Software Controlled Power Management

Yung-Hsiang Lu, Tajana Šimunić, Giovanni De Micheli
Computer System Laboratory, Stanford University
{luyung, tajana, nanni}@stanford.edu

Abstract

Reducing power consumption is critical in many system designs. Dynamic power management is an effective approach to decrease power without significantly degrading performance. Power management decisions can be implemented in either hardware or software. A recent trend on personal computers is to use software to change hardware power states. This paper presents a software architecture that allows system designers to investigate power management algorithms in a systematic fashion through a template. The architecture exploits the Advanced Configuration and Power Interface (ACPI), a standard for hardware and software. We implement two algorithms for controlling the power states of a hard disk on a personal computer running Microsoft Windows. By measuring the current feeding the hard disk, we show that the algorithms can save up to 25% more energy than the Windows power manager. Our work has two major contributions: a template for software-controlled power management and experimental comparisons of management algorithms for a hard disk.

1. Introduction

Low power consumption is an important goal in designing modern electronic systems. Most previous studies of low-power techniques focused on either *hardware* (HW) [14] or *software* (SW) [12] to reduce power consumption. *Dynamic power management* (DPM) [2] is an effective approach to reduce power consumption without significantly degrading performance. DPM shuts down devices when they are not needed and wakes them up when necessary. Recently, Intel, Microsoft and Toshiba proposed the *Advanced Configuration and Power Interface* (ACPI) [1] to provide a uniform HW/SW interface for power management. ACPI allows hardware vendors, *operating system* (OS) designers, and device driver programmers to use the same interface. A hardware device complies with ACPI if it can properly respond to ACPI calls such as setting and querying power states. DPM requires specialized techniques and tools to use software to control hardware power states; therefore, it is a codesign problem, even though it differs from those typically discussed in literature [6].

In the recent past, research mainly focused on designing power management algorithms (also called policies). DPM algorithms can be divided into two major categories: predictive [7] [9] and stochastic [4] [13]. Typically, DPM algorithms are evaluated by simulations instead of measurements on real machines, due to the difficulty of setting up an environment that is flexible enough to test a variety of algorithms. Because of the complex interactions between hardware and software, only experimental evaluation of systems running real programs can validate the effectiveness of an algorithm. This paper addresses the implementation issue and evaluates algorithms on a computer running a commercial OS.

We designed and implemented a software architecture that allows system designers to perform power management through a template. The template is implemented as kernel-level *filter drivers* (FD) that attach to the device drivers from HW vendors. A *power manager* (PM) is a program that implements policies and sends the FD management decisions to change power states. The FD also reports device utilization for future management decisions. The FD has a generic interface; therefore, designers can evaluate the same algorithms on different HW devices or different algorithms on the same devices.

In this paper, we address the problem of power managing the hard disk in a personal computer. We set up a personal computer running Microsoft Windows and measured the power used by the hard disk. We implemented two management algorithms to compare with the standard power manager of Windows. The first algorithm is a time-out scheme that performs its own idleness detection by communicating with other programs. The other algorithm is an adaptive scheme that adjusts the time-out value by considering the bursty nature of disk accesses. These two algorithms outperform the PM in Windows by more than 20% in sample workloads.

DPM is applicable to embedded systems and it is of critical importance for mobile systems. The techniques presented in this paper can be applied to mobile systems with ACPI and Windows without modification. For other systems, our approach is still applicable but it requires different implementation.

2. Software Controlled Power Management

Software-controlled power management is a technique to reduce power consumption on computers [10]. In the past, the lack of standardization made interfaces like *advanced power management* (APM) [8] very specific to each system; as a result, it was hard to port and extend. In order to achieve larger power saving in a uniform fashion, Intel, Microsoft and Toshiba proposed ACPI as a standard for both hardware and software. Device drivers use the specification for device-specific power management; the power management API can also be exported from the OS to application programs. Figure 1 shows the ACPI interface [3] in which the PM resides in the operating system.

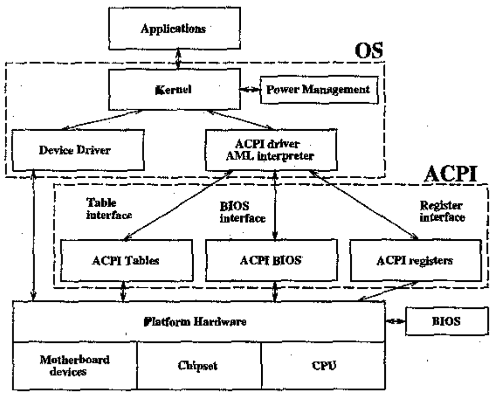

Figure 1: ACPI Interface and PC Platform

ACPI controls the power states of a whole system as well as the power states of each device. An ACPI-compliant system has five global states: `SystemStateS0`, the working state, and `SystemStateS1` to `SystemStateS4`, representing the sleeping states. An ACPI-compliant device has four states: `PowerDeviceD0 (D0)`, the working state and, `PowerDeviceD1 (D1)` to `PowerDeviceD3 (D3)`, representing the sleeping states. The sleeping states are differentiated by the power consumed and the time to wake up; the deeper sleeping state, the less power consumed and the longer time to wake up.

Similar to other I/O activities, *I/O request packets* (IRP) are used for power management commands. However, special IRP's are required to synchronize ACPI commands so that the transient current does not exceed the maximum capability of the power supply. A device driver has to appropriately respond to power IRP's issued from upper-level drivers or OS. Although ACPI has been used as an interface among hardware, device drivers, and operating systems, it is still a challenging task to design and evaluate software-controlled power management algorithms across devices in a uniform way. In order to facilitate this HW/SW design cycle, we propose a software architecture that exports power management capability outside the operating system through a template.

3. Application-Level Power Management

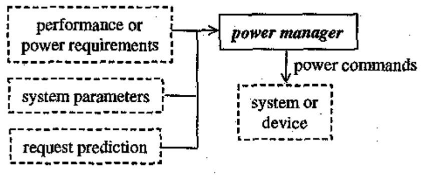

Figure 2: PM Controls Power States Based on System Parameters and Requests

Figure 2 shows that a PM makes decisions based on information about the system and requests. System parameters include the power at each state, transition energy and delay. The PM issues state-transition commands to a system or a device to meet the performance or power requirements (or both) by predicting future requests. For example, time-out is a widely used prediction scheme based on the assumption that if a device has been idle for a while, it will not be used in the near future.

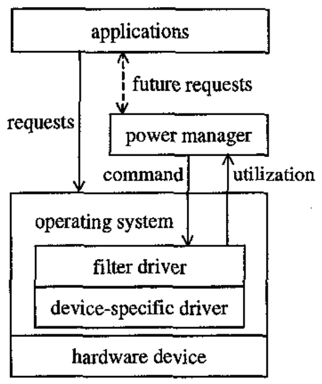

Figure 3: Filter Driver Architecture

Figure 3 shows our software architecture. An FD is attached to the vendor-specific device driver. Both drivers reside in the operating system. Application programs such as word processors or spreadsheets send requests to the OS. A power manager is a separate program that collects device utilization information from the FD and issues state-transition commands to it. When the PM issues a command, the FD creates a power IRP and sends it to the device. The dashed line shows that the PM may communicate with other applications to acquire information about future requests. This feature requires support from the applications. There are two major advantages in our software architecture.

First, in contrast to Figure 1, the power manager resides outside the operating system in this architecture. As a result, designers can evaluate DPM algorithms on real machines without considering the details of ACPI. With a template, system designers do not have to deal with the interactions between the OS, the device drivers, and individual devices. As a result, the product development process is shortened. Our software architecture facilitates the process and encourages designers to evaluate different DPM alternatives.

Second, application programs can request task-specific power management on multiple devices by communicating with the PM. Consider a viewer program that downloads and presents real-time news from the Internet. The program requires the network adapter, the graphics adapter and the hard disk running at full performance. Turning off one of these three devices would disturb the presentation and seriously affect user satisfaction. Instead, the program can inform the PM to prevent the OS from shutting down any of these devices. Task-specific power management makes predictive wake-up a feasible solution to improve performance while saving power. Predictive wake-up was first proposed in [16] to reduce the performance penalty during wake-up; however, accurate prediction is difficult for operating systems because they do not have enough knowledge about the future behavior of applications. Inaccurate prediction may either waste energy (wake up too early) or degrade performance (wake up too late) [5]. Our template makes it possible to perform predictive wake-up since PM can get information from applications about their future behavior to achieve high performance with low power consumption.

4. DPM Algorithms

Several shutdown algorithms were proposed to reduce power consumption [7] [9] [13] [15]. However, "time-out after idleness" is the only shutdown algorithm available on most personal computers running Windows. Users set a time-out value, typically several minutes, in Windows; it detects all disk activities and shuts down a hard disk when it is idle longer than the value. In our evaluation version of Windows NT, the minimum time-out is three minutes.

We implemented two power managers for comparison. The first is a time-out algorithm that performs its own idleness detection by communicating with other programs. Whenever an application issues a disk access, a message is sent to the PM. The PM keeps track of the timestamps of all accesses and shuts down the disk when the last access occurs more than one minute ago. The second is is an adaptive algorithm [11] that dynamically adjusts the time-out value by assuming that disk accesses are clustered into sessions with varying durations. It periodically checks whether a disk access occurred in the last period. If the disk is in the spinning state and no access occurs, the time-out value is decremented on the assumption that the disk has served a short session. If an access occurs and the time-out value is smaller than a threshold, the value is incremented to avoid shutting down the disk too early in a long session. By adjusting the value, the PM can shut down the disk earlier for shorter sessions and later for longer sessions. Figure 4 shows the adaptive algorithm, fully described in [11].

5. Experimental Results

We used an ACPI-compliant personal computer running Windows NT 5.0 beta for the experiments. The com-

```
/* PL/AL: predicted/actual session length */
/* a: attenuation factor */
/* SE: predicted session end time */
/* Th: threshold; inc: increment constant */
switch(state) {
 case spinUp:
  state = spinning; PL = a * PL + (1 - a) * AL;
  SE = now + PL; break;
 case spinDown:
  state = sleeping; break;
 case sleeping:
  if (a request arrives) { state = spinUp; }
  break;
 case ?spinDown:
  if ((now > SE) && ((now - SE)/PL) < Th1)
   { state = sleeping; }
  else { state = spinning; }
  break;
 case spinning:
  if (a request arrives) {
   if ((now > SE) && ((now - SE)/PL < Th2))
    /* almost ready to shut down; defer SE */
    { PL += inc1; SE += inc2; }
  } else {
   state = ?spinDown; PL -= inc1; SE -= inc2; }
  break;
}
```

Figure 4: Adaptive Algorithm

puter contains an IBM Deskstar EIDE hard disk. The 12V and 5V power lines go through two digital multimeters as shown in Figure 5. Both meters contain RS-232 ports for computerized measurements. The hard disk can be in one of three states: D0 when it is reading or writing, D1 when the plates are spinning and D3 when the plates stop spinning. This hard drive does not support the D2 state. I/O requests only wait for seek and rotation delays when the disk is at D1. If a request arrives when the hard disk is at D3, it has to wait for the wake-up procedure in addition to the seek and rotation delays. Figure 6 shows the transitions among these power states. We measured the time and current and found that the disk consumed 3.48 W and 0.75 W in states D1 and D3 respectively. It took approximately 8.1 seconds and 53 Joule to wake up the disk from D3 to D0. It took 1.1 seconds to enter D3 from D1. We did not consider the shutdown energy because it is negligible and the shutdown time is too short to make an accurate estimation. The energy at D0 for reading and writing is ignored because DPM cannot change the number of disk accesses; the power manager merely shuts down the disk when there is no access.

Two applications (AP1 and AP2) were used as sample workloads for comparison. Repeatable disk accesses were necessary for fair comparison; therefore, we did

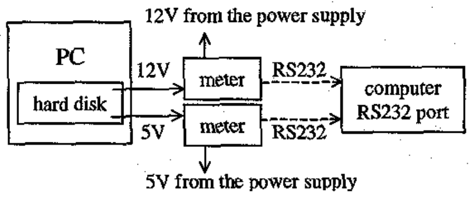

Figure 5: Measure Hard Disk Power

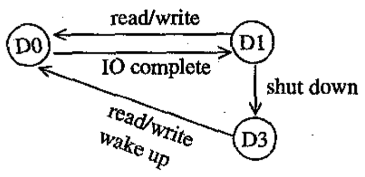

Figure 6: Power State Transition

not directly compare power saving in user-interactive environment due to the difficulty of exactly regenerating user requests. The first workload reads, writes, appends and seeks files. The second copies, deletes, and searches for files, and compiles programs. Each application has three identical processes running concurrently. The time between file accesses is uniformly distributed between 30 seconds and 10 minutes. The time-out value for the first PM was one minute; the initial time-out for the second PM was two minutes and the increment or decrement factor was two seconds according to the principles explained in [11]. We measured the power consumption by each algorithm for half an hour and compared the results with two alternatives: an "always-on" algorithm that kept the disk spinning to provide the maximal performance, and the Windows PM with three-minute time-out. Table 1 shows the comparison. The first row shows the sleeping time (the duration when the disk is at D3). The second row shows the numbers of shut-down commands issued from the PM; a larger number indicates more requests have to wait for the disk to wake up. This is the performance penalty for power saving. The third row shows the total energy consumed. We calculated the energy by the formula: $energy = time_{D3} \cdot power_{D3} + time_{D1} \cdot power_{D1} + \#wakeup \cdot energy_{wakeup}$. The last row normalizes the total energy by the result from the PM in Windows.

The table shows that the power manager in Windows had fewer wake-ups; in other words, fewer disk accesses suffered from the wake-up delay. However, this performance advantage was achieved by consuming more energy. When AP1 and AP2 were executed at the same

	always on			Windows			one-min			adaptive		
workload	AP1	AP2	both	AP1	AP2	both	AP1	AP2	both	AP1	AP2	both
sleep time (sec)	0			183	316	99	634	863	801	807	889	960
# shut-down	0			4	3	2	11	10	13	12	8	18
energy (J)	6264			5976	5560	6100	5116	4438	4766	4697	4261	4597
ratio	1.05	1.13	1.03	1.00	1.00	1.00	0.86	0.80	0.78	0.79	0.77	0.75

Table 1: Power Consumption of Different Policies

time, Windows rarely found a chance to shut down the disk. The one-minute time-out algorithm saved up to 22% of power compared to the PM in Windows. The adaptive algorithm consistently outperformed the other algorithms and saved as much as 25% of power. This encouraging result shows the potentially large design space for power management algorithms.

6. Conclusion

We designed and implemented an ACPI-based software architecture that allows power managers to be implemented at the application level. The architecture provides a uniform interface for designers to investigate power management algorithms on different hardware devices. We implemented two algorithms and measured the power of a hard disk and showed that they could outperform the power manager in Windows by as much as 25%.

7. Acknowledgments

This work is supported by MARCO/SIA. We would like to thank Patrick Burke and Charles Orgish at Stanford and Luca Benini and Alessandro Bogliolo at DEIS-Università di Bologna for their assistance and valuable comments.

8. References

[1] ACPI. http://www.teleport.com/˜acpi.

[2] L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management of Electronic Systems. In *International Conference on Computer-Aided Design*, pages 696–702, 1998.

[3] L. Benini and G. D. Micheli. *Dynamic Power Management: Design Techniques and CAD Tools.* Kluwer, 1997.

[4] E.-Y. Chung, L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management for Non-Stationary Service Requests. In *Design Automation and Test in Europe*, 1999.

[5] F. Douglis, P. Krishnan, and B. Marsh. Thwarting the Power-Hungry Disk. In *USENIX Winter Conference*, pages 293–306, 1994.

[6] W. Fornaciari, P. Gubian, D. Sciuto, and C. Silvano. Power Estimation of Embedded Systems: A hardware / Software Codesign Approach. *IEEE Transactions on VLSI Systems*, 6(2):266–275, June 1998.

[7] C.-H. Hwang and A. C. Wu. A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation. In *International Conference on Computer-Aided Design*, pages 28–32, 1997.

[8] Intel, Advanced Power Management Overview. http://developer.intel.com/IAL/powermgm/apmovr.htm.

[9] K. Li, R. Kumpf, P. Horton, and T. Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *USENIX Winter Conference*, pages 279–292, 1994.

[10] J. R. Lorch and A. J. Smith. Software Strategies for Portable Computer Energy Management. *IEEE Personal Communications*, 5(3):60–73, June 1998.

[11] Y.-H. Lu and G. D. Micheli. Adaptive Hard Disk Power Management on Personal Computers. In *Great Lakes Symposium on VLSI*, 1999.

[12] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh. Techniques for Low Energy Software. In *International Symposium on Low Power Electronics and Design*, pages 72–75, 1997.

[13] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. In *Design Automation Conference*, pages 182–187, 1998.

[14] A. Raghunathan, N. K. Jha, and S. Dey. *High-Level Power Analysis and Optimization.* Kluwer, 1998.

[15] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen. Predictive System Shutdown and Other Architecture Techniques for Energy Efficient Programmable Computation. *IEEE Transactions on VLSI Systems*, 4(1):42–55, March 1996.

[16] J. Wilkes. Predictive power conservation. Technical report, Hewlett-Packard, HPL-CSP-92-5, 1992.