DOI:10.1145/1378727.1378740

Smart power management is all about doing more with the resources we have.

BY MATTHEW GARRETT

Powering Down

POWER MANAGEMENT—FROM laptops to rooms full of servers—is a topic of interest to everyone. In the beginning there was the desktop computer. It ran at a fixed speed and consumed less power than the monitor it was plugged into. Where computers were portable, their sheer size and weight meant that you were more likely to be limited by physical strength than battery life. It was not a great time for power management.

Now consider the present. Laptops have increased in speed by more than 5,000 times. Battery capacity, sadly, has not. With hardware becoming increasingly mobile, however, users are demanding that battery life start matching the way they work. People want to work from cafés. Long-haul flights are now perceived as the ideal opportunity to finish a presentation. Two hours of battery life just isn't going to cut it; users are looking for upward of eight hours. What's drawing that power, and more importantly, how can we reduce it?

The processor is, perhaps, the most obvious target of power management. On a modern system the CPU is likely to be the single component consuming the most power. Switching the state of hundreds of millions of transistors takes power. Modern processors can generate well over 100 watts of heat. How can this be reduced?

At the most basic level the answer is simple: reduce the power taken to switch those transistors. Making them smaller is one mechanism, as is reducing the amount of power lost via leakage. There are limits, however, to how much power can be saved via fundamental improvements in semiconductor technology, and vendors are being forced to adopt increasingly high-tech solutions to maintain the rate of progress in this respect. We cannot rely on technological breakthroughs to be the silver bullet. We need to be smarter in how we use what we already have available.

The most reasonable approach to reducing the power consumption of processors is to realize that under normal usage patterns processors will not run at 100% utilization all the time. The first attempt to take advantage of this was in the Pentium era with the addition of idle power savings. The HLT (halt) instruction allowed operating systems to indicate that they had nothing to execute, letting the processor put itself into some form of power saving until the next interrupt arrived. Runtime processor power management had arrived.

The APM (Advanced Power Management) specification extended this functionality. Rather than simply halting the processor, the APM CPU idle call allowed the processor to be put into a lower power state where it continued to execute instructions. If system load reached a threshold level, the operating system could then send a CPU busy call to restore the processor to full operating speed.

This concept of runtime power management has been further enhanced in recent years. CPU clock and volt-

age scaling (such as Intel's SpeedStep) allows the clock rate of an idle processor to be reduced. Running more slowly provides greater tolerances for the processor, thus reducing the supply voltage. Reducing the clock speed provides a linear reduction in power usage; the simultaneous reduction of voltage allows the power consumption to be reduced quadratically.

As an orthogonal approach, modern processors implement dynamic power management by allowing the operating system to trigger entry into a range of low-power states when there is nothing to be executed. The most basic of these corresponds to the traditional behavior of the HLT instruction, but deeper sleep states allow the processor package to disable parts of itself. In the deepest states the processor can dissociate itself from the memory bus, disable much of the cache, and reach a state where it consumes a negligible quantity of power.

Moving between these low-power states takes time, with the deeper states taking longer to enter and leave. Since parts of the processor have been disabled, there's no way for code to be executed. The processor must be raised back to the full-power state before it can handle the next instruction.

As a result, deeper sleep states provide their greatest benefits only when the system will spend a significant period of time (20 milliseconds or more) truly idle. Most preemptive multitasking operating systems implement scheduling by allocating a time slice to running applications and forcibly switching to another task at the end of this time slice. Traditionally this time slice has been on the order of 10 milliseconds, meaning that an interrupt will be fired 100 times a second to enforce

scheduling. This interrupt is fired even if the system is idle, thus waking up the processor. A traditional fixed-tick operating system is therefore unable to obtain the greatest benefits from modern processor power savings.

A new model of scheduling was introduced in version 2.6.21 of the Linux kernel. Rather than having a static timer tick, the kernel looks for a list of tasks waiting to run. If the system is idle, it will look for the next expected timer expiry (such as an application that has requested to sleep for 200 milliseconds) and program the timer interrupt to fire at that point. The system will then sleep until either the next interrupt is fired or some external hardware interrupt occurs.

In the best-case scenario, this dynamic tick model allows the processor to be almost entirely shut down until the next time the user interacts with the computer. The real-world scenario is, unsurprisingly, worse. Many software programs set timers, and when these timers expire the processor must wake up to handle the interrupt.

These timers are often unnecessary. Some are simply polling loops that could be replaced by an interrupt-driven design. Others are straightforward design flaws—for example, an email client that checks local mailboxes for changes every 100 milliseconds even though the mail download interval is set to five minutes. Removing these loops can have a surprisingly large impact upon power consumption.

Sometimes, however, an application has no choice but to poll. In this case, it makes sense to consolidate as many loops as possible. Because of the latency required when switching into low-power states, a processor that wakes once a second and spends

four milliseconds executing code will actually draw less power than waking twice a second and spending two milliseconds executing code each time. The GLIB library used by GTK (GIMP toolkit) includes a helper function to make this easier. The `g_timeout_add_seconds()` function provides a time-out that will fire after a certain number of seconds, but only with second-level granularity. All time-outs scheduled to fire in a given second will be called at once, avoiding the need to add manual synchronization. Even better, this is synchronized across all GLIB-using applications. Using this in all applications on a system can significantly reduce the number of wake-ups per second, providing a drop in power consumption.

Of course, these issues are not limited to user-space code. Device drivers may have the same issues, and for many of the same reasons. The Linux kernel includes a function similar to GLIB's, allowing unavoidable polling to be synchronized. In some cases, however, the issues can be subtler. For example, hardware may continue sending events even when no information is present, unless it's told to stop. If the driver is well written, it will handle the case of empty data packets cleanly; thus, the absence of any support for quiescing the hardware may go unnoticed. Driver authors should disable hardware when it's not doing anything and avoid polling it if nobody is going to use the information.

The combination of processor-frequency scaling and idle-power states provides a somewhat surprising result. On almost all modern hardware, if there is code to run, then it is more power efficient to run the processor at full speed. This "race to idle" concept

How to Reduce Power Consumption

► **Hardware designers:** Make sure that your hardware never requires polling. Send interrupts on all status changes. Provide fine-grained control of logical subunits of the hardware in order to let the operating system power down as much as possible based on the functional

constraints imposed by the user.

► **Device driver authors:** Make sure that your driver doesn't poll. If nothing is listening to the driver (for example, if nothing is using a Webcam) then quiesce the hardware. Provide a rich interface to the device, allowing user space to

make it clear which functionality it requires. Leave the device in the lowest power state that satisfies these criteria.

► **Kernel developers:** Remove any dependency on a fixed clock. If you know that no tasks are waiting to run, don't wake up hun-

dreds of times a second to check the run queue.

► **Application authors:** Don't poll. Only run timer loops when absolutely necessary and make sure they're synchronized. Don't keep hardware open if you don't need it.

stems from the power consumption of an idle processor being much lower than an active processor, even if running at a lower speed. The overall power consumption will be less if the processor spends a short time at full speed and then falls back to idle, rather than spending twice as long being active at a lower frequency.

Paying the Price for Graphics

Although the processor is the major component in determining power consumption, other parts of the platform also contribute. LCD panels in laptops are perhaps the most obvious secondary source. Modern hardware is moving from traditional cathode tube backlights to LED lighting, saving a small but significant quantity of power while providing the same level of illumination. Subtler techniques are beginning to appear in recent graphics chipsets. Intel has introduced technology to monitor the color distribution across the screen, which allows the backlight to be dimmed when the majority of the screen is dark. The intensity of brighter colors can be increased in order to compensate, resulting in power savings with little user-visible degradation.

Compressing the contents of the framebuffer can result in an additional reduction of power draw. A simple run-length encoding is often enough to achieve a significant reduction in image size, which means that the video hardware can reduce the amount of data that has to be read from the framebuffer memory and transferred to the screen. This simple optimization can save about 0.5 watts. An intriguing outcome of this is that desktop wallpaper design may influence system power consumption.

Hard drives have long been one of the more obvious aspects of power management. Even with all the advances made in recent years, hard-drive technology still requires an object to spin at high speed. In the absence of perpetual motion, this inevitably consumes power. For more than a decade operating systems have included support for spinning hard drives down after periods of inactivity. This indeed reduces power consumption, but it carries other costs. One is that disks are rated for only a certain number of

There are limits to how much power can be saved via fundamental improvements in semiconductor technology, and vendors are being forced to adopt high-tech solutions to maintain the rate of progress. We cannot rely on technological breakthroughs to be the silver bullet. We need to be smarter in how we use what we already have available.

spin-up/spin-down cycles and risk mechanical failure if this number is exceeded. A less-than-optimal approach to saving hard-drive power can therefore reduce hard-drive life expectancy.

Another cost is that spinning a disk back up uses more power than keeping a disk spinning. If access patterns are less than ideal, a drive will be spun up again shortly after being spun down. Not only will this result in higher average power consumption, but it will also lead to undesirable latency for applications trying to use the disk.

Making effective use of this form of power management requires a sensible approach to disk I/O. In the absence of cached data, a read from disk is inevitably going to result in spinning the disk back up. From a power management viewpoint, it makes more sense for an application to read all the data it is likely to need at startup and then avoid reading from disk in the future. Writes are more interesting. For most functionality, writes can be cached indefinitely. This avoids unnecessary disk spin-up, but increases the risk of data loss should the machine crash or run out of power.

The Linux kernel's so-called "laptop mode" offers a compromise approach in which writes are cached for a user-definable period of time if the disk is not spun up. They will be written out either when the user's time threshold is reached or when a read is forced to hit the disk. This reduces the average length of time that dirty data will remain cached, while also trying to avoid explicitly spinning the disk up to flush it.

Another way of avoiding write-outs is to reduce the number of writes in the first place. At the application level, it makes sense to collate writes as much as possible. Instead of writing data out piecemeal, applications should keep track of which information needs writing and then write it all whenever triggering any sort of write access.

At the operating-system level, writes can be reduced through careful consideration of the semantics of the file system. The metadata associated with a file on a Unix system includes a field that records the last time a file was accessed for any reason. This means that any read of a file will trigger a write to update the atime (time of last access),

even if the read is from the cache. One work-around is to disable atime updates entirely, but this breaks certain applications. A subtler work-around is to update the atime if the file was modified more recently than it was last read. This avoids breaking mail applications that depend upon the atime to determine whether mail has been read or not. This provides a dramatic reduction in write activity to the disk and makes it more likely that spinning the drive down will be worthwhile.

The drive is not the only part of the I/O system where savings are possible. The AHCI (Advanced Host Controller Interface) specification for serial ATA controllers includes link-level power management. In the absence of any pending commands, the link between the drive and the controller can be powered down, saving about 0.5 watts of power. The cost is a partial reduction in functionality—hotplug events will no longer be generated.

Network hardware provides a similar dilemma. If an Ethernet device is not in use, it makes sense to power it down. If the Ethernet PHY is powered down, however, there will be no interrupt generated if an Ethernet cable is plugged into the device. Power has been saved, but at the cost of some functionality. An almost identical problem occurs when detecting hotplugging of monitors.

This is perhaps the most unfortunate side of power management. In many cases, hardware consumes power because that power is required to provide functionality. Many users may not care about that loss of functionality, but disabling it by default causes problems for those who do. Although hardware support for power management has improved hugely over recent years, the biggest challenge facing operating-system developers may in fact be how to integrate that support in a way that doesn't frustrate or confuse users.

In an attempt to encourage power management, Intel has released a tool called PowerTOP (referring to the top command used to see which processes are consuming the most CPU on Unix systems). PowerTOP uses diagnostic information from the Linux kernel to determine which applications are triggering CPU wake-ups, allowing

developers to spot misbehaving applications and optimize their power consumption.

PowerTOP has already provided benefits for Linux desktop software. The 7.04 release of Ubuntu ran at about 400 wake-ups a second. Optimizations and bug fixes in response to issues raised by PowerTOP allowed this to be reduced to fewer than 100 on most systems, with figures under 30 achievable on systems with disabled wireless and Bluetooth.

PowerTOP also provides information about other aspects of a system that may be consuming power. The system is probed to determine whether it has any configuration that may impair power savings, such as disabled USB autosuspend or audio codec power management, and PowerTOP suggests ways to fix the problem. The information provided is probably excessively technical for the average user, but it allows vendors to ensure they are taking full advantage of available power management options.

Though developed and sponsored by Intel, the advice provided by PowerTOP is fairly general and appropriate to most Linux-based systems. Arguably, the most important way in which it has been successful is not in the functionality it provides in itself, but the increased awareness of the issues involved that it has generated in the open source community. It remains to be seen whether proprietary vendors will start providing similar functionality to users and developers, but many of the same issues apply and need to be solved in similar ways.

Power Management for the Masses

The One Laptop Per Child XO machine is an interesting case study in power management, perhaps sharing more in common with the embedded world than with traditional laptops. Its aggressive power management is designed to allow the platform to suspend even when the machine is in use, something made possible by the display controller's ability to scan out the framebuffer even when the CPU isn't running. The mesh networking capability of the hardware requires machines to continue routing packets even when suspended, and hence the wireless hardware has also been de-

signed to forward data without processor intervention. Price considerations have meant that the XO machine cannot depend on the latest battery technology; therefore, the machine must consume as little power as possible to keep the network functioning effectively.

This level of power management would be unthinkable in the traditional laptop world, where even the best implementations still take on the order of a second to return to a state where the user can interact with the system. A major focus has therefore been to reduce the time taken to bring the device back from suspend, making it practical to suspend the device when idle without impairing the user experience. This requires a high level of robustness, and much of the development work has been focused on ensuring that components resume in a reliable and consistent manner. A failure rate of one in every 10,000 suspend/resume cycles might be considered acceptable in the mainstream laptop world, but would impair the user experience on the XO.

As users demand longer battery life and become increasingly concerned about wasted energy, power management has become more important to vendors. A well-rounded power management strategy requires integration of hardware, firmware, and software, as well as careful consideration of how to obtain the maximum savings without making the user aware of any compromised functionality. Future years are likely to see tighter integration and a greater awareness of good power management practices, and all-day computing may soon become a practical reality.

Matthew Garrett (mjg59@srcf.ucam.org) is a software engineer at Red Hat, specializing in power management and mobile hardware support.

A previous version of this article appeared in *ACM Queue*, Nov/Dec 2007, Vol. 5, No. 7.