Fine-Grained Energy Profiling for Power-Aware Application Design

Aman Kansal
Microsoft Research
One Microsoft Way, Redmond, WA
kansal@microsoft.com

Feng Zhao
Microsoft Research
One Microsoft Way, Redmond, WA
zhao@microsoft.com

ABSTRACT

Significant opportunities for power optimization exist at application design stage and are not yet fully exploited by system and application designers. We describe the challenges developers face in optimizing software for energy efficiency by exploiting application-level knowledge. To address these challenges, we propose the development of automated tools that profile the energy usage of various resource components used by an application and guide the design choices accordingly. We use a preliminary version of a tool we have developed to demonstrate how automated energy profiling helps a developer choose between alternative designs in the energy-performance trade-off space.

1. INTRODUCTION

Energy optimization in software design is gaining importance due to increases in the recurring energy utility bill as well as the high cost of power provisioning and backup infrastructure, along with growing citizen concerns about the carbon footprint of computing [14, 5, 6]. There is a significant need and potential for energy optimization as large parts of the IT infrastructure operate at a low average utilization and the idle power consumption of a computer is typically 50-60% of peak power. Energy optimization is also important for mobile or wireless devices due to limited battery life.

There are at least three levels at which energy optimizations are being made. The first is at the computer architecture level. Techniques have been proposed to reduce processor energy usage through multi-core processors [15], designing energy proportional hardware [1], and increasing the number of performance and sleep states [22]. Benchmarking techniques to quantify the energy costs of different computer architectures have been proposed [29]. A second level of optimizations is at the system layer, either at a single server or across multiple servers, through energy efficient use of the underlying hardware. In our own work, we developed load management techniques across servers to reduce the energy of one large-scale Internet service by 30% without a significant performance hit [3]. Other techniques include multi-server power coordination [24], hard disk spin-down [21], synchronization aware multithreading [23], and compiler driven optimizations [30, 26].

A third level of optimizations is at the application layer, which is attractive for several reasons. First, this layer has the most information on the actual user impact of performance and energy trade-offs, enabling more aggressive performance sacrifice in unimportant areas compared to lower layer techniques. For instance, an application may throttle processor frequency to the minimum value required to satisfy user perceptible delay behavior [16]. Second, application specific optimizations can be made at this layer such as changing the algorithm used [28], accuracy of computation (eg. changing from double precision to single), or quality of service provided [10]. Third, energy usage at the application layer may be made dynamic [8]. For instance, an application hosted in a data center may decide to turn off certain low utility features if the energy budget is being exceeded, and an application on a mobile device may reduce its display quality [11] when battery is low. This is different from system layer techniques that may have to throttle the throughput resulting in users being denied service.

While many application specific energy optimizations have been researched, there is a lack of generic tools that a developer may use at design time. Application specific optimizations require significant development effort and are often only applicable to specific scenarios. We propose to address this problem by providing automated tools for fine-grained application energy measurements at design time. The aim is to provide the developer with much needed visibility into an application's energy usage. The measurements will allow an application developer to easily compare and choose among various design options that require only simple changes in the source code such as choosing among different library function calls, declaring variables using alternative data structures or precision, changing the storage access patterns, or others that are already available as part of the development environment but are hard to select for the particular application under development. Detailed energy visibility also enables the selection of appropriate knobs for performance tuning which can then be used at run time to dynamically change energy usage for varying workload and energy availability.

The **contributions** of this paper are twofold. The work addresses the research challenges in enabling generic application layer energy optimization, and describes the tools that are needed for application design-time energy profiling and performance scaling. We show that many of the building blocks required to build such tools are already available and discuss the research issues to be addressed in order to integrate these components into reusable measurement capabilities.

We conducted experiments using a preliminary version of the proposed measurement tools to demonstrate an example of power-aware application design: choosing among two different library routines for the same functionality. The design choice does not change the number of lines of code needed, but can impact the energy usage. Both library routines already exist as part of an available development tool set and our objective is not to design a specific alternate routine with optimized energy consumption.

The goal of the paper is to propose the development of automated fine-grained energy profiling tools as an important research area in addressing the significant potential for application layer energy optimization. These tools should be integrated with compilers and integrated development environments to provide visibility into the energy impact of application design choices. Such tools could lead to a significant reduction in the energy cost of the global IT infrastructures with a relatively small additional effort in design.

2. APPLICATION ENERGY USAGE

Let us first consider the various components of energy consumption for an application. Suppose the energy consumed by the computing infrastructure for executing an application is E_{App} which may depend on the performance required for the application, and also on the specific system resources used by the application.

Part of E_{App} is spent on running the application software and underlying system software. Let E_{Active} denote this part. Another portion of the energy, denoted E_{Wait}, is spent in wait states, when a subsystem is powered up but the application is using another subsystem. Ideally, unused subsystems should enter a low power sleep state. However, entering a sleep state and resuming from it has a non-zero overhead, making it impractical to enter sleep states for small periods of inactivity. Some subsystems may not have easy to use low power sleep modes and the idling cost is paid even if the subsystem is not used at all by the application. The rest, denoted E_{Idle}, is consumed by the computing equipment in idle state, when not doing any work for any application. However, this idle energy may depend on the application mix hosted by a system since with a certain mix, the system may be able to enter sleep states while with another mix, the idle periods may be too small to enter sleep states.

Another energy component of interest in operating a computing infrastructure is the energy spent on cooling the computer equipment. This is the energy spent on transferring out the energy E_{App} that is dissipated as heat inside a server or data center. The exact amount of energy spent depends on several factors, including server chassis design, location within data center, and efficiency of the cooling system design. However, a simple assumption that can be made at design time is that reducing E_{App} will help reduce cooling energy as well.

Thus, the key **challenge** in quantifying E_{App} is that given a computer system configuration, we need to obtain a detailed understanding of E_{Active}, E_{Wait}, and E_{Idle} for each application, where the wait state and idle state energies are attributed to the responsible applications correctly. Obtaining the total energy consumed by an application is not enough to characterize these metrics individually. Further, optimizing the application design will require some insight into how the energy used is spread across various system *resources* such as the CPU, disk, memory, or a network interface, as well across various *semantic activities* within the application. Hence, a detailed energy breakdown by each resource and activity is needed. This detailed breakdown must be provided for each individual application under test rather than the system as a whole since the system may be running other applications in parallel, and both the system resources and certain OS services will be shared across applications. Generating appropriate representative workloads at profile time is also necessary to address the above challenge.

3. DESIGN TIME PROFILING AT WORK

Various design choices in the application may affect the power used in complex ways and detailed visibility into energy usage is needed to develop generic and transparent energy optimization techniques at the application layer.

We propose fine grained application energy profiling as a useful tool that enables application developers to make energy efficient design choices. Alternative methods may exist to achieve the same functionality, such as by using a different library routine, and the developer can use detailed profiling to determine the best choice. The developer may have access to a power management technique to improve the energy usage of one resource but since it may affect the energy usage of other resources, accurate profiling is required to determine the overall energy usage changes. The developer may have new ideas to optimize energy by designing a new algorithm or data structure. The profiling tools will help understand the quantitative advantage of the new idea. The developer may also know which aspects of the application performance are flexible and the profiling tools may be used to determine where the energy savings are the maximum.

Before discussing the various challenges in building such fine grained power measurement tools, we show an example to demonstrate how this proposed measurement capability enables application layer energy optimization at design time.

Consider an application that processes a file containing several data items. There are two choices of the file read-write routines available to the developer to access the secondary storage: one that compresses the stored data and one that does not. Both alternatives are available in existing development tools, through the `FileStream()` and `GZipStream()` classes, part of the .Net Framework's System.IO and System.IO.Compression class libraries respectively, and the application developer wishes to determine the best choice. In this simple example, we assume we have available a data file representative of real data files that the application will process after deployment. Partial source code used for the two alternatives is shown below:

Alternative 1: Without Compression

```
f = new FileStream("/var/records.mds");
f.Open("rw");
/* Workload: read/edit data items */
f.Close();
```

Alternative 2: With Compression

```
f = new GZipStream("/var/records.zds");
f.Open("rw");
/* Workload: read/edit data items */
f.Close();
```

Both options are equally easy to code for the developer since both library routines exist and the choice of the routine does not change the application processing logic in any way. However, they internally use different amount of processing and result in different resource usage: compression will reduce IO load but may increase CPU load. The actual costs depend on the data processed. For particular application domains such as video streaming, compression may be well-known to be useful due to constraints on the bottleneck resource, but in general, the deign choice is not obvious for a given application. Energy profiling may be applied to guide this choice.

3.1 Prototype Tool

We composed an illustrative version of the proposed profiling tool for the measurements in this examplle. A block diagram of our tool is shown in Figure 1.
The tool consists of three major components:

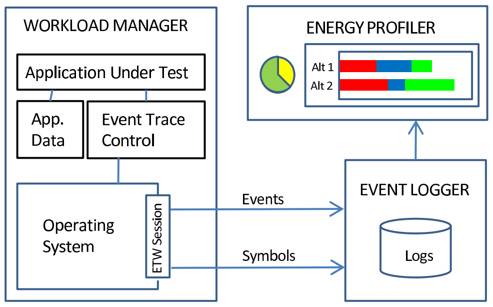


Figure 1: Application Energy Measurement Tool

1. Workload Manager. This component initiates the event tracing and runs the application code with a fixed data workload. Each time any change is made to the application code, the workload manager may invoke the measurement process and initiate the event tracing for it. The workload manager uses the Event Tracing for Windows (ETW) [19] kernel level tracing system for generating events corresponding to operating system operations, including CPU use, disk I/O and page faults, heap range creation, and context switches.
2. Event Logger. This component logs the events generated by ETW to a log file. It uses Windows Xperf [20] for logging.
3. Energy Profiler. This is the component that parses event traces to associate resource usage with the application of interest and outputs the energy usage of the application across various system resources. The design of this component involves several challenges, discussed in section 4. In our illustrative version, we parse the event information to count the resource usage and convert this to energy data using the power specifications of the resources used. The system events contain significant detailed information such as event timestamps, process ID's, thread ID's, and files being used. We use events corresponding to CPU usage and disk activity for the processes corresponding to the application. System disk activity that corresponds to the files used by the test application are also counted as part of the application's resource use, since this activity occurs only to support that particular application. This version of our tool does not provide the energy usage of DRAM or network resources.

3.2 Experimental Data

For the example application considered above, the tool outputs the following data. Figure 2 shows the CPU usage history over the execution time duration for both alternatives. Figures 3(a) and 3(b) show the disk activity, integrated over one second intervals. The figures shows that the CPU usage increases dramatically when compression is used. This may be seen by the developer using existing tools such as `top` in Unix and the `Task Manager` in Windows. Also, the disk usage is reduced significantly with compression.

The energy numbers obtained by our tool for both alternatives are shown in Table 1. The data shows that using the compressed file stream in fact saves energy and improves the performance (total time to process the offered workload). Here f_{Active} represents the CPU percentage time spent on running the application, E_{CPU} represents the energy used by the CPU, and T is the time spent running the application workload. In our experiment, E_{CPU}, is computed by adding the idle and active mode energy consumptions:

$$E_{CPU} = \{P_{Active} * f_{Active} + P_{Idle} * (1 - f_{Active})\} * T$$

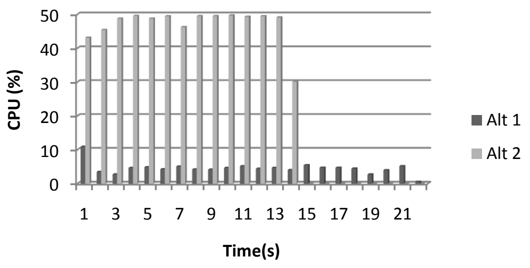
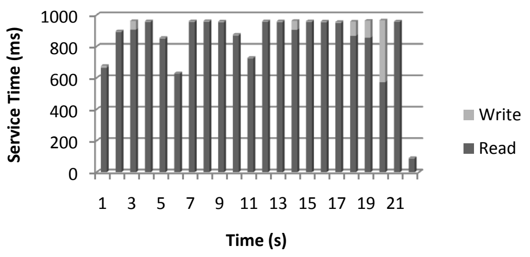
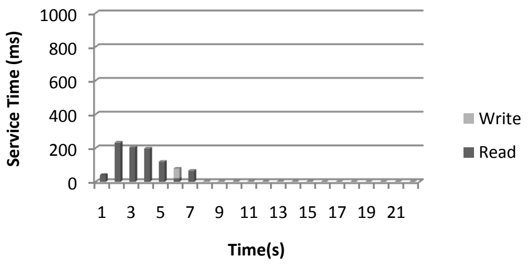


Figure 2: CPU loads for the two alternatives.



(a) Alternative 1.



(b) Alternative 2.

Figure 3: Disk Load.

where P_{Active} is the power consumption of the CPU in active mode and P_{Idle} is the power consumption in idle mode. The above formula assumes that all time not spent on servicing the application is idle time. This is an approximation, as the CPU spends some extra time on system layer tasks. In our measurements, the CPU percentage for such activities was small. In actual practice, that energy would be amortized over multiple applications running on the CPU and hence is not included here. A similar formula was used for computing the energy consumed in the disk as well. The data file being processed by the application contained a set of event logs collected by our tool, and compressed from 1.183GB to 71.99MB when used with `GzipStream()`. The machine used was a Lenovo Thinkpad T61p, with an Intel Core 2 Duo processor with two 2.2GHz cores and a Hitachi 7200rpm disk drive. The workload consisted of reading each data item, editing a few random bytes in each item, and writing it back.

If such detailed visibility into energy costs is available to the developer at design time with each design change or additional feature added to the application, then it becomes easier to optimize the energy footprint of the application. At times this visibility may also help improve performance, such as in the above example, or trade-off performance for cost savings.

Table 1: Total and Breakdown of Energy Usage.

Cost	Uncompressed IO	Compressed IO
Total Time, T (s)	22.358	14.935
CPU: f_{Active} (%)	4.37	44.25
E_{CPU} (J)	409.12	377.56
Disk Active (s)	19.218	0.965
Disk Energy (J)	92.96	25.29
Total Energy (J)	502.08	402.85

4. PROFILING TECHNIQUES AND CHALLENGES

Existing power profiling strategies such as [29, 25] quantify the total energy used by an application, with varying throughput. These techniques focus on architecture and system layer optimizations rather than on the application. Such aggregate power profile information is not sufficient for designers to take full advantage of application layer optimizations since this information does not necessarily reveal exactly how the energy usage is split across different parts of the application or among different resources used by the application. Also, the opportunity for trading-off application performance for cost savings in energy is not directly exposed through such techniques. Profiling at the application layer is typically limited to manual methods that evaluate specific design modifications proposed in a given problem context. Hence, new research challenges remain in the design of automated fine grained power profiling tools.

There are several ways to measure energy usage. A first approach is to estimate it purely based on analytical models. Theoretical analysis of an algorithm's execution complexity can be related to the approximate energy usage using the processor, memory, and storage power models [13, 12]. While this approach is very useful for making early stage design decisions, it quickly becomes intractable when the effect of various system layer optimizations, memory hierarchies, IO buffering, and the use of low power states is to be modeled. Also, such an approach is not straightforward to automate, limiting its use in practice.

A second approach is to use power models for simulating the application execution instead of analytical estimation. Hardware power models obtained off-line were used in [27] to simulate the application energy usage. The application was instrumented to record the power state transitions of various resources such as the processor and network interface, and the recorded traces were post-processed to estimate the energy usage. Such a technique suffices when a single application is running on the system, but when each system resource is being used by multiple applications, we need finer grained measurements to associate the energy consumed with corresponding applications.

A third approach, also the one discussed in this paper, is to measure the application energy by executing it. This has the potential to provide most accurate information. One alternative to enable such measurement is to instrument each resource in hardware to measure energy usage, such as in [17] where hardware instrumentation was used in an embedded platform to enable monitoring the energy used by each application or hardware resource. This approach is very useful but has high cost overheads. Most existing computing equipment does not have fine granularity power measurement hardware instrumentation and adding such instrumentation is not cheap. Current power measurement capabilities that exist, such as those exposed through the Advanced Configuration and Power Interface (ACPI)[1] or server motherboard/chassis instrumentation being standardized through Intelligent Platform Management Interface (IPMI)[2], are very coarse grained. They provide total energy consumption or chassis level sensing rather than for each individual resource, and also at a very low sampling rate compared to the operating system context switches, limiting the use of such instrumentation to measure the power usage for each thread or process individually on any resource.

[1]http://www.acpi.info/

The key research challenge that remains is the development of energy profiling tools that can yield such fine grained measurements. In our illustrative tool we followed the approach of kernel supported resource counting and associating that with energy usage. This approach is promising because it leverages existing system capabilities but has the potential to yield fairly accurate measurements. Several challenges remain in fully realizing its potential but many useful building blocks already exist that make the problem tractable. Kernel instrumentation, such as one used in our illustrative tool, already exists in most modern operating systems, allowing easy tracking of resource usage for individual applications. These measurements are not directly related to power usage but techniques to solve this problem are under development. In [4], energy models for various resources were measured off-line and the resource usage of a given workload was tracked at run time. The resource usage was converted to energy consumption using the previously obtained off-line power models. Exploring this approach further may be helpful for obtaining fine grained measurement where the resource usage is tracked at individual process or thread level and the associated energy consumption is computed for each application by resource. In [9], a complete system model that allows estimating an application and operating system's energy usage on a given platform was developed. Techniques to associate hardware resource counters with system energy usage were also developed in [2]. Combining such techniques with appropriate analytical and simulation based estimates to make up for lack of fine grained measurement infrastructures, becomes an interesting research problem.

Tools also exist for tracking a program's time spent on various methods and resources, using software instrumentation, such as Microsoft-CLR Profiler [18], and gprof [7]. However, these require programmer effort in instrumenting the application, and automating this step would be crucial to make the energy profiling tools operate transparently at application design time.

In summary, the key challenges for initial exploration may be summarized as:

Track E_{Active}, E_{Wait} and E_{Idle}: Track exact resource usage and associate it with the power consumption, visualized per resource. Automated event tracing and parsing will be required. The techniques must minimize the hardware instrumentation required for profiling.

Attribute E_{Wait}, E_{Idle} correctly: The idle energy cost and the energy spent in wait states depends on the application mix. For instance, introduction of a new application may cause the system to spend more time in idle state instead of sleep state. Each resource, and even software services in the system may be shared across multiple applications. This makes it non-trivial to attribute the wait and idle energy costs to relevant applications and activities within applications.

Provide Workloads: Provide techniques to generate workloads used during design time profiling. Standard benchmarks may not cover a specific application's usage scenario, and hence specific workloads may be needed. In some scenarios, these may be easy to obtain (eg., a yellow pages query application would know its dataset and typical queries in advance) while in others, non-trivial

[2]http://www.intel.com/design/servers/ipmi/index.htm

effort may be needed. Techniques to automatically generate a range of workloads that stress different aspects of the application and methods that minimize developer input for this task, are required.

Support remote profiling: In many cases, applications developed on a workstation are meant to run on very different hardware configurations, such as a blade server or a mobile device. The energy footprint could vary greatly between the target device and developer's machine. Tools that enable the developer to profile the application on remote hardware will help obtain accurate profile data for the target hardware.

5. DISCUSSION AND CONCLUSIONS

The profiling tools and challenges discussed above not only enable easier design time energy optimizations but also open the doors to other energy management possibilities.

Energy-Performance Trade-Offs: Understanding the energy usage of an application can be thought of as quantifying the interdependence between three quantities: workload, performance, and energy. This relationship can be very complex for a given application as each application expends energy in multiple resources in a networked computer system. Both the total load on each resource and the pattern with which the load is applied to the resource affect the ability of the resource to enter sleep states. The performance scaling options, sleep states, and transition overheads are not identical for each resource. Further, the resources used could spread across multiple network nodes. The effect of changing power modes of the individual components may affect the performance differently for each application. For instance, an applications needing continuous but variable workload may benefit most from processor frequency scaling and DRAM bank shut-downs, while others with intermittent load may benefit from disk spin-speed reduction or wireless network interface duty cycling. The performance metric itself is multidimensional and consists of various parameters including execution latency, throughput, graphic interface resolution, refresh frequency, etc. The fine-grained application energy measurement techniques discussed above provide a useful set of tools for characterizing the energy-performance relationship for an application.

Hardware Variations: We discussed the design time profiling tools assuming the target hardware is available, either locally or remotely, to perform the measurements. This may not be possible in many situations, such as when the application is to be run on many heterogeneous devices, with vastly different configurations. The automatic conversion of profiles on one system to another is a challenging task. Combining simulation and model based estimation effectively with available profiling, appropriately converting performance and timing information, and modeling the effect of varying power management capabilities, yields several interesting research problems.

Cross Layer Interaction: In certain scenarios, it is desirable to consider cross-layer techniques, so that lower layers can be informed by the decisions at the upper layer for achieving a more global optimum. An interesting characteristics in this interaction is the differences in time scales: the lower layer time scale is typically the shortest, involving hardware power state throttling while the application level has the longest time scale. A research problem to address is the abstraction of control knobs that a higher layer can exploit.

We believe that addressing the challenges discussed in this paper is interesting and useful because automated fine grained energy measurements will facilitate the move from pure performance optimizations to energy-performance aware designs.

6. REFERENCES

[1] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[2] F. Bellosa. The benefits of event driven energy accounting in power-sensitive systems. In *9th ACM SIGOPS European Workshop*, pages 37–42, 2000.

[3] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI*, San Francisco, CA, April 2008.

[4] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full system power analysis and modeling for server environments. In *Workshop on Modeling, Benchmarking, and Simulation*, 2006.

[5] E.P.A. Report on server and data center energy efficiency. Technical report, US Environmental Protection Agency, Energy Star Program, August 2007.

[6] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, pages 13–23, 2007.

[7] J. Fenlason and R. Stallman. Gnu gprof. http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_chapter/gprof_toc.html.

[8] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 48–63, 1999.

[9] S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. John. Using complete machine simulation for software power estimation: The softwatt approach. In *HPCA*, pages 141–150, February 2002.

[10] C. Im and S. Ha. Energy optimization for latency- and quality-constrained video applications. *IEEE Design and Test of Computers*, 21(5):358 – 366, Sept.-Oct 2004.

[11] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan. Energy-adaptive display system designs for future mobile environments. In *ACM MobiSys*, pages 245–258, 2003.

[12] R. Jain, D. Molnar, and Z. Ramzan. Towards a model of energy complexity for algorithms [mobile wireless applications]. In *IEEE Wireless Communications and Networking Conference*, pages 1884– 1890, March 2005.

[13] R. Jain, D. Molnar, and Z. Ramzan. Towards understanding algorithmic factors affecting energy consumption: switching complexity, randomness, and preliminary experiments. In *DIALM-POMC '05: Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 70–79, New York, NY, USA, 2005. ACM.

[14] J. G. Koomey. Estimating total power consumption by servers in the U.S. and the world. Technical report, Lawrence Berkeley National Laboratory, February 2007.

[15] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *ACM/IEEE MICRO*, pages 81–92, 2003.

[16] X. Liu, P. Shenoy, and M. D. Corner. Chameleon: Application Level Power Management. *IEEE Transactions on Mobile Computing*, To Appear 2008.

[17] D. McIntire, T. Stathopoulos, and W. Kaiser. etop: sensor network application energy profiling on the leap2 platform. In *IPSN*, pages 576–577, 2007.

[18] J. Meier, S. Vasireddy, A. Babbar, and A. Mackman. How to: Use CLR profiler. http://msdn2.microsoft.com/en-us/ library/ms979205.aspx.

[19] Microsoft. Event tracing for windows. Microsoft Developer Network, http://www.microsoft.com/whdc/devtools/ tools/EventTracing.mspx.

[20] Microsoft. XPerf. http://msdn2.microsoft.com/en-us/library/cc305187.aspx.

[21] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. In *FAST*, February 2008.

[22] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *NSDI*, San Francisco, CA, April 2008.

[23] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing energy-performance tradeoffs for multithreaded applications on multiprocessor architectures. In *SIGMETRICS*, 2007.

[24] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: A unified multi-level power management architecture for the data center. In *ASPLOS*, March 2008.

[25] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD*, pages 365–376, 2007.

[26] J. Shirako, M. Yoshida, N. Oshiyama, Y. Wada, H. Nakano, H. Shikano, K. Kimura, and H. Kasahara. Performance evaluation of compiler controlled power saving scheme. In *20th ACM Int'l Conference on Supercomputing Workshop on Advanced Low Power Systems (ALPS2006)*, July 2006.

[27] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *ACM SenSys*, pages 188–200, 2004.

[28] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *ACM Mobicom*, pages 181–190, 1998.

[29] Standard performance evaluation corporation. SPECpower. http://www.spec.org/power_ssj2008/.

[30] F. Xie, M. Martonosi, and S. Malik. Compile-time dynamic voltage scaling settings: opportunities and limits. *SIGPLAN Not.*, 38(5):49–62, 2003.