# Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach

Sudhanva Gurumurthi   Anand Sivasubramaniam   Mary Jane Irwin   N. Vijaykrishnan
Mahmut Kandemir
Dept. of Computer Science and Engineering
The Pennsylvania State University, University Park, PA 16802
{gurumurt,anand,mji,vijay,kandemir}@cse.psu.edu

Tao Li   Lizy Kurian John
Dept. of Electrical and Computer Engineering
University of Texas at Austin, Austin, TX 78712
{tli3,ljohn}@ece.utexas.edu

## Abstract

*Power dissipation has become one of the most critical factors for the continued development of both high-end and low-end computer systems. The successful design and evaluation of power optimization techniques to address this vital issue is invariably tied to the availability of a broad and accurate set of simulation tools. Existing power simulators are mainly targeted for particular hardware components such as CPU or memory systems and do not capture the interaction between different system components. In this work, we present a complete system power simulator, called SoftWatt, that models the CPU, memory hierarchy and a low-power disk subsystem and quantifies the power behavior of both the application and operating system. This tool, built on top of the SimOS infrastructure, uses validated analytical energy models to identify the power hotspots in the system components, capture relative contributions of the user and kernel code to the system power profile, identify the power-hungry operating system services and characterize the variance in kernel power profile with respect to workload. Our results using Spec JVM98 benchmark suite emphasize the importance of complete system simulation to understand the power impact of architecture and operating system on application execution.*

## 1 Introduction

Performance optimization has long been the goal of different architectural and systems software studies, driving technological innovations to the limits for getting the most out of every cycle. This quest for performance has made it possible to incorporate millions of transistors on a very small die, and to clock these transistors at very high speeds. While these innovations and trends have helped provide tremendous performance improvements over the years, they have at the same time created new problems that demand immediate consideration. An important and daunting problem is the power consumption of hardware components, and the resulting thermal and reliability concerns that it raises. As power dissipation increases, the cost of power delivery to the increasing number of transistors and thermal packaging for cooling the components goes up significantly [3, 33]. Cooling systems need to be designed to tackle the peak power consumption of any component. These factors are making power as important a criterion for optimization as performance in commercial high end systems design.

Just as with performance, power optimization requires careful design at several levels of the system architecture [13]. At the circuit level, several techniques such as clock gating, supply voltage scaling and supply voltage gating have been proposed to reduce both dynamic and leakage power [5]. Architectural level power saving techniques typically detect idleness of components not being used and appropriately transition them to a lower power consuming mode. Even the software - the operating system (OS), compiler and the application - has an important role to play in power efficient systems design. The operating system, which plays the role of hardware manager, can schedule jobs [23], allocate and manage memory [19], and control peripherals [20] to reduce overall system power. The compiler can generate code and data transformations to increase idleness of hardware components so that they can be transitioned to low power modes more effectively [7] Finally, algorithmic transformations in the application have been shown to give significant power savings [30].

The successful design and evaluation of such optimization techniques is invariably tied to a broad and accurate set of rich tools that are available for conducting these studies. The crucial role of design and evaluation tools for performance optimization has been well illustrated by several studies over the years, and the community at large is currently expending a lot of effort in the development of similar tools for power estimation and optimization. There are tools to facilitate this at the circuit [1], gate [32], and architectural [4, 35, 29] levels. However, power estimation and optimization tools at the architectural and software levels are still in their infancy. These tools are also not well-integrated enough to study several issues all at one, or to see how one optimization affects the complete system behavior and not just the target of the optimization. For instance, with today's tools, one could try out a compiler loop transformation technique by using tools such as SUIF [11] to conduct source-to-source transformations, run the output through gcc to get binaries, and run them on a simulator such as Wattch [4] to evaluate the power savings. This could help us study the impact of the optimization on the datapath, cache and memory power concurrently. However, this still does not tell us how the TLB behavior changes, and how the OS execution is affected as a result. Further, this does not provide us with the complete picture since there are still several other hardware components (e.g. the disk) that could be affected.

A similar observation was made a few years back in the context of performance, leading to a consideration of the complete system in the evaluation. Instrumentation/measurement or hardware profiling using performance counters on an actual platform is one way of complete system performance evaluation [16, 15]. However, we do not have access to all the relevant hardware events for accurate power calculations in complete system power profiling on today's systems. This approach also makes it difficult to perform detailed software and hardware profiling of several components at the same time without significant intrusion. Instead, a simulation based strategy can do the same without the intrusion, albeit at a high simulation cost. This is the strategy adopted in the SimOS [28] tool, which performs a complete system performance simulation of applications running on a multi-issue superscalar processor together with an actual commercial operating system (IRIX 5.3).

With the goal of conducting complete system power profiling (from the hardware and software viewpoint), we have extended SimOS to include power models for different hardware components (the processor datapath, caches, memory, and disk). Consequently, we have a powerful tool that can give us detailed performance and power profiles for different hardware and software components over the course of execution of real applications running on a commercial operating system. While there have been some previous endeavors in building complete system power estimation tools [26, 6, 2, 18, 22], most of these are limited to either specific environments/applications or have targeted embedded platforms. SoftWatt, the tool that is discussed in this paper, is the first one to target complete system power profiles of high end systems, that can be used for design and evaluation of power optimization techniques. Such a tool can help us answer several important issues that are explored in this paper:

- How does the power consumption of the complete system vary during the execution of a program? Does it remain constant, or are there variations/spikes? Are these variations, if any, due to user or kernel activity, and what hardware components are being exercised at those times? Even if the overall power consumption does not vary too much, are there changes between the relative proportions of user, kernel, and idle mode power consumption over time?

- What is the major contributor for power consumption from the software angle? Is it the kernel instructions, user instructions or the idle process (idling in a lot of commercial OS including IRIX is done by busy-waiting and is not necessarily a low power consumer)? What hardware components are the dominant power consumers in the user and kernel modes?

- Within the kernel, what OS services are the dominant power consumers? Why? Do performance and power profiles go hand-in-hand for all these results?

- What is the variation in energy consumption for a kernel service from one invocation to another? i.e., Is the per-invocation energy consumption of a kernel service very data dependent? If not, then from the simulation angle, there can be reasonable simulation speedups to be gained by not getting into the details of simulating kernel services.

- From the detailed hardware and software power profiles, can we make suggestions where future work needs to be directed for significant gains? What kind of hardware/software optimizations are suggested by the power profiles?

This paper examines several of these questions using our tool, SoftWatt, by experimenting with different Spec JVM98 benchmarks [31] together with the Java Virtual Machine (JVM) runtime system executing on IRIX 5.3. The performance and power profiles are given for different hardware components such as the processor datapath, L1 and L2 I/D caches, memory and disk.

The rest of this paper is organized as follows. Section 2 describes the simulation framework. Section 3 presents the power characterization of the Spec JVM98 benchmark suite Section 4 presents the impact of disk power-management to the system power consumption and performance. Section 5 discusses the characterization results and concludes the paper.

## 2 Simulator Design

The first step towards a comprehensive study of the power consumption of a computer system is the development of a suitable simulation infrastructure. SimOS, which provides a very detailed simulation of the hardware so as to be able to run the IRIX 5.3 operating system, is our base simulator. SimOS also provides interfaces for event-monitoring and statistics collection. The simulator has three CPU models, namely, *Embra*, *Mipsy*, and *MXS*. Embra employs *dynamic binary translation* and provides a rough-characterization of the workload. Mipsy provides emulation of a MIPS R4000-like architecture. It consists of a simple pipeline with blocking caches. MXS emulates a MIPS R10000-like [36] superscalar architecture. The overall design of the energy simulator is given in figure 1.

We modified MXS CPU and the memory-subsystem simulators to instrument accesses to their different components. This enables us to analyze our simulations using the Timing Trees [12] mechanism provided by SimOS. The MXS CPU simulator does not report detailed statistics about the memory subsystem behavior. Due to this limitation in SimOS, we use Mipsy for obtaining this information.

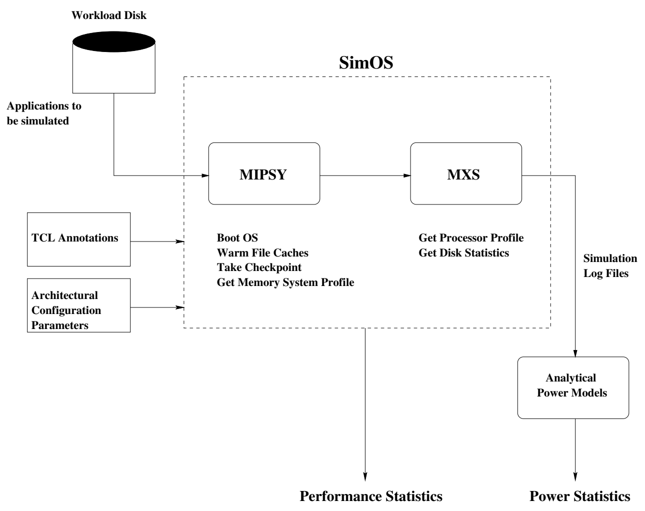
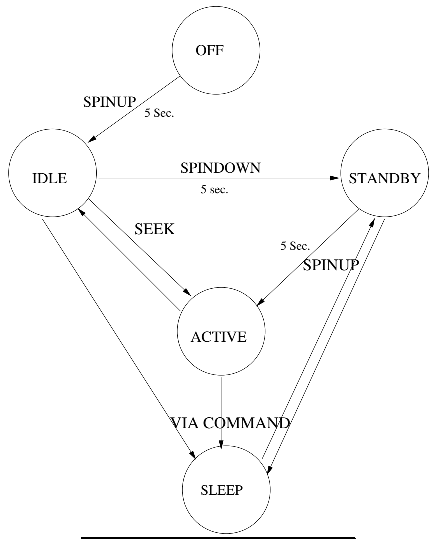


**Figure 1. Simulator Design**



| Mode | Power (W) |
|---|---|
| Sleep | 0.15 |
| Idle | 1.6 |
| Standby | 0.35 |
| Active | 3.2 |
| Seeking | 4.1 |
| Spin up | 4.2 |

**Figure 2. MK3003MAN Operating Modes State Machine and Power Values**

MXS is used to obtain detailed information about the processor.

We also incorporated a disk-power model into SimOS for studying the overall system power consumption. SimOS models a HP97560 disk. This disk is not state-of-the art and does not support any low-power modes. We therefore incorporated a layer on top of the existing disk model to simulate the TOSHIBA MK3003MAN [34] disk, a more representative modern disk that supports a variety of low-power modes. The operating-modes state-machine implemented for this disk is shown in figure 2. The disk transitions from the IDLE state to the ACTIVE state on a seek operation. The time taken for the seek operation is reported by the disk simulator of SimOS. This timing information is used to calculate the energy consumed when transitioning from the IDLE to the ACTIVE state. In the IDLE state, the disk keeps spinning. A transition from the IDLE state to the STANDBY state involves spinning down the disk. This operation incurs a performance penalty. In order to service an I/O request when the disk is in the STANDBY state, the disk has to be spun back up to the ACTIVE state. This operation incurs both a performance and energy penalty. The SLEEP state is the lowest-power state for this disk. The disk transitions to this state via an explicit command.

We assume that the spin up and spin down operations take the same amount of time, and that the spin down operation does not consume any power. Our model also assumes that transition from the ACTIVE to the IDLE state takes zero time and power, as in [20]. Currently, we do not utilize the SLEEP state. We suitably modified the timing modules of SimOS to accurately capture mode-transitions. While it is clear that modeling a disk is important from the energy perspective, the features of a low-power disk can also influence the operating system routines such as the idle process running on the processor core. Hence, a disk model helps to characterize the processor power more accurately. During the I/O operations, energy is consumed in the disk. Further, as the process requesting the I/O is blocked, the operating system schedules the idle process to execute. Therefore, energy is also consumed in both the processor and the memory subsystem.

SoftWatt uses analytical power models. A post-processing approach is taken to calculate the power values. The simulation data is read from the log-files, pre-processed, and is input to the power models. This approach causes the loss of per-cycle information, as data is sampled and dumped to the simulation log-file at a coarser granularity. However, there is no slowdown in the simulation time beyond that incurred by SimOS itself. This is particularly critical due to the time-consuming nature of MXS simulations. The only exception to this rule is the disk-energy model, where energy-consumption is measured during simulation to accurately account for the mode-transitions. This measurement incurs very little simulation overhead. SoftWatt models a simple conditional clocking model. It assumes that full power is consumed if any of the ports of a unit is accessed; otherwise no power is consumed.

The per-access costs of the cache-structures are calculated based on the model presented in [17, 4]. The clock generation and distribution network is modeled using the technique proposed in [9], which has an error-margin of 10%. The associative structures of the processor are modeled as given in [25, 4]. In order to validate the entire CPU model, we configured SoftWatt to calculate the maximum CPU power of the R10000 processor. In comparison to the maximum power dissipation of 30 W reported in the R10000 data sheet [27], SoftWatt reports 25.3 W. As detailed circuit-level information is not available at this level,

| Parameter | Value |
|---|---|
| Instruction Window Size | 64 |
| Register File | 34 INT, 32 FP |
| Load/Store Queue | 32 |
| Fetch Width per Cycle | 4 |
| Decode Width per Cycle | 4 |
| Issue Width per Cycle | 4 |
| Commit Width per Cycle | 4 |
| Functional Units | 2 Ints,2 FP |
| Branch History Table | 1024 |
| Branch Target Address Table | 1024 |
| Return Address Stack | 32 |
| Memory Size | 128 MB |
| Cache Hierarchy | 2-Level with L1 D- and I-Cache and Unified L2 Cache |
| Instruction Cache Size | 32KB |
| Instruction Cache Line Size | 64B |
| Instruction Cache Associativity | 2 |
| Data Cache Size | 32 KB |
| Data Cache Line Size | 64B |
| Data Cache Associativity | 2 |
| L2 Cache Size | 1MB |
| L2 Cache Line Size | 128B |
| L2 Cache Associativity | 2 |
| Unified TLB (fully assoc) entries | 64 |
| Feature Size | 0.35 um |
| Vdd | 3.3 V |
| MHz | 200 |

**Table 1. System Model**

generalizations made in the analytical power models result in an estimation error.

## 3 Power Characterization of Spec JVM98 Benchmarks

### 3.1 System Configuration and Benchmarks

Table 1 gives the baseline configuration of SoftWatt that was used for our experiments. We chose the Spec JVM98 benchmarks [31] for conducting our characterization study. A description of these benchmarks is given in [10]. We chose this benchmark suite for two reasons. First, Java is becoming an increasingly popular language in many power-critical applications. Second, Java applications are also known to exercise the operating system more than traditional benchmark suites [21]. Thus, they form an interesting suite to characterize for power using our complete system power simulator, which includes the operating system as well. We excluded the `mpegaudio` benchmark from our characterization study, as it failed to execute on the MXS simulator. For all other Spec JVM98 benchmarks, the just-in-time (JIT) compilation mode was employed, and the s10 dataset was used. This dataset exercises the Java garbage-collector and the simulation also completes in a reasonable amount of time. On the average, the s10 dataset took 30 hours per application to complete on a 400 MHz Sun Sparc. All the benchmarks were initially run on the relatively fast Mipsy simulator. The file-caches were warmed and a checkpoint was taken before the program was loaded. Then, the benchmarks were run on the Mipsy simulator to get statistics about the memory system behavior, and then on MXS to get processor statistics. For all the benchmarks, the profiled period consists of all the phases of the Java application execution until the program exits and returns to the shell-prompt.

In our characterization of the benchmarks, we use two metrics, namely, *power* and *energy*. We focus on the *average* power consumption. In the presence of dynamic thermal management techniques, a system can be designed accounting for average power consumption instead of peak power [3]. Our tool can also be used to obtain the peak power consumption from the profiles. Energy consumption is an important metric when considering battery life in high-performance mobile systems such as laptop computers. We also use an additional metric, namely, the *Energy-Delay Product (EDP)*, which captures the tradeoffs in design decisions for energy versus that for performance. Unless explicitly mentioned, the term ”power consumption” refers to the power consumed in the processor datapath and the memory subsystem only.

### 3.2 Characterization

Figures 3 and 4 give the performance and power profiles of the `jess` benchmark, for four different phases (modes) of execution: user mode, kernel mode, kernel synchronization and idle. The profiles for the other benchmarks is given [10]. The kernel execution is split into the portion where instructions are executed and the portion where synchronization operations are performed. Idle refers to the idle times during the workload's execution. In Figure 3, the first two profiles, from the left, were obtained by executing them on Mipsy. The third profile was obtained by configuring MXS to be a single-issue processor. It is clear that the average power of the memory subsystem is more than twice that of the processor datapath. Therefore optimizing the memory subsystem is very important to reduce the overall power consumption in a single issue machine. The processor profiles in figure 4 were obtained on MXS. In the power profiles, the influence of clock-loading is not shown.

The bulk of the time is spent executing user-instructions. Also, the benchmarks show a greater percentage of operating system activity in the superscalar machine compared to that of the single-issue configuration. On the average, the percentage of kernel activity increases from 14.28% in a single-issue processor to 21.02% in the superscalar processor. This is because kernel code has a lower IPC (Instructions Per Cycle) and worse branch-prediction accuracy compared to the user code [21]. In all the power profiles, it is observed that the idle-mode initially dominates the power consumption, and then significantly decreases. The same trend can be seen in the execution profile as well. The initial idle-periods are due to the loading of the Java class files from the disk when executing the benchmarks. After this period, the required data is found in the file-cache most of the time. Further, it is observed that the memory subsystem energy increases steeply at the beginning of the profiled period. This is because of the cold-start misses in the caches, which cause several memory accesses. As the L2 cache and memory have a high per-access cost, the average power is high at the beginning. After the initial period, both the profiles even out, with the subsystems consuming a more or less constant amount of power.

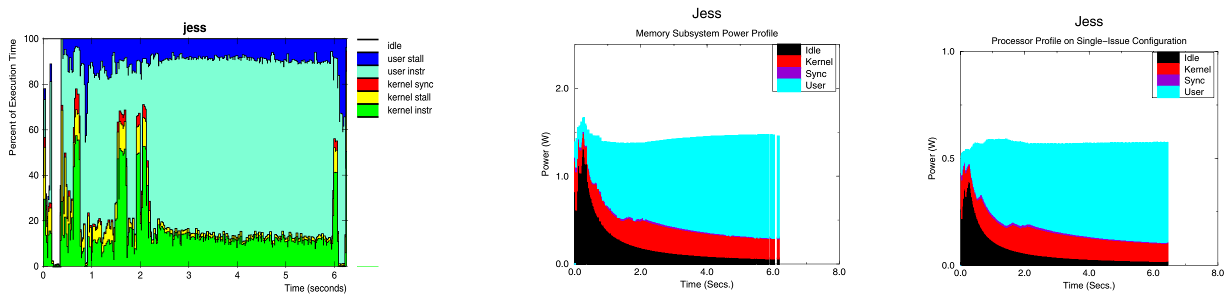


**Figure 3. Profile of Memory Subsystem Behavior**

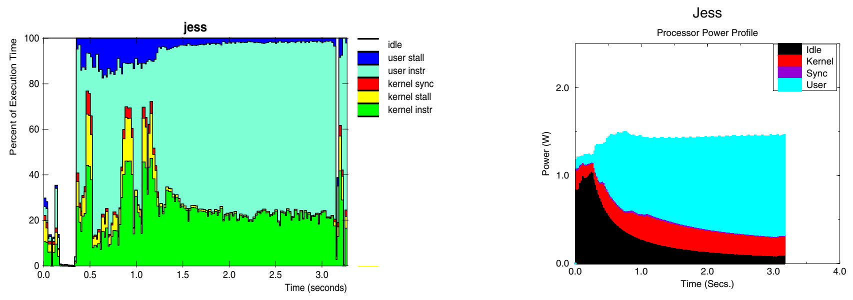


**Figure 4. Profile of Processor Behavior**

Figure 5 presents the overall power budget of the system, including the disk. Here, the breakdown is presented over all the modes of execution combined, and is averaged over all the benchmarks. The term "conventional" disk refers to a disk without any mode transitions. We assume that the disk can perform read/write, seek, or just keep spinning consuming the power as if it were in the ACTIVE mode. This model is the baseline disk configuration and gives an upper bound of its power consumption. It can be observed that, when no power-related optimizations are done, the disk is the single largest consumer of power in the system. Therefore, not considering this component when evaluating the power-consumption of a system does not give us the complete picture. However, the bulk of the power is still consumed in the processor datapath and memory system components. Therefore, these components are also an important target for power optimizations.

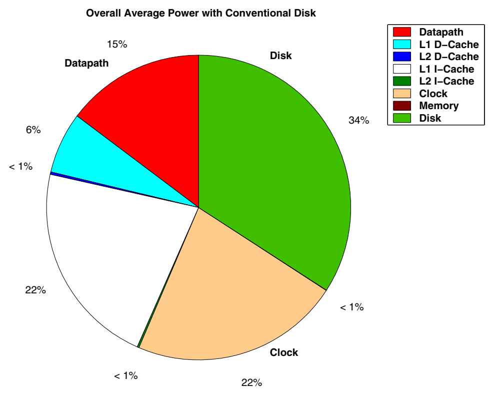


**Figure 5. Overall Power Budget with A Conventional Disk - The slices of the pie are in an anti-clockwise order of the entries in the legend. The disk contributes 34% to the average power.**

Now, we investigate the power consumption in the processor, caches, and memory. For the sake of clarity, the load-store queue, issue-window, register renaming unit, resultbus, register file, and ALUs have been clubbed together as the "datapath" in all the graphs. The graphs with the breakdown of the datapath components is given in [10]. Table 2 gives the breakdown of percentage of the cycles executed and the energy consumed for all the benchmarks. The largest fraction of the execution time is spent in the user-mode. This holds from the energy angle as well. However, the user-mode accounts for a larger proportion of the energy consumption than the fraction of the cycles. Thus, the proportion of the energy consumption for the kernel is lower than that of its execution fraction. Similarily, the idle-times consume a slightly lesser fraction of the energy than execution cycles. These variations can be explained by considering the average power (shown in figure 6). The user-mode accounts for the highest average power. The main contributor to the user-mode power consumption is the L1 I-cache. As observed from table 3, the user code has the highest number of (L1) instruction cache references per cycle. This is because, user-code exhibits higher instruction-level parallelism (ILP) compared to kernel code. Consequently, the effective fetch-width of the user code is higher than that of the kernel code. Power consumption of the data-caches is the highest for user-code, again due to its higher ILP. Similarily, the ALU-use per cycle is 0.76 for the user code which is much larger than that of the other phases (0.42 for the kernel, 0.59 for Synchronization and 0.26 for the Idle mode).

Focusing on the kernel synchronization mode, it is observed that synchronization operations are expensive in terms of power consumption. More specifically, they are found to intensely exercise the L1 I-cache and the ALUs, as compared to the other activities in the kernel mode [10]. This is because synchronization operations perform comparison and increment/decrement operations, within a tight loop. However, synchronization operations constitute less than 1% of the overall energy consumption of the benchmarks. Overall we observe that, the main power-hungry units are still the L1 caches and the clock, even when considering kernel code.

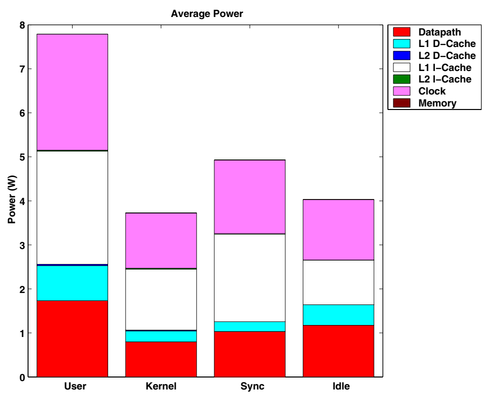


**Figure 6. Average Power (averaged over all the benchmarks)**

By including the IDLE state in the disk configuration, the dominance of the disk in the power budget decreases from 34% to 23%. This is shown in figure 7. This optimization provides significant power-savings, and also alters the overall picture. Now the L1 I-cache and the clock dominate the power profile.

## 3.3 Kernel Behavior

In this section, we give a detailed power/energy characterization of the operating system. We breakdown the kernel activity into services and compare the energy behavior of key kernel services with their performance in an attempt to answer the following questions:

- Do the services that account for the bulk of the kernel cycles also constitute the bulk of the energy consumption?
- Is their per-invocation energy consumption data-dependent? If so, how can this information be used to accelerate the simulation process?

As observed in the previous subsection, the operating system itself can be a significant consumer of energy (up to

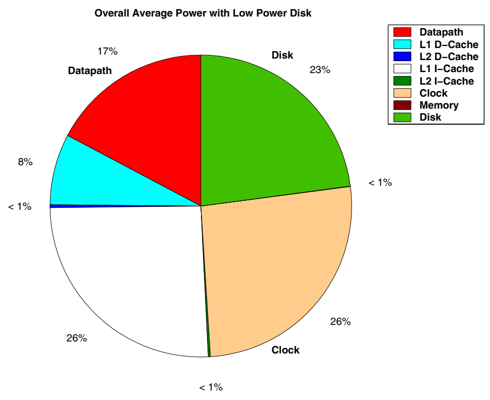


**Figure 7. Percentage Contribution of the Disk with the IDLE state to the Power Consumption - The disk contributes 23% to the average power**

17% in `jack`, for kernel instructions and synchronization operations combined). Table 4 gives the results for the services that account for the bulk of the kernel execution-time and compares the the execution cycles to their relative energy consumption contributions for each of the benchmarks. MIPS architectures have a software-managed TLB. The operating system handles the misses by doing the required address translation, reloads the TLB, and then restarts the user process. These operations are done by the *utlb* service. *demand_zero* zeroes out a newly allocated page and the *cacheflush* routine flushes the I-/D-caches. *vfault* is the validity-fault handler.

From the table, it is clear that those services that account for the bulk of the kernel execution time also account for the bulk of the energy consumption. However, the percentage of energy consumed for *utlb* is proportionately smaller compared to its execution time. As *utlb* accounts for the bulk of the operating system activity in these benchmarks, its behavior reflects on the kernel as a whole. Again, this trend can be explained by considering the average power of the services. Figure 8 shows the average power of four key kernel services. The power numbers presented have been averaged over all the invocations of the service over the entire profiled period, and then averaged over all the benchmarks.

Clearly, *utlb* has a much lower average power than the other services considered. The handler is not data-intensive, and therefore, does not exercise the data caches and the load/store queue. As these units are not accessed, the clock power is lower as well, leading to a smaller average power.

Table 5 shows how much variation there actually exists between the invocations of the services across the benchmarks. The variation is measured using the coefficient of deviation and is expressed as a percentage. The services presented can be roughly categorized as those being completely internal to the kernel (*utlb, demand_zero, cacheflush*) and those that are invoked by an user-program (*read, write, open*). The internal OS services show very small deviation in their energy behavior per invocation. On the other hand, the other externally-invoked services, which

| Benchmark | User | | Kernel Inst. | | Kernel Sync. | | Idle | |
|---|---|---|---|---|---|---|---|---|
| | Cycles | Energy | Cycles | Energy | Cycles | Energy | Cycles | Energy |
| compress | 88.24 | 93.74 | 7.95 | 4.18 | 0.2 | 0.14 | 3.61 | 1.94 |
| jess | 63.69 | 77.15 | 24.57 | 15.12 | 0.86 | 0.68 | 10.88 | 7.05 |
| db | 66.1 | 81.19 | 24.28 | 13.22 | 0.75 | 0.54 | 8.87 | 5.05 |
| javac | 64.2 | 78.47 | 27.54 | 15.98 | 0.55 | 0.44 | 7.71 | 5.11 |
| mtrt | 80.62 | 90.07 | 14.8 | 7.44 | 0.26 | 0.17 | 4.32 | 2.32 |
| jack | 69.02 | 81.36 | 27.91 | 16.43 | 0.63 | 0.51 | 2.44 | 1.7 |

**Table 2. Percentage Breakdown of Energy and Cycles**

| Benchmark | User | | Kernel Inst. | | Kernel Sync. | | Idle | |
|---|---|---|---|---|---|---|---|---|
| | iL1Ref | dL1Ref | iL1Ref | dL1Ref | iL1Ref | dL1Ref | iL1Ref | dL1Ref |
| compress | 2.0088 | 0.6833 | 1.1203 | 0.2080 | 1.5560 | 0.1745 | 0.7612 | 0.3546 |
| jess | 1.9861 | 0.6217 | 1.1143 | 0.2164 | 1.5956 | 0.1775 | 0.8267 | 0.3851 |
| db | 2.0911 | 0.6699 | 1.0602 | 0.1892 | 1.5240 | 0.1832 | 0.7244 | 0.3375 |
| javac | 1.9685 | 0.5604 | 1.0346 | 0.1835 | 1.5355 | 0.1720 | 0.8110 | 0.3778 |
| mtrt | 2.1105 | 0.6473 | 1.085 | 0.1908 | 1.5177 | 0.1697 | 0.7524 | 0.3505 |
| jack | 1.8465 | 0.5869 | 1.041 | 0.1931 | 1.5585 | 0.1708 | 0.8718 | 0.4061 |

**Table 3. Cache References Per Cycle**

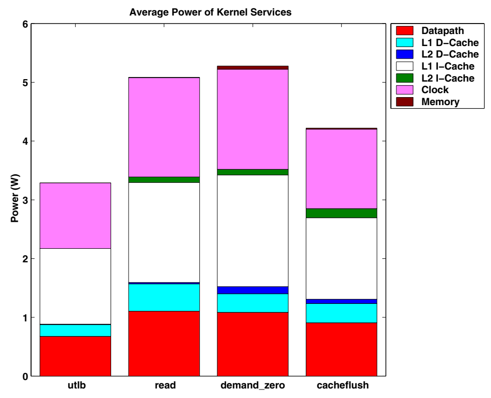


**Figure 8. Average Power of Operating System Services**

are I/O system calls, show a greater coefficient of deviation. This is because the energy consumed per invocation depends on factors such as the data transfer-size, whether the data is available in the file-cache etc. This result suggests that, given a trace of the number of invocations of the various kernel-services for a given workload, it is possible to get a rough estimate, with an error margin of about 10%, of the kernel energy consumption, without actually performing a detailed simulation. Such traces can be obtained using tools such as `prof` and `truss`, which are common in many Unix-based systems. Further, we found that, the per-cycle processor and memory-system access-behavior of the idle-process can be accurately predicted and is independent of the workload. We used this property in our disk model, where spin ups/spin downs could be simulated by just fast-forwarding the simulation by the requisite number of cycles rather than actually simulating it. This eliminates any additional cycles incurred in the simulation (and thus longer simulation time) due to the full use of the disk model.

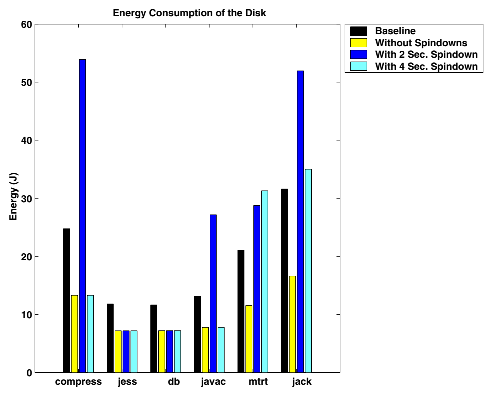
## 4 Disk Power Management

As seen in section 3, the disk is the single most power-hungry unit in the computer system. The power consumption of the disk becomes paramount for large server machines, such as web-servers and file-servers, which perform a large amount of I/O [14]. This problem is tackled by using disk power management schemes [8, 24]. Modern-day disks are designed to support a variety of low-power modes. The disk transitions to one of these modes during inactive periods. We investigated the benefit of employing such a disk using the Toshiba disk model. We considered four different disk configurations for our study, namely:

1. The baseline disk used in section 3.
2. A disk that supports the IDLE low-power mode but no STANDBY mode.
3. A disk that supports the STANDBY mode with a spin down threshold of 2 seconds, in addition to the IDLE mode.
4. A disk that supports the STANDBY mode with a spin down threshold of 4 seconds, in addition to the IDLE mode.

Disk configuration 2 models a disk where there is a transition to the IDLE mode immediately after a read/write operation completes. The disk keeps spinning while in the IDLE mode and transitions back to the ACTIVE state when a seek operation is to be performed. In disk configurations 3 and 4, the disk spins down to the STANDBY mode after a fixed period of inactivity (with respect to the disk), called the Spin down Threshold. We chose a spin down threshold of 2 seconds for configuration 3 based on the results given in [20]. Figure 9 gives the results of our study for all the benchmarks.

For this study, we use energy as the metric instead of power, as we wish to evaluate designs which depend on periods of activity rather than a single processor cycle. The



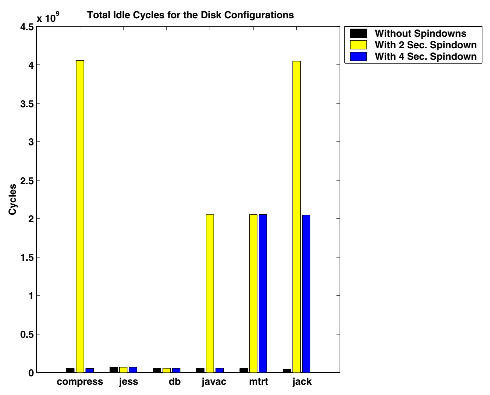


**Figure 9. Energy-Performance Tradeoffs for the Disk Configurations**

bar graphs in Figure 9 give the energy consumption of just the disk for the four configurations and the variation in the number of idle-cycles. As mentioned in section 2, the cycles due to disk activity are accounted for as idle-cycles in the execution profile.

Transitioning to the IDLE mode after a read or write provides significant energy benefit for all the benchmarks. Further, since transitions to the IDLE mode take zero time, there is no performance degradation, and an energy-benefit is always obtained. Therefore, the baseline disk configuration is not considered when comparing the configurations for their performance. `jess` and `db` are unaffected by using configuration 3 because of their short running times. For `compress`, `javac`, `mtrt`, and `jack`, there is severe energy and performance degradation. Their execution time and energy consumed (in the absence of spindowns) is a much smaller fraction to that of the spin-operations, and a large amount of energy and time is consumed to spin the disk back up. During the execution of `compress` and `jack`, the disk spins up and down multiple times, leading to a large increase in the number of idle-cycles and the overall energy consumption. This is reflected clearly in the performance graph. When we go from a spin down time of 2 seconds to 4 seconds, we see that the overall energy and performance behavior of `compress`, and `javac` becomes that of configuration 2. For `jack`, there is a 33% improvement in the energy-efficiency and there is also a significant reduction in the number of idle-cycles. This is because the longer spin down timeout value eliminates a pair of spin down and spin-up. However, it is interesting to observe that the energy consumption actually *increases* for `mtrt`, though the number of idle-cycles remains unaffected. That is because, for both configurations 3 and 4, two spindowns and spin-ups are performed. In configuration 3, the disk spins down to the STANDBY mode earlier than in configuration 4. As the STANDBY mode has a lesser power-cost than the IDLE mode, configuration 3 consumes lesser energy. From the above results, we make the following important observation: Disk spindowns should be done only if the time between consecutive disk accesses is much larger than the spin down and spin-up time.

## 5 Discussion and Conclusion

SoftWatt provides a simulation infrastructure for carrying out detailed studies about the power and performance behavior of applications. The detailed architecture and operating system simulation of SimOS coupled with the analytical power models make SoftWatt an ideal vehicle for conducting research in power-efficient design at several levels of the system. We summarize the results of our characterization study as follows:

- From a system perspective, the disk is the single largest consumer of power accounting for 34% of the system power. The adoption of the disk with low-power features shifts this power hotspot to the clock distribution and generation network and the on-chip first level instruction cache. The setting of the disk spin down threshold is critical in the shifting of this hotspot. Further, for single-issue processor configurations, we find that the memory subsystem has a higher average power than the processor core.

- Among the four different software modes, the user mode consumes the maximum power. Among the other modes, the kernel synchronization operations are expensive in terms of their power consumption. However, their contribution to overall system energy is small due to the infrequent synchronization operations when executing the Spec JVM98 benchmarks. Though the kernel mode has the least power consumption overall, due to the frequent use of kernel services, it accounts for 15% of the energy consumed in the processor and memory hierarchy. Thus, accounting for the energy consumption of the kernel code is critical for estimating the overall energy budget. This estimate is particularly important for high-performance mobile environments such as laptops.

- Among the kernel services, the *utlb* and *read* services are the major contributers to system energy. However, the frequently used *utlb* routine has a smaller power consumption as compared to *read* as it exercises fewer components. Further, the per-invocation of the kernel services is fairly constant across different applications. Thus, it is possible to estimate the energy consumed by

| Benchmark | Service | Num | % Cycles | % Energy |
|---|---|---|---|---|
| **compress** | | | | |
| | utlb | 7132786 | 76.2862 | 64.2989 |
| | read | 5863 | 9.46498 | 13.7241 |
| | demand_zero | 3080 | 4.46058 | 6.91512 |
| | cacheflush | 1558 | 1.33649 | 1.39134 |
| | open | 192 | 1.04054 | 1.18379 |
| | vfault | 972 | 0.84626 | 1.12367 |
| | write | 71 | 0.82243 | 0.0.74204 |
| | tlb_miss | 12209 | 0.716817 | 0.917478 |
| **jess** | | | | |
| | utlb | 8351936 | 64.8216 | 53.7089 |
| | read | 14902 | 16.5106 | 20.7921 |
| | BSD | 18066 | 4.15149 | 5.53606 |
| | demand_zero | 2585 | 3.20818 | 4.19697 |
| | tlb_miss | 92554 | 2.93511 | 4.329 |
| | open | 327 | 1.4382 | 1.63077 |
| | cacheflush | 2371 | 1.42624 | 1.52855 |
| | vfault | 1017 | 0.638494 | 0.826016 |
| **db** | | | | |
| | utlb | 9311336 | 75.6565 | 66.6431 |
| | read | 6289 | 7.04481 | 10.1373 |
| | write | 698 | 5.12059 | 5.22395 |
| | demand_zero | 2172 | 2.57247 | 3.86259 |
| | tlb_miss | 53764 | 1.75243 | 2.82191 |
| | du_poll | 4066 | 1.08423 | 1.22557 |
| | cacheflush | 1540 | 0.981458 | 1.10068 |
| | open | 188 | 0.76878 | 0.913507 |
| **javac** | | | | |
| | utlb | 12815956 | 78.782 | 71.6722 |
| | read | 6205 | 5.47241 | 7.96247 |
| | demand_zero | 3402 | 3.70849 | 4.86183 |
| | tlb_miss | 134265 | 3.33207 | 5.51917 |
| | open | 434 | 1.58547 | 2.09804 |
| | cacheflush | 2802 | 1.33713 | 1.65195 |
| | xstat | 142 | 0.627263 | 0.879387 |
| | vfault | 1054 | 0.517107 | 0.739405 |
| **mtrt** | | | | |
| | utlb | 11871047 | 81.3054 | 72.199 |
| | read | 6400 | 6.35944 | 8.87615 |
| | demand_zero | 2868 | 3.23787 | 4.40053 |
| | tlb_miss | 84966 | 2.43972 | 3.65625 |
| | cacheflush | 1681 | 0.929139 | 1.03098 |
| | open | 210 | 0.739026 | 0.880839 |
| | write | 88 | 0.623178 | 0.582169 |
| | vfault | 1039 | 0.57036 | 0.792793 |
| **jack** | | | | |
| | utlb | 30131127 | 71.0119 | 64.0483 |
| | read | 40079 | 16.7512 | 18.9097 |
| | BSD | 68612 | 6.6143 | 7.36693 |
| | tlb_miss | 204529 | 1.8767 | 3.03969 |
| | demand_zero | 3484 | 1.43321 | 1.88598 |
| | cacheflush | 2039 | 0.386741 | 0.44586 |
| | open | 239 | 0.292891 | 0.35692 |
| | clock | 963 | 0.265881 | 0.235892 |

**Table 4. Breakdown of Kernel Computation by Service - Cycles vs. Energy**

| | Energy Per Invocation | |
|---|---|---|
| **Service** | **Mean** | **Coefficient of Deviation (%)** |
| utlb | $2.1276 \times 10^{-07}$ | 0.13971 |
| demand_zero | $5.408 \times 10^{-05}$ | 1.4927 |
| cacheflush | $2.1606 \times 10^{-05}$ | 2.4698 |
| read | $4.8894 \times 10^{-05}$ | 6.615 |
| write | 0.00025351 | 10.6632 |
| open | 0.00015586 | 10.0714 |

**Table 5. Variation in Behavior of Operating System Services**

kernel code with an error margin of about 10% without detailed energy simulation.

- Whenever the operating system does not have any process to run, it schedules the idle-process. Though this has no performance implications, over 5% of the system energy is consumed during this period. This energy consumption can be reduced by transitioning the CPU and the memory-subsystem to a low-power mode or by even halting the processor, instead of executing the idle-process.

While the simulation results in this work were presented using the Spec JVM98 benchmarks, this tool will be invaluable in analyzing other workloads such as database workloads. The characterization and energy optimization of such workloads is becoming a major issue as power-hungry web hosters can consume as much as 10 to 30 Megawatts. We also plan to investigate acceleration of energy simulation using fast forward techniques without significant loss in accuracy.

## Acknowledgements

This research has been supported in part by several NSF grants: 0073419, 0093082, 0093085, 0103583, 0097998, 9701475, 9988164, Sun Microsystems, Gigascale Silicon Research Center, and equipment grant EIA-9818327.

## References

[1] Avant! Star-Hspice. http://www.avanticorp.com/products.

[2] K. Baynes, C. Collins, E. Fiterman, C. Smit, T. Zhang, and B. Jacob. The performance and Energy Consumption of Embedded Real-Time Operating Systems. Technical Report UMD-SCA-TR-2000-04, University of Maryland, November 2000.

[3] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA-7)*, January 2001.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000.

[5] A. Chandrakasan, W. J. Bowhill, and F. Fox. *Design of High-Performance Microprocessor Circuits*. IEEE Press, 2001.

[6] T. L. Cignetti, K. Komarov, and C. S. Ellis. Energy Estimation Tools for the Palm. In *Proceedings of ACM MSWiM 2000: Modeling, Analysis and Simulation of Wireless and Mobile Systems*, August 2000.

[7] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. DRAM Energy Management Using Software and Hardware Directed Power Mode Control. In *Proceedings of the 7th International Conference on High Performance Computer Architecture*, January 2001.

[8] F. Douglis and P. Krishnan. Adaptive disk spin-down policies for mobile computers. *Computing Systems*, 8(4):381–413, 1995.

[9] D. Duarte, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir. Formulation and Validation of an Energy Dissipation Model for the Clock Generation Circuitry and Distribution Networks. In *Proceedings of the 2001 VLSI Design Conference*, 2001.

[10] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. Technical Report CSE-01-029, The Pennsylvania State University, November 2001.

[11] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, E. Bugnion, and M. S. Lam. Maximizing multiprocessor performance with the SUIF compiler. *IEEE Computer*, 29(12):84–89, 1996.

[12] S. A. Herrod. *Using Complete Machine Simulation to Understand Computer System Behavior*. PhD thesis, Stanford University, February 1998.

[13] M. Irwin, M. Kandemir, N. Vijaykrishnan, and A. Sivasubramaniam. A Holistic Approach to System Level Energy Optimization. In *Proceedings of the International Workshop on Power and Timing Modeling, Optimization, and Simulation*, September 2000.

[14] J. Jones and B. Fonseca. Energy Crisis Pinches Hosting Vendors. http://iwsun4.infoworld.com/articles/hn/xml/01/01/08/010108hnpower.xml.

[15] R. Joseph, D. Brooks, and M. Martonosi. Runtime Power Measurements as a Foundation for Evaluating Power/Performance Tradeoffs. In *Proceedings of the Workshop on Complexity Effectice Design*, June 2001.

[16] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and A. Sivasubramaniam. vEC: Virtual Energy Counters. In *Proceedings of the ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'01)*, June 2001.

[17] M. B. Kamble and K. Ghose. Analytical Energy Dissipation Models for Low Power Caches. In *Proceedings of the International Symposium on Low-Power Electronic Design*, pages 143–148, August 1997.

[18] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, and A. Sangiovanni-Vincentelli. Efficient Power Estimation Techniques for HW/SW Systems. In *Proceedings of IEEE Volta*, 1999.

[19] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power Aware Page Allocation. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, November 2000.

[20] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. Quantitative Analysis of Disk Drive Power Management in Portable Computers. Technical Report CSD-93-779, University of California, Berkeley, 1994.

[21] T. Li, L. K. John, N. Vijaykrishnan, A. Sivasubramaniam, J. Sabarinathan, and A. Murthy. Using Complete System Simulation to Characterize SPECjvm98 Benchmarks. In *Proceedings of the International Conference on Supercomputing (ICS) 2000*, May 2000.

[22] J. R. Lorch. A complete picture of the energy consumption of a portable computer. Master's thesis, University of California, Berkeley, December 1995.

[23] J. R. Lorch and A. J. Smith. Scheduling Techniques for Reducing Processor Energy Use in MacOS. *Wireless Networks*, 3(5):311–324, October 1997.

[24] Y.-H. Lu and G. D. Micheli. Adaptive hard disk power management on personal computers. In *Proceedings of the IEEE Great Lakes Symposium*, March 1999.

[25] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In *Proceedings of the 24th International Symposium on Computer Architecture*, 1997.

[26] R. P.Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha. Power Analysis of Embedded Operating Systems. In *Proceedings of the 37th Conference on Design Automation*, pages 312–315, 2000.

[27] R10000 Microprocessor User's Manual. http://www.sgi.com/processors/r10k/manual/t5.ver.2.0.book_4.html.

[28] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel and Distributed Technology: Systems and Applications*, 3(4):34–43, 1995.

[29] T. Simunic, L. Benini, and G. D. Micheli. Cycle-Accurate Simulation of Energy Consumption in Embedded Systems. In *Proceedings of the Design Automation Conference*, June 1999.

[30] A. Sinha, A. Wang, and A. Chandrakasan. Algorithmic Transforms for Efficient Energy Scalable Computation. In *Proceedings of the IEEE International Symposium on Low-Power Electronic Design (ISLPED' 00)*, August 2000.

[31] Spec JVM98 Benchmark Suite. http://www.spec.org/osg/jvm98/.

[32] Synopsys Power Compiler. http://www.synopsis.com/products/power/power.html.

[33] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing Power in High-Performance Microprocessors. In *Proceedings of the Design Automation Conference*, June 1998.

[34] Toshiba Storage Devices Division. http://www.toshiba.com/.

[35] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin. The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool. In *Proceedings of the Design Automation Conference (DAC)*, June 2000.

[36] K. C. Yeager. The MIPS R10000 Superscalar Microprocessor. *IEEE Micro*, 16(2):28–40, April 1996.