in a Linux context.

2.1 Using an AC Watt Meter

Consumer-grade Watt Meters with a resolution of 0.1Watt and 1-second sampling rate are available for about 100 U.S. Dollars.[3] While intended to tell you the cost of operating your old refrigerator, they can just as easily tell you the A/C draw for a computer.

It is important to avoid the load of battery charging from this scenario by measuring. This can be done by measuring only when the battery is fully charged, or for laptops that allow

it, running on A/C with the battery physically removed.

You'll be able to see the difference between such steady-state operations as LCD on vs. off, LCD brightness, C-states and P-states. However, it will be very difficult to observe transient behavior with the low sampling rate.

Unfortunately, the A/C power will include the loss in the AC-to-DC power supply "brick." While an "External Power Adapter" sporting an Energy Star logo[4] rated at 20 Watts or greater will be more than 75% efficient, others will not meet that criteria and that can significantly distort your measurement results.

So while this method is useful for some types of comparisons, it isn't ideal for predicting battery life. This is because most laptops behave differently when running on DC battery vs. running on AC. For example, it is extremely common for laptops to enable deep C-states only on DC power and to disable them on AC power.

2.2 Using a DC Watt Meter on the DC converter

It is possible to modify the power adapter by inserting a precise high-wattage low-ohm resistor in series on the DC rail and measuring the voltage drop over this resistor to calculate the current, and thus Watts.

This measurement is on the DC side of the converter, and thus avoids the inaccuracy from AC-DC conversion above. But this method suffers the same basic flaw as the AC meter method above, the laptop is running in AC mode, and that is simply different from DC mode.

---

[3]Watt's Up Pro: https://www.doubleed.com

[4]http://www.energystar.gov

2.3 Replacing the Battery with a DC power supply

The next most interesting method to measure battery consumption on a laptop is to pull apart the battery and connect it to a lab-bench DC power supply.

This addresses the issue of the laptop running in DC mode. However, few reading this paper will have the means to set up this supply, or the willingness to destroy their laptop battery.

However, for those with access this type of test setup, including a high-speed data logger; DC consumption rates can be had in real-time, with never a wait for battery charging.

Further, it is possible that system designers may choose to make the system run differently depending on battery capacity. For example, high-power P-states may be disabled when on low battery power—but these enhancements would be disabled when running on a DC power supply that emulates a fully charged battery.

2.4 Using a DC Watt Meter on an instrumented battery

Finally, it is possible to instrument the output of the battery itself. Like the DC power supply method above, this avoids the issues with the AC wattmeter and the instrumented power converter method in that the system is really running on DC. Further, this allows the system to adapt as the battery drains, just as it would in real use. But again, most people who want to measure power have neither a data logger, nor a soldering iron available.

2.5 Using Built-in Battery Instrumentation

Almost all laptops come with built in battery instrumentation where the OS read capacity, calculate drain and charge rates, and receive capacity alarms.

On Linux, `/proc/acpi/battery/*/info` and `state` will tell you about your battery and its current state, including drain rate.

Sometimes the battery drain data will give a good idea of average power consumption, but often times this data is mis-leading.

One way to find out if your drain rate is accurate is to plot the battery capacity from fully charged until depleted. If the system is running a constant workload, such as idle, then the instrumentation should report full capacity equal to the design capacity of the battery at the start, and it should report 0 capacity just as the lights go out—and it should report a straight line in between. In practice, only new properly conditioned batteries do this. Old batteries and batteries that have not been conditioned tend to supply very poor capacity data.

Figure 1 shows a system with an old $(4.4AH * 10.4V) = 47.520\,Wh$ battery. After fully charging the battery, the instrumentation at the start of the 1st run indicates that the battery capacity of under 27.000 Wh. If the battery threshold warning was enabled for that run, the system would have shut down well before 5,000 seconds—even though the battery actually lasted past 7,000 seconds.

The 1st run was effectively conditioning the battery. The 2nd run reported a fully charged capacity of nearly 33.000 Wh. The actual battery life was only slightly longer than the initial conditioning run, but in this case the reported capacity was closer to the truth. The
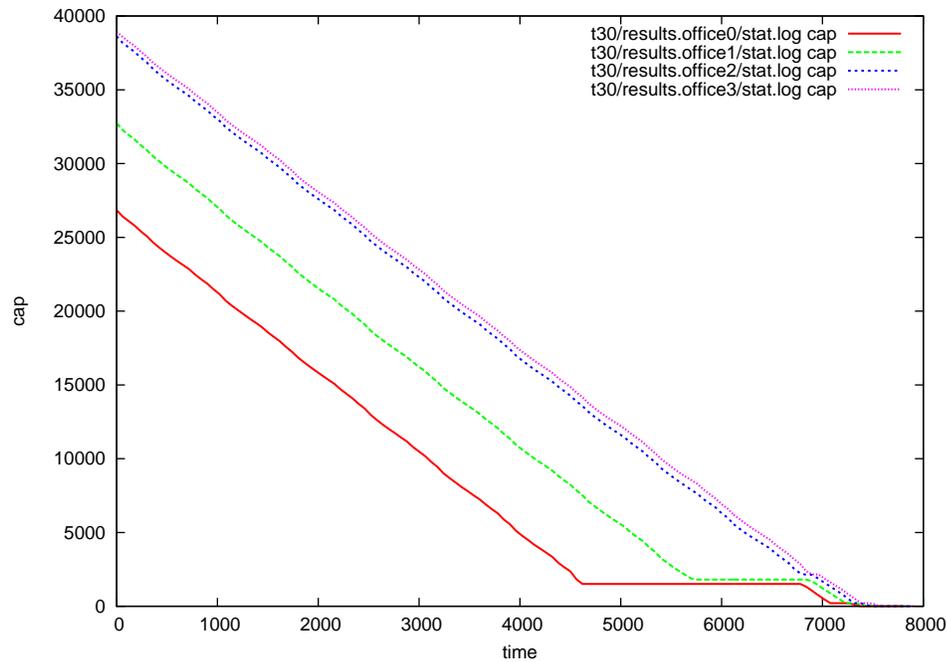
Figure 1: System A under-reports capacity until conditioned

3rd run started above 38.000 Wh and was linear from there until the battery died after 7,000 seconds. The 4th run showed only marginally more truthful results. Note that a 10% battery warning at 4,752 would actually be useful to a real user after the battery has been conditioned.

Note also that the slope of all 4 lines is the same. In this case, the rate of discharge shown by the instrumentation appears accurate, even for the initial run.

The battery life may not be longer than the slope suggests, it may be shorter. Figure2 shows system B suddenly losing power near the end of its conditioning run. However, the 2nd (and subsequent) runs were quite well behaved.

Figure 3 shows system C with a drop-off that is sure to fool the user's low battery trip points. In this case the initial reported capacity does not change, staying at about 6800 of 71.000 Wh (95%). However, the first run drops off a cliff at about 11,000 seconds. The second and third

runs drop at about 13,500. But subsequent runs all drop at about 12,000 seconds. So conditioning the battery didn't make this one behave any better.

Finally, Figure 4 shows system D reporting initial capacity equal to 100% of its 47.950 Wh design capacity. But upon use, this capacity drops almost immediately to about 37.500 Wh. Even after being conditioned 5 times, the battery followed the same pattern. So either the initial capacity was correct and the drain rate is wrong, or initial capacity is incorrect and the drain rate is correct. Note that this behavior went away when a new battery was used. A new battery reported 100% initial capacity, and 0% final capacity, connected by a straight line.

In summary, the only reliable battery life measurement is a wall clock measurement from full charge until the battery is depleted. Depleted here means ignoring any capacity warnings and running until the lights go out.
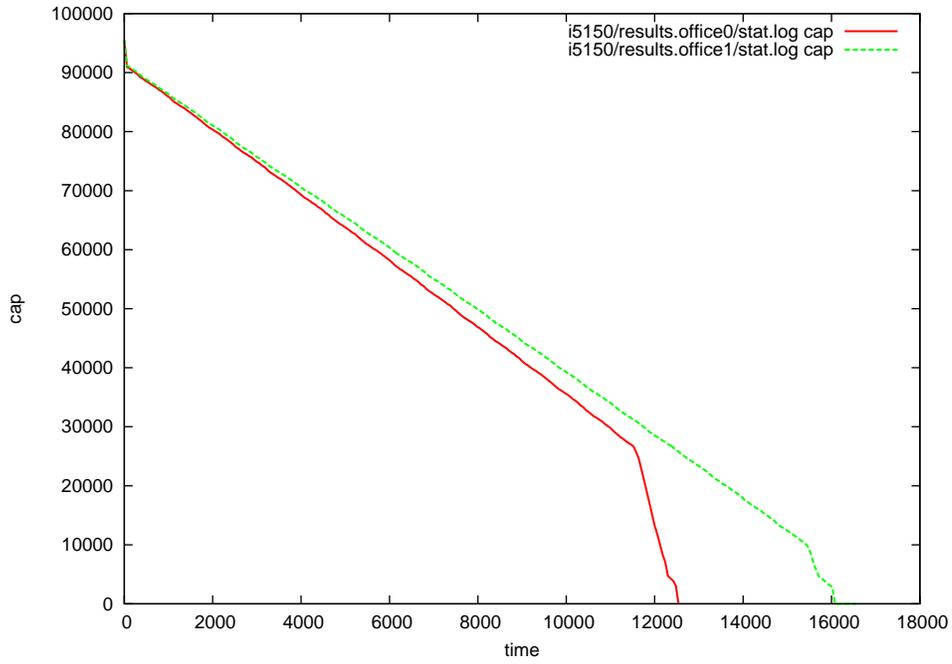
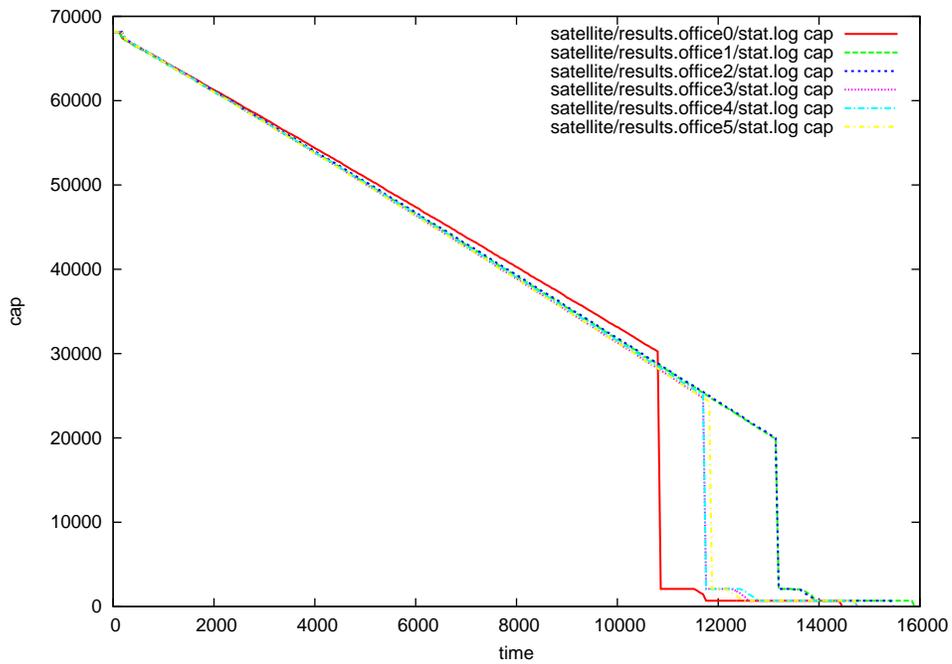Figure 2: System B over-reports capacity until conditioned

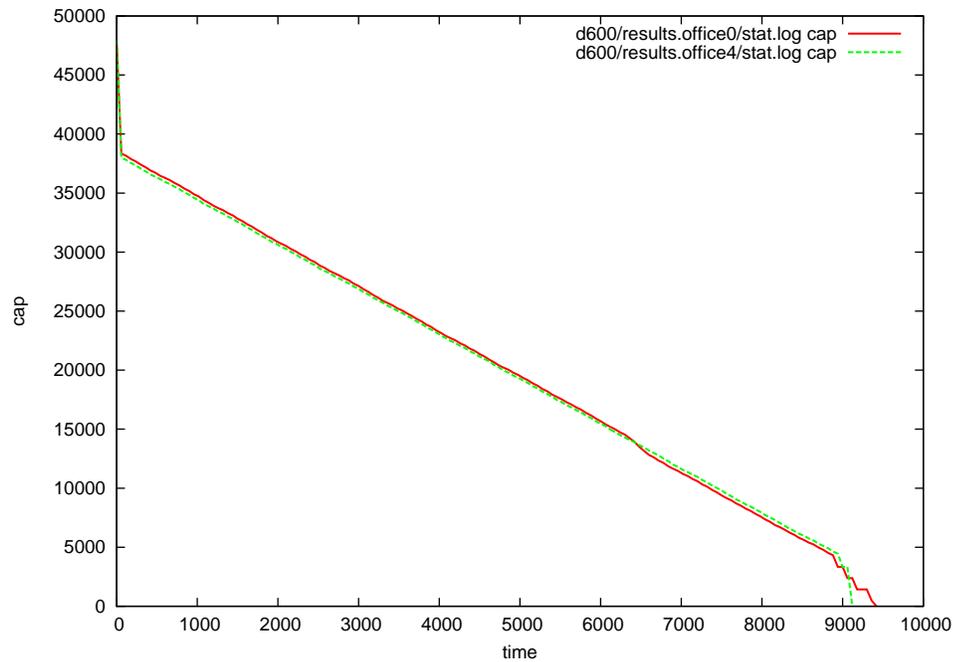Figure 3: System C over-reports final capacity, conditioning does not help

Figure 4: System D over-reports initial capacity, conditioning does not help

3 Linux Battery Life Toolkit

The Linux Battery Life Toolkit (bltk) consists of a test framework and six example workloads. There are common test techniques that should be followed to assure repeatable results no matter what the workload.

3.1 Toolkit Framework

The toolkit framework is responsible for launching the workload, collecting statistics during the run, and summarizing the results after a test completes.

The framework can launch any arbitrary workload, but currently has knowledge of 6 example workloads: Idle, Reader, Office, DVD Player, SW Developer, and 3D-Gamer.

3.2 Idle Workload

The idle workload simply executes the framework without invoking any programs. Statistics are collected the same way as for the other workloads.

3.3 Web Reader Workload

The web reader workload opens an HTML-formatted version of *War and Peace* by Leo Tolstoy[5] in Firefox® and then sends "next page" keyboard events to browser every two minutes, simulating interaction with the human reader.

---

[5] We followed the lead of BAPCo's MobileMark here on selection of reading material.

3.4 Open Office Workload

Open Office rev 1.1.4 was chosen for this toolkit because it is stable and freely available. It is intended to be automatically installed by the toolkit to avoid results corruption due to local settings and version differences.

3.4.1 Open Office Activities

Currently 3 applications from OpenOffice suite are used for the Office workload, `oowriter`, `oocalc` and `oodraw`. The set of common operations is applied to these applications to simulate activities, typical for office application users.

Using `oowriter`, the following operations are performed:

- text typing

- text pattern replacement

- file saving

Using `oocalc`, the following operations are performed:

- creating spreadsheet;

- editing cells values;

- assigning math expression to the cell;

- expanding math expression over a set of cells;

- assigning set of cells to the math expression;

- file saving;

Using `oodraw`, the following operations are performed:

- duplicating image;

- moving image over the document;

- typing text over the image;

- inserting spreadsheet;

- file saving.

3.4.2 Open Office User Input

User input consists of actions and delays. Actions are represented by the key strokes sent to the application window though the X server. This approach makes the application perform the same routines as it does during interaction with the real user.[6] Delays are inserted between actions to represent a real user.

The Office workload scenario is not hard-coded, but is scripted using the capabilities shown in Appendix A.

3.4.3 Open Office Performance Scores

A single iteration of the office workload scenario completes in 720 seconds. When a faster machine completes the workload in less than 720 seconds, it is idle until the next iteration starts.

$$720 seconds = Workload\_time + Idle$$

Workload time consists of Active_time—the time it takes for the system to start applications and respond to user commands—plus the delays that the workload inserts to model user type-time and think-time.

---

[6]The physical input device, such as keyboard and mouse are not used here.

$$Workload\_time = Active\_time + Delay\_time$$

So the performance metric for each workload scenario iteration is Active_time, which is calculated by measuring Workload_time and simply and subtracting the known Delay_time.

The reported performance score is the average Active_time over all loop iterations, normalized to a reference Active_time so that bigger numbers represent better performance:

$$Performance\_score = 100 * Active\_reference / Average\_Active\_measured$$

3.5 DVD Movie Playback Workload

`mplayer` is invoked to play a DVD movie until the battery dies. Note that `mplayer` does not report frame rate to the toolkit framework. For battery life comparisons, equal work/time must be maintained, so that it is assumed, but not verified in these tools that modern systems can play DVD movies at equal frame rates.

3.6 Software Developer Workload

The software developer workload mimics a Linux ACPI kernel developer: it invokes `vi` to insert a comment string into one of the Linux ACPI header files and then invokes `make -j N` on a Linux kernel source tree, where N is chosen to be three times the number of processors in a system. This cycle is extended out to 12 minutes with idle time to more closely model constant work/time on different systems.

The Active_time for the developer workload is the time required for the `make` command to complete, and it is normalized into a performance score the same was as for the Office workload.

$$Performance\_score = 100 * Active\_reference / Average\_Active\_measured.$$

3.7 3D Gamer Workload

A 3D gamer workload puts an entirely different workload on a laptop, one that is generally more power-hungry than all the workloads above.

However, we found that 3D video support is not universally deployed or enabled in Linux, nor are there a lot of selections of games that are simultaneously freely available, run on a broad range of platforms, and include a demo-mode that outputs performance.

`glxgears` satisfies the criteria for being freely available, universally supported, and it reports performance; however, that performance is not likely to closely correlate to what a real 3D game would see. So we are not satisfied that we have a satisfactory 3D-Gamer metric yet.

In the case of a 3D game workload, a reasonable performance metric to qualify battery life would be based on frames/second.

$$3D\_Performance\_Score = FPS\_measured / FPS\_reference$$

4 Example Measurement Results

This section includes example battery life measurements to show what a typical user can do on their system without the aid of any special instrumentation.

Unless otherwise specified, the Dell Inspiron™ 6400 shown in Table 1 was used as the example system.

Note that this system is a somewhat arbitrary reference. It has a larger and brighter screen

System	Dell Inspiron 6400
Battery	53 Wh
Processor	Intel Core Duo T2500, 2GHz 2MB cache, 667MHz bus
LCD	15.4" WXGA, min bright
Memory	1GB DDR2, 2DIMM, 533MHz
Distribution	Novell SuSE 10.1 BETA
GUI	KDE
Linux	2.6.16 or later
HZ	250
cpufreq	ondemand governor
Battery Alerts	ignored
Screen Saver	disabled
DPMS	disabled
Wired net	disabled
Wireless	disabled

Table 1: Nominal System Under Test

than many available on the market. It arrived with a 53 Wh 6-cell battery, but is also available with an 85 Wh 9-cell battery, which would increase the absolute battery life results by over 50%. But the comparisons here are generally of this system to itself, so these system-specific parameters are equal on both sides.

4.1 Idle Workload

4.1.1 Idle: Linux vs. Windows

Comparing Linux[7] with Windows[8] on the same hardware tells us how Linux measures up to high-volume expectations.

This baseline comparison is done with pure-idle workload. While trivial, this "workload" is also crucial, because a difference in idle power consumption will have an effect on virtually all other workloads.

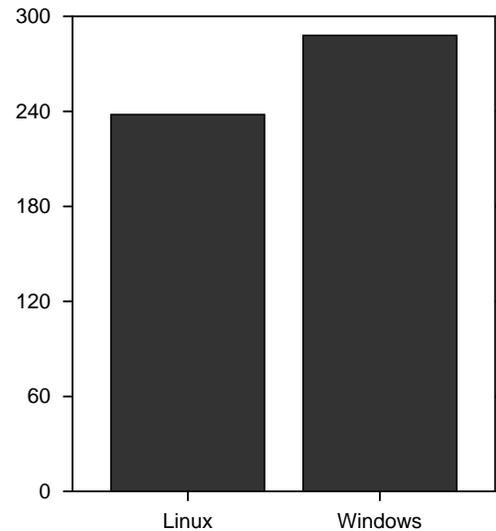[7]Linux-2.6.16+ as delivered with Novell SuSE 10.1 BETA

[8]Windows® XP SP2

Figure 5: Idle: Linux vs. Windows

Here the i6400 lasts 288 minutes on Windows, but only 238 minutes on Linux, a 50 minute deficit. One can view this as a percentage, eg. Linux has $238/288 = 83\%$ of the idle battery life as compared to Windows.

One can also estimate the average power using the fixed 53 Wh battery capacity. $(53Wh * 60min/Hr)/288min = 11.0W$ for Windows. $(53Wh * 60min/Hr)/238min = 13.4W$ for Linux. So here Linux is at a 2.4W deficit compared to Windows in idle.

4.1.2 Idle: The real cost of the LCD

While the i6400 has a larger than average display, the importance of LCD power can not be over-stated—even for systems with smaller displays.

The traditional screen saver that draws pretty pictures on an idle screen is exactly the opposite of what you want for long battery life. The reason isn't because the display takes more power, the reason is because it takes the proces-
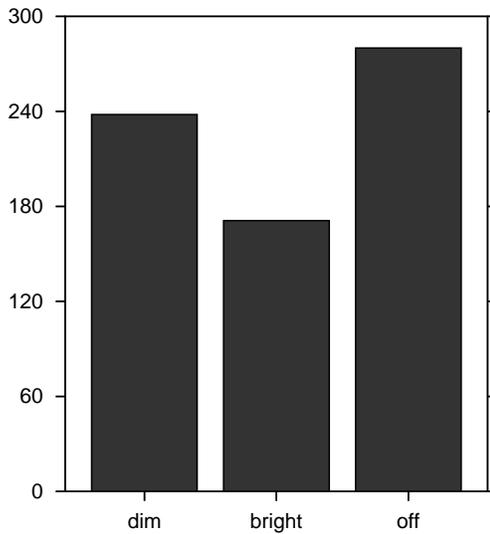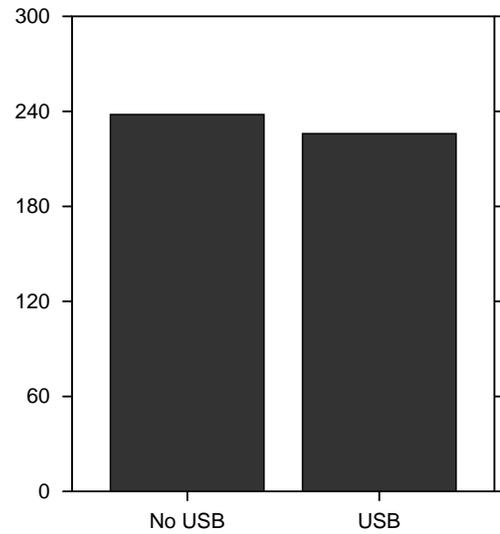
Figure 6: Idle: Effect of LCD

Figure 7: Idle: Effect of USB

sor out of the deepest available C-state when there is nothing "useful" to do.

The CPU can be removed from that equation by switching to a screen saver that does not run any programs. Many select the "blank" screen saver on the assumption that it saves power— but it does not. A Black LCD actually saves no power vs. a white LCD. This is because the black LCD has the backlight on just as bright as the white LCD, but it is actually using fractionally more energy to block that light with every pixel.

So the way to save LCD power is to dim the back-light so it is no brighter than necessary to read the screen; and or turn it off completely when you are not reading at the screen. Note that an LCD that is *off* should appear *black* in a darkened room. If it is glowing, then the pixels are simply fighting to obscure a backlight that is still on. A screen saver that runs no programs and has DPMS (Display Power Management Signaling) enabled to turn off the display is hugely important to battery life.

On the example system, the 238-minute "dim"

idle time drops to 171 for maximum LCD brightness, and increases to 280 minutes for LCD off. Expressed as Watts, dim is 13.4W, bright is 18.6W, and off is 11.4W. So this particular LCD consumes between 2.0 and 7.2W. Your mileage will vary.

Note that because of its large demands system power, analysis of the power consumption of the other system components is generally most practical when the LCD is off.

4.1.3 Idle: The real cost of USB

The i6400 has no integrated USB devices. So if you execute `lsusb`, you'll see nothing until you plug in an external device.

If an (unused) USB 1.0 mouse is connected to the system, battery life drops 12 minutes to 226 from 238. This corresponds to $(14.1 - 13.4) = 0.7W$.
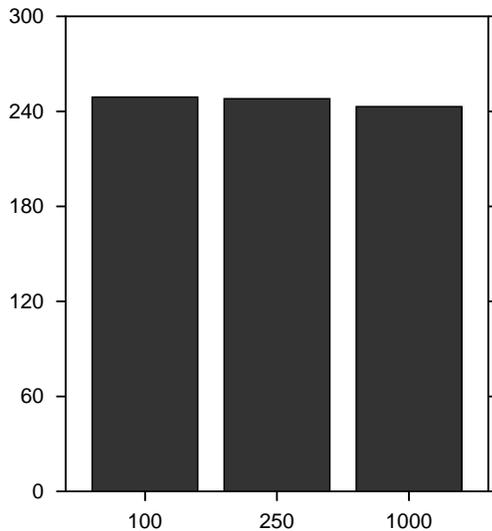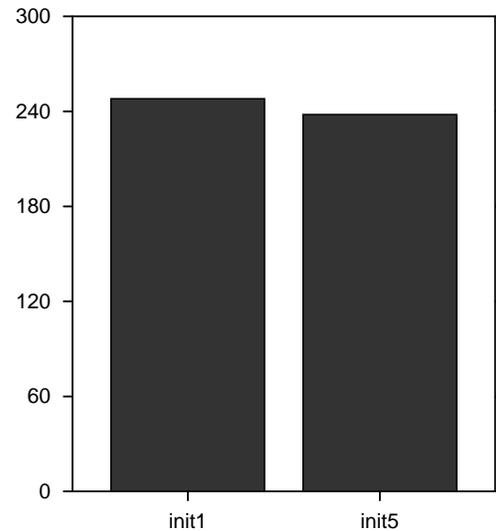
Figure 8: Idle: Effect of HZ

Figure 9: Idle: init1 vs. init5

4.1.4   Idle: Selecting HZ

In Linux-2.4, the periodic system timer tick ran at 100 HZ. Linux-2.6 started life running at 1000 HZ. Linux-2.6.13 added CONFIG_HZ, with selections of 100, 1000, and a compromise default of 250 HZ.

Figure 8 shows that the selection of HZ has a very small effect on the i6400, though others have reported larger differences on other systems. Note that since this difference was small, this comparison was made in single-user mode.

4.1.5   Idle: init1 vs. init5

The Linux vs. Windows measurement above was in multi-user GUI mode—though the network was disabled. One question often asked if the GUI (KDE, in this example) and other standard daemons have a significant effect on Linux battery life.

In this example, the answer is yes, but not much. Multi-user battery life is 238 min-

utes, and Single-user battery life is 10 minutes longer at 248—only a 4% difference. Expressed as Watts, $13.4 - 12.8 = 0.6W$ to run in multi-user GUI mode.

However, init5 battery consumption may depend greatly on how the administrator configures the system.

4.1.6   Idle: 7200 vs. 5400 RPM Disk Drives

The i6400 arrived with a 5400 RPM 40GB Fujitsu MHT2040BH SATA drive. Upgrading that drive to a 7200 RPM 60GB Hitachi HTS721060G9SA00 SATA drive reduced single-user[9] idle battery life by 16 minutes, to 232 from 248 (6%). This corresponds to an average power difference of 0.89W. The specifications for the drives show the Hitachi consuming about 0.1W more in idle and standby, and the same for read/write. So it is not im-

---

[9]init1 idle is used as the baseline here because the difference being measured is small, and to minimize the risk that the two different drives are configured differently.

mediately clear why Linux loses an additional 0.79W here.

4.1.7   Idle: Single Core vs. Idle Dual Core

Disabling one of the cores by booting with `maxcpus=1` has no measurable effect on idle battery life. This is because the BIOS leaves the cores in the deepest available C-state. When Linux neglects to start the second core, it behaves almost exactly as if Linux had started that core and entered the deepest available C-state on it.

Note that taking a processor off-line at run-time in Linux does not currently put that processor into the deepest available C-state. There is a bug[10] where offline processors instead enter C1. So taking a processor offline at run-time can actually result in worse battery life than if you leave it alone and let Linux go idle automatically.

4.1.8   The case against Processor Throttling (T-States)

Processor Throttling States (T-states) are available to the administrator under `/proc/acpi/processor/*/throttling` to modulate the clock supplied to the processors.

Most systems support 8 throttling states to decrease the processor frequency in steps of 12.5%. Throttling the processor frequency is independent of P-state frequency changes, so the two are combined. For the example, Table 2 shows the potential effect of throttling when the example system is in P0 or P3.

---

[10] `http://bugzilla.kernel.org/show_bug.cgi?id=5471`

State	P0 MHz	P3 MHz
T0	2000	1000
T1	1750	875
T2	1500	750
T3	1250	625
T4	1000	500
T5	750	375
T6	500	250
T7	250	125

Table 2: Throttling States for the Intel® Core™ Duo T2500

Throttling has an effect on processor frequency only when the system is in the C0 state executing instructions. In the idle loop, Linux is in the Cx state (*x: x != 0*) where no instructions are executed and throttling has no effect, as the clock is already stopped.

Indeed, T-states have been shown to have a net *negative* impact on battery life on some systems, as they can interfere with the mechanisms to efficiently enter deep C-states.

On the example system, throttling the idle system down to the T7, the slowest speed, had a net *negative* impact on idle battery life of 4 minutes.

Throttling is used by Linux for passive cooling mode and for thermal emergencies. It is not intended for the administrator to use throttling to maximize performance/power or extend battery life. That is what cpufreq processor performance states are for. So the next time you are exploring the configuration menus of the powersavd GUI, do NOT check the box that enables processor clock throttling. It is a bug that the administrator is given the opportunity to make that mistake.
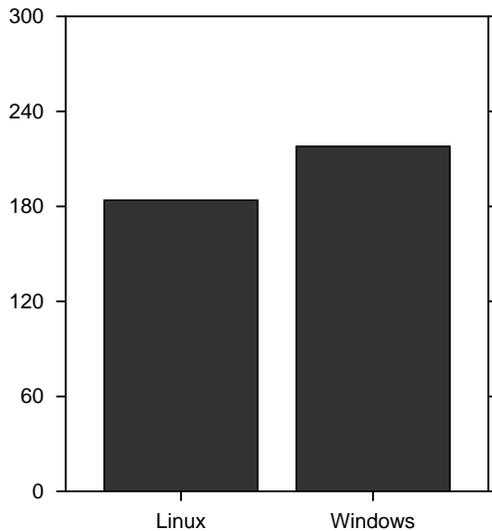
Figure 10: DVD: Linux vs. Windows

Here the i6400 plays a DVD on Linux for 184 minutes (3h4m). The i6400 plays the same DVD on Windows for 218 minutes (3h38m). This 34 minute deficit puts Linux at about 84% of Windows. In terms of Watts, Linux is at a $(17.3 - 14.6) = 2.7W$ deficit compared to Windows on DVD movie playback.

4.3.2 DVD Movie Single vs. Dual Core

DVD playback was measured with 1 CPU available vs. 2 CPUS, and there was zero impact on battery life.

4.3.3 DVD Movie: Throttling is not helpful

DVD playback was measured at T4 (50% throttling) and there was a net *negative* impact of 9 minutes on battery life. Again, throttling should be reserved for thermal management, and is almost never an appropriate tool where efficient performance/power is the goal.

4.2 Reader Workload equals Init 5 Idle

Adding the Reader workload to init5 idle results in exactly the same battery life—238 minutes. The bar chart is left as an exercise for the reader.

4.3 DVD Movie Workload

4.3.1 DVD Movie on Linux vs. Windows

The DVD movie playback workload is also attractive for comparing Linux and Windows. This constant work/time workload leaves little room for disagreement about what the operating environment is supplying to the user. DVD movie playback is also a realistic workload, people really do sit down and watch DVDs on battery power.

However, different DVD player software is used in each operating environment. The Windows solution uses WinDVD®, and the Linux measurement uses mplayer.

4.4 Office Workload Battery Life and Performance

The Office workload battery life and performance are shown in Figure 11 and Figure 12, respectively. The example system lasted 232 minutes with `maxcpus=1` and a 5400 RPM drive, achieving a performance rating of 94 (UP5K in Figures 11 and 12). Enabling the second core cost 6 minutes (–3%) of battery life, but increased performance by 89% to to 178, (MP5K in Figures 11 and 12).

Upgrading the 5400 RPM disk drive to the 7200 RPM model had an 18 minute (8%) impact on the UP battery life, and an 12 minute (5%) impact on MP battery life. But the 7200 RPM drive had negligible performance benefit on this
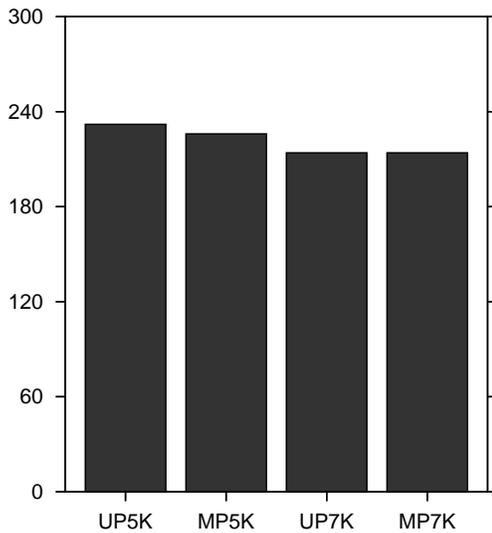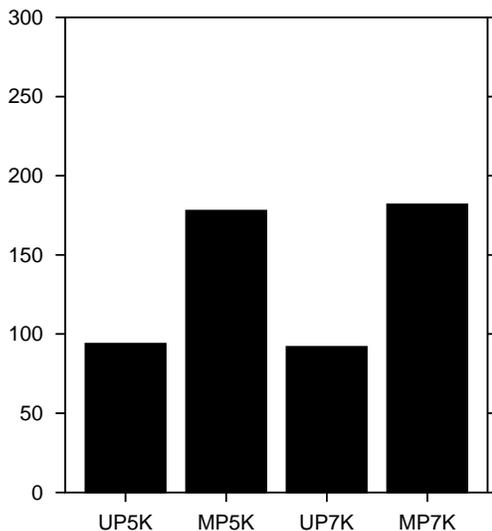
Figure 11: Office Battery Life
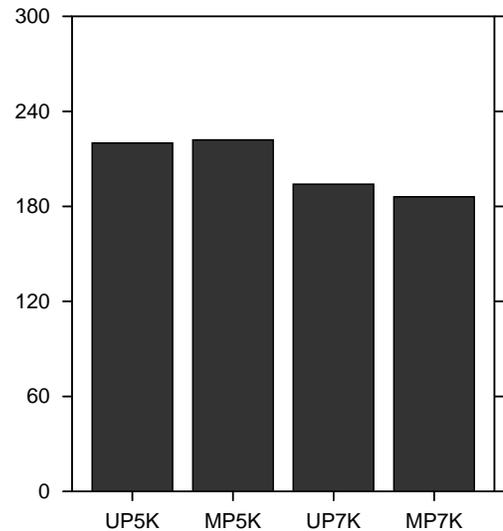

Figure 12: Office Performance

Figure 13: Developer Battery Life

workload. (UP7K and MP7K in Figures 11 and 12).

Note that the size of memory compared to the working set of the Office workload impact how much the disk is accessed. Were memory to be smaller or the workload modified to access the disk more, the faster drive would undoubtedly have a measurable benefit.

In summary, the second core has a significant performance benefit, with minimal battery cost on this workload. However, upgrading from a 5400RPM to 72000 RPM drive does not show a significant performance benefit on this workload as it is currently implemented.

4.5 Developer Workload Battery Life and Performance

The Developer workload battery life and performance are shown in Figure 13 and Figure 14, respectively.

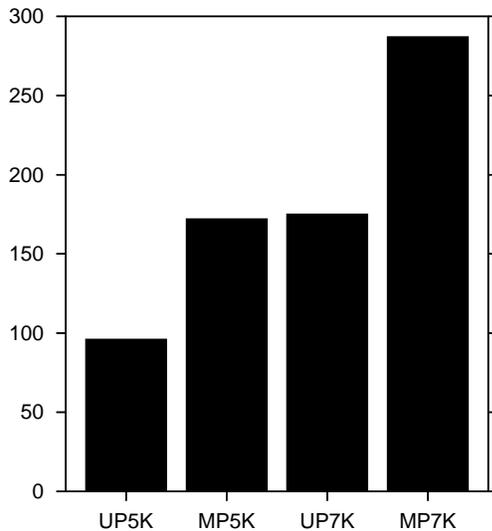Here the `maxcpus=1` 5400 RPM baseline scores 220 minutes with performance of 96.

Figure 14: Developer Performance

Enabling the second core had a net positive impact on battery life of 2 minutes, and increased performance to 172 (+79%). Starting from the same baseline, upgrading to the 7200 RPM drive from the 5400 RPM drive dropped battery life 26 minutes to 194 from 220 (–12%), but increased performance to 175 (+82%). Simultaneously enabling the second core and upgrading the drive reduced battery life 34 minutes to 186 from 220 (15%), but increased performance to 287 (+198%).

Clearly developers using this class of machine should always have both cores enabled and should be using 7200 RPM drives.

5 Future Work

5.1 Enhancing the Tools

The current version of the tools, 1.0.4, could use some improvements.

- Concurrent Office applications. The current scripts start an application, use it, and then close it. Real users tend to have multiple applications open at once. It is unclear if this will have any significant effect on battery life, but it would be better eye candy.

- Add sanity checking that the system is properly configured before starting a measurement.

5.2 More Comparisons to make

The example measurements in this paper suggest even more measurements.

- Effect of run-time device power states.

- Comparison of default policies of different Linux distributors.

- Benefits of the laptop patch?

- USB 2.0 memory stick cost

- Gbit LAN linked vs unplugged

- WLAN seeking

- WLAN associated

- Bluetooth

- KDE vs. Gnome GUI

- LCD: brightness vs power consumption is there an optimal brightness/power setting?

- Power consumption while suspended to RAM vs. power consumption to reboot. What is break-even for length of time suspended vs halt, off, boot?

- Suspend to Disk and wakeup vs. staying idle

- Suspend to RAM and wakeup vs staying idle

6 Conclusion

The authors hope that the tools and techniques shown here will help the Linux community effectively analyze system power, understand laptop battery life, and improve Linux power management.

Appendix A: Scripting Commands

Keystrokes, keystroke conditions (like <Alt>, <Ctrl>, <Shift>, etc.) and delays are scripted in a scenario file along with other actions (`run command`, `wait command`, `select window`, `send signal`, etc.). The scenario file is passed to the workload script execution program, strings are parsed and appropriate actions are executed.

The scenario script is linear, no procedure defining is (currently) supported. Each string consists of 5 white space separated fields and begins with command name followed by 4 arguments (State, Count, Delay, String). For each particular command arguments could have different meanings or be meaningless, though all 4 arguments should present. The following commands are implemented:

Commands to generate user input

`DELAY 0 0 Delay 0`
   Suspends execution for 'Delay' msecs;

`PRESSKEY State Count Delay String`
   Send *Count* `State + String` keystrokes with *Delay* msec intervals between them to the window in focus, i.e. command `PRESSKEY S 2 500 Down` would generate two `<Shift> + <Down>` keystrokes with 1/2 second intervals. The state values are:

S for Shift,

A for Alt,

C for Ctrl.

Some keys should be presented as their respective names: Up, Down, Left, Right, Return, Tab, ESC.

`RELEASEKEY 0 0 0 String`
   Similar to `PRESSKEY` command, except the *Release* event being sent. It could be useful since some menu buttons react on key release, i.e. the pair of `PRESSKEY 0 0 <Delay> Return` and `RELEASEKEY 0 0 <Delay> Return` should be used in this case.

`TYPETEXT State 0 Delay String`
   Types text from *String* with *Delay* msecs interval between keystrokes. If *State* is `F` then the text from *String* file is typed instead of *String* itself.

`ENDSCEN 0 0 0 0`
   End of scenario. No strings beyond this one will be executed.

Commands to operate applications

`RUNCMD 0 0 0 String`
   Execute command *String*, exits on completion.

`WAITSTARTCMD 0 Count Delay String`
   Checks *Count* times with *Delay* msecs intervals if *String* command is started (total wait time is *Count * Delay* msecs).

`WAITFINISHCMD 0 Count Delay String`
   Checks *Count* times with *Delay* msecs intervals if *String* command is finished (total wait time is *Count * Delay* msecs).

Commands to interact with X windows

```
SETWINDOWID State 0 0 String
```
Makes window with X window ID located in 'String' object active. If *State* is F, then *String* is treated as a file; if E or O, as an environment variable;

```
SETWINDOW 0 0 0 String
```
Waits for window with *String* title to appear and makes it active.

```
FOCUSIN 0 0 0 0
```
Sets focus to current active window.

```
FOCUSOUT 0 0 0 0
```
Gets focus out of current active window.

```
ENDWINDOW 0 0 0 String
```
Waits for window with *String* title to disappear.

```
SYNCWINDOW 0 0 0 0
```
Tries to synchronize current active window.

To reach one particular window, SETWINDOW and FOCUSIN commands should be performed.

Commands to generate statistics

```
SENDWORKMSG 0 0 0 String
```
Generate 'WORK' statistics string in log file with the *String* comment.

```
SENDIDLEMSG 0 0 0 String
```
Generate 'IDLE' statistics string in log file with the *String* comment.

Note that the harness generates statistics regularly, so the above commands are intended to generate strings to mark the beginning and ending of the set of operations (e.g. 'hot-spot'), for which special measurements are required.

Debugging Commands

```
TRACEON 0 0 0 0
```
Enable debug prints;

```
TRACEOFF 0 0 0 0
```
Disable debug prints;

References

[ACPI] Hewlett-Packard, Intel, Microsoft, Phoenix, Toshiba *Advanced Configuration & Power Specification*, Revision 3.0a, December 30, 2005. `http://www.acpi.info`.

[Linux/ACPI] Linux/ACPI Project Home page, `http://acpi.sourceforge.net`.

[MM02] MobileMark® 2002, Business Applications Performance Corporation, `http://bapco.com`, June 4, 2002, Revision 1.0.

[MM05] MobileMark® 2005, Business Applications Performance Corporation, `http://bapco.com`, May 26, 2005, Revision 1.0.