Chapter 1

Power Measuring and Profiling: State-of-the-Art

Hui Chen and Weisong Shi

Department of Computer Science
Wayne State University
Detroit, MI, USA
Email:huichen, weisong@wayne.edu

As the commitment to reduce environmental impact are becoming important objectives for our society. The energy efficiency of computer systems became the most valuable research topic for researchers, architects, system designers, software developers and so forth. A lot of work has been done to decrease power dissipation and increase energy effi-

ciency. For example, DVFS technique was designed to save energy when the workload of the system is low. In addition, some researchers tried to control energy consumption in operating system through managing energy as one kind of resource. Among all the area of power-aware system research, power profiling is very important. It can not only be used for power-performance evaluation, software power dissipation analysis, but also can be used to trigger power-aware mechanisms and evaluating the effectiveness of these mechanisms.

Power profiling includes hardware-based method and software-based method. Hardware-based power profiling mainly uses different kinds of instruments to measure the power of a device directly. The result is much more accurate than software-based profiling. If is usually used to evaluate the effectiveness of power saving techniques. However, hardware-based method can is limited to measure component level power profiles. Even though not as accurate as hardware-based method, software-based power profiling tries to estimate the power of different levels by designing a group of power models. In this survey, we first describe the valuable power profiling techniques, and then we use several case studies to show the usage of these techniques.

1.1 Introduction

In the late 1990s, power, energy consumption and power density have become the limiting factors not only for the system design of portable and mobile devices, but also for high-end systems [45]. The design of the computer system had changed from the performance-centric stage to the power-aware stage.

Accompanied by the increasing of the integration of CMOS (complementary metal oxide semiconductor) circuits and the clock frequency, the power density of the hardware circuits increased quickly. To cool down the processor with a cost-effective method may be challenging, if the power density continues to increase. This problem makes power became one of the first-class system design considerations, although power is not a new problem for computer system design, and performance is not the only objective any more. Making performance/power tradeoffs become very important when we design the hardware architecture of computer systems.

As the trend of personal computer, portable computers and mobile devices require long lasting batteries, otherwise the user experience will be bad. To extend the battery lifetime, we not only need to develop new battery techniques, but also need to improve the energy efficiency of these devices. The improving of energy efficiency of mobile devices requires both hardware methods and software methods. The hardware methods aim to design low-power circuits and support dynamic power management strategies, and the software methods tries to make power/performance tradeoffs with the APIs supplied by hardware.

Finally, accompanied by the emerging of the Internet, a lot of large data centers are built by those big companies to supply stable, good quality service for the customers. Even when the workload is low, these servers still have very large power dissipation, because they are designed for the peak workload. They need to keep these servers always on, even though most of the time the workload is trivial. Furthermore, about the similar amount of cost as used for energy consumption needs to spend on cooling down the data center. These reasons cause the energy unproportional problem in data centers [4].

1.1.1 Terminologies

Before describing more details about power measuring and profiling, we first introduce some basic terminologies.

- *Energy*: In computer systems, energy, in joule, is the electricity resource that can power the hardware devices to do computation. Energy is more used for the research of mobile platform and data centers. For mobile device, energy is strongly related with the battery lifetime. For data center, which consumes a large amount of energy, energy is used as the concern of electricity costs. Usually, researches in these areas use energy efficiency, such as PUE (power usage effectiveness), as the metric to evaluate their work.

- *Power*: Power is the dissipate rate of energy. The unit of power is watts (W), which means joules per second. Originally, this metric is used to reflect the current delivery and voltage regulator of the circuits. In system research, power may also be used for the abstract concepts, such as process and operating system. For example, we may say that the power of a process is 1W. This means that the execution of this process causes the hardware circuits dissipate 1W of power.

- *Power Density*: Power density means the amount of power dissipated per unit areas. This metric represents the heat problem of on the processor die, because power has the direct relationship with heat.

- *Static Power*: Static power, which is mainly leakage power, is the power caused by the incomplete turning off of transistors. This terminology denotes the basic power that a hardware device needed when it is not active.

- *Dynamic Power*: Dynamic power is the power caused by the switching of the capacitance voltage states. This terminology denotes the extra part of power needed to make the hardware device work.

- *Idle Power*: Idle power means the power of the full system when the system is in the idle state. This terminology is used for system-level research, because the hardware devices are not really inactive, even when the system is idle. Idle power includes the dynamic power of the system hardware and a part of dynamic power caused by some system processes.
- *Active Power*: Active power is the extra power dissipated by the system when the it is doing the computation.

1.1.2 Power-aware System Design

In the early age, researchers try to decrease the power and eliminate the waste of energy during the architecture design stage, because the power problem obstructs the development of computer systems. These techniques include multi-core on chip processor (CMP), core independent functional units, dynamic frequency and voltage scaling and clock gating. Most of these techniques are designed to decrease the energy consumption of the processor, which is the dominant energy consumption of a typical computer system. In addition, some other techniques, such as the phase-change memory, and solid-state disk driver, are also proposed to decrease the power dissipation of other devices.

Although hardware based power management techniques have been proven as useful for reducing the energy consumption of the computer system, and they will be continuing important in the future. However, researchers point out that these techniques alone are not enough, and higher-level strategies for reducing energy consumption will be increasingly critical [11]. Current operating system is not designed by considering the power problem, thus, even if the system is idle it still consumes a large amount of energy, which is sawn as waste and not used for computing. Vahdat *et al.* argues that traditional operating system design should be revisited for energy efficient [53]. H. Zeng *et al.* present EcoSystem [61], which tries to management power as one kind of system resources. A most recent power-aware operating system called Cinder [49], which is designed for mobile devices. Except for designing new operating systems, some other high-level power-aware strategies, such as power-aware scheduling [43], are also globally researched. Moreover, power analysis is equally important when design softwares. Some scholars even proposed energy complexity [31] as one of the metrics for algorithm design. Most recent research has already targeted on power-aware programming. For example, Hönig —em et al designed a system, called SEEP [26], to find the base energy demand of the programs. Bhattacharya *et al.* analyzed the energy waste caused by “Software Bloat” [7].

1.1.3 Power Measuring & Profiling

The power measuring and profiling are a new area that arising in parallel with the power problem. As energy consumption becomes one of the foremost considerations when designing computer systems, power profiling become a key issue in the community of computer systems. As the basis of the power-aware system research, power profiling are not only needed to evaluate power optimization techniques and to make power/performance trade-off, but also important to supply critical power information for operating systems and power-aware software.

The early stage publications [11, 39, 55, 58] are mainly based on simulation because they are used during the hardware design stage to make power/performance tradeoffs. Most of them estimate the power consumption of the hardware circuits based on the classical power model shown as Equation 1.1 and Equation 1.2. Here, C is the load capacitance, V_{dd}^2 is the supply voltage, α is the activity ratio of the circuits, f is the clock frequency and $I_{leakage}$ is the leakage current of the circuits. The content of simulation methods is beyond the content of this survey.

$$P_{dynamic} = CV_{dd}^2\alpha f \tag{1.1}$$

$$P_{static} = I_{leakage}V_{dd}^2 \tag{1.2}$$

Even though, these circuit-level power models are good to estimate the circuits' power, they cannot be used to estimate the power of the real products. To understand the power dissipation of the real systems become critical, thus a lot of works use instruments to measure the power of these devices directly. The measured power is more accurate than estimated power, accordingly it is used by many articles to analyze the power problems of the computer system [32, 35, 40], to validate the effect of the power-aware strategies, and to build software power models with linear regression method [5, 28, 33]. While the measured power is accurate, these methods also have two limitations. First, low level power can hardly be measured. Second, the measured power cannot be used by power aware strategies or the cost is too high.

To cope with the limitations of direct measurement, the following research is trying to use software method to estimate the power on the online system. These works are done from different levels: component level [9, 14, 19, 12], core level [6] ,CPU functional unit level [28], process level [18] and virtual machine level [36]. These articles are either use operating system events or hardware performance counter events to build their power model.

Basically, we can classify power measuring and profiling techniques into three categories: simulation approach, hardware measurement, and software power profiling. In the following section, we mainly describe the last two approaches.

1.2 Hardware-based Power Measurement

Hardware-based power measurement is the methods that use instruments to measure the current or voltage of the hardware devices and further use these measured values to compute the power of the measured object. The instruments used to do the measurement includes different types of meters, special hardware devices that can be embedded into the hardware platforms, and power sensors that designed within the hardware. Normally, these methods can only measure the component level power, because the highly integration of the hardware circuits makes the lower level functional units become difficult to measure. Most publications [28, 32] make use of micro-benchmarks, which stress one or several special functional unit, to isolate lower level power.

1.2.1 Power Measurement with Meters

Direct power measurement with meters is a straight forward method to understand the power dissipation of devices and the full system. Many articles [10, 28] rely on power meters to measure the real power and use it to validate there research work or to make analyzation. Moreover, some works [22] measure the hardware components' power and breakdown it into lower levels based on some indicators that could reflect the activity of these lower level units. The differences of these methods are which type of meters is used to do the measurement and at which place the measurement is done.

One type of the globally used meters is the digital multimeter, which is easy to use. Generally, these meters can sample the measured object each second. The result can be collected with the serial port that is connected to the data collection system. Using this method we need to disconnect the wires that we want to measure and connect a small resister (normally less than 0.5Ω). Finally, we measure the voltage of the resistor, and compute the power on this wire. Figure 1.1 shows an example of this method. Joseph *et al.* use multimeters to measure the power while executing different benchmarks and make power/performance tradeoffs [32]. Using this method, Feng *et al.* measured the node level and component level power for a node of the distributed system [20].

Another kind of meters that many people use is clamp meter, which can measure the current without disconnect the wire. Usually a clamp meter has larger measurement range than multimeter, thus they can be used to measure the power of systems that current is much higher. The connection of the measurement platform of clamp meter is similar to the multimeter. In [35], the authors adopt the direct power measurement method with clamp meters to measure the power on a Cray XT4 supercomputer under several HPC (High Performance Computer) workloads. Their results show that, computation-

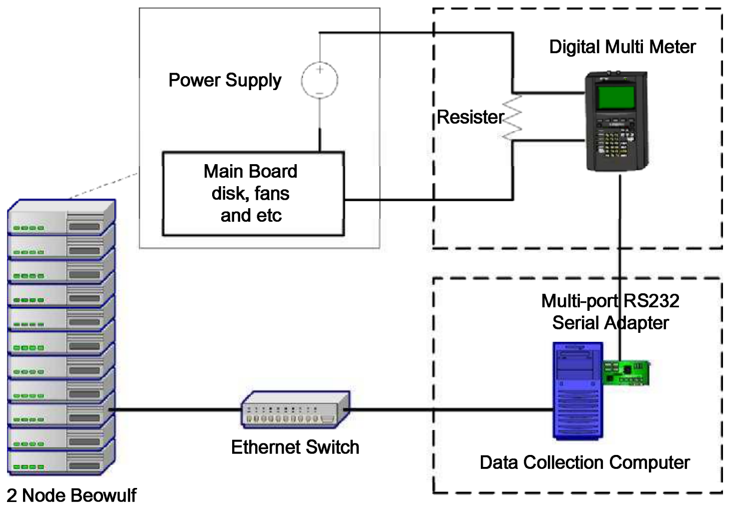


FIGURE 1.1: An example of using multimeter to measure the power. (source [20])

intensive benchmarks generate the highest power dissipation while memory-intensive benchmarks yield the lowest power usage.

The multimeter and clamp meter are mainly used to measure the DC power by connecting them between the power supply and the measured component. In addition, we also use one kind of power meter, such as "Watts UP" [2], to measure the AC power. This kind of meters can only measure the system level power, because only power supplies are powered by AC. AC power is good for understanding the total system power characters, but improper for lower-level analysis, because the transform efficiency is not constant during the measurement.

1.2.2 Power Measurement with Special Designed Devices

While direct measuring with meters is simple, it does not supply methods to control the process of the measurement process. For example, to synchronize the measured power to the monitoring of the performance metrics. Thus, some special designed power measurement devices are presented to measure the power in these circumstances. One of the early works is Itsy [54], which is used to measure the power of mobile devices. The platform is integrated between the power supply and the measured mobile device as shown in Figure 1.2. This framework could not only measure the power but also could trigger the measured the devices (Section 1.3.2 will describe an example of using this platform).

PLEB [51] is a single board on computer that is design with a set of current

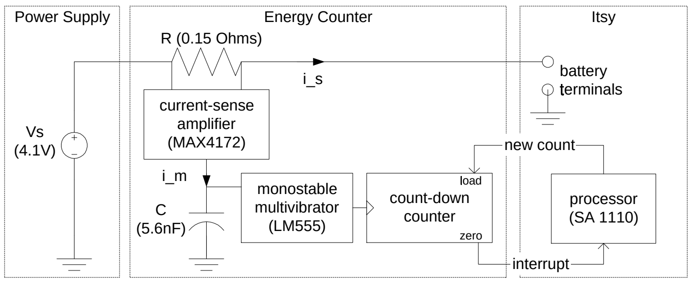


FIGURE 1.2: The Itsy energy-driven statistical sampling prototype. (source [20])

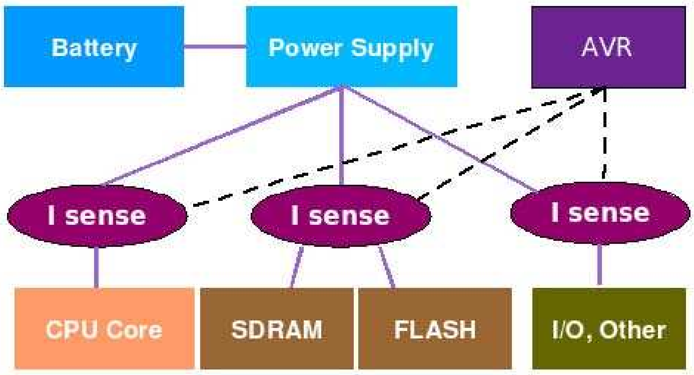


FIGURE 1.3: The structure of PLEB. (source [51])

sensors on-board. Furthermore, the micro-controller of this device is integrated with an analogue-to-digital converter to read the sensors. This platform can be used to isolate the power of processor, memory, flash driver and I/O devices. Figure 1.3 shows the structure of PLEB.

1.2.3 Integrating Sensors into Hardware

The last type of approach is mainly used by high-performance servers. In the past several years, the majority of servers are designed contains service processor [23, 25], which is a hardware and software integrated platform that works independent of the main processor and the server's operating system. The hardware of the service processor may either embedded on the motherboard or on a plug-in card. Most of the hardware of the service processor includes power sensors to monitor the power, that are supplied to the administrator for power management. For example, Intel's service processor Intelligent

Platform Management Interface (IPMI) [27] supports APIs to read the power information monitored by the sensors.

Other techniques [1, 52], which are designed to improve the energy efficiency of data centers, that are used on servers integrate power sensors in deeper level. IBM BladeCenter and System x^{TM} servers supply PowerExecutive solutions, which enables customers to monitor actual power draw and temperature loading information [1].

Though on-line power-aware applications can use this method, it is difficult to yield low-level power information, in which case, hardware circuits are too complicated to distinguish the originality of power dissipation. In addition, power monitoring circuits also dissipate a large amount of power as well.

1.3 Software-based Power Profiling

Although the hardware approach can supply very accurate power information, the problems we listed in the last section limit its application range. Software-based approach, however, can be used to supply more fine-grained online power information, which could be used by power-aware strategies. The live power information of systems is highly needed for designing high-level energy efficient strategies. For example, Ecosystem [61] and [41], which propose the concept that managing system energy as a type of resource, requires the support of real-time power information on different levels. As part of the energy-centric operating system, energy profiles are also needed by new power-aware scheduling algorithms [3, 37]. Furthermore, compared with hardware measurement, software-based method is much more flexible. Usually, we can apply them to different platforms without changing the hardwares.

1.3.1 Power Model

Software-based approach usually builds power models to estimate the power dissipation of different levels: instruction level, program block level, process level, hardware component level, system level and so forth. These methods first try to find the power indicators that could reflect the power of these software or hardware unit. Then they build the power model with these power indicators and fine tune the parameters of the power model. Finally, they verify the accuracy of the power models by comparing with the result measured by hardwares or applying the power information into a power-aware strategy to test its usability. Based on the difference of the power indicators, we categorize these methods into two categories: system profile-based method and hardware performance counter (PMC) based method.

The metrics of the power model may varied for each type of usage. Here, we list several important metrics we used to evaluate a power model.

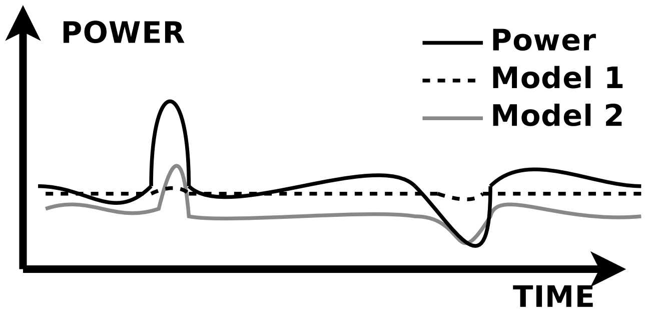


FIGURE 1.4: An example show the responsiveness of the power model. (source [6])

- **Accuracy:** The accuracy of the power model defines how accurate the estimated power is relative to the measured power. For many applications accuracy is the foremost requirement. For example, power/performance tradeoff rely on the accurate power estimation to analyze, or the result is meaningless.
- **Simplicity:** The simplicity of the power model is also important in some circumstances, such as supply real-time power information on mobile platforms. In these circumstances, if the power model is too complicated, the overhead of the power estimation will be too high to be able to used. This is usually because the power monitor needs to collect the events that will be used by the power model. Based on our experiment, accuracy is usually contradicted to simplicity, because a more accurate power model is usually relying on a more complicated power model and the sampling rate is also high.
- **Responsiveness:** The responsiveness of the power model means whether it can reflect the variation of the power, as shown in Figure 1.4. In this example, the responsiveness of Model 2 is better than Model 1. Even though Model 1 seems more accurate. However, if we use this model for the power-aware policies, the chance of trigger these policies will be lost.
- **Granularity:** Granularity means to which level that the power model can estimate. For example, a power model that can estimate the main functional units of CPU has higher granularity than another power model that can only estimate the CPU power. Usually, high granularity means high accuracy and poor simplicity.

Different with the power model that used for simulation, the power models we described in this section are used to supply online (realtime) power information. Furthermore, these power models mainly use system profile or hardware performance counters. These publications are more concentrated on estimating the power of hardware components or software modules, for which the energy efficiency could be improved by using some energy aware strategies with the power information estimated. In the following sections, we summarize these power models based on the information that the power model used.

1.3.1.1 System Profile-based Power Model

System profile or system events are a set performance statistical information supplied by the operating system. These events reflect the current state of the hardwares and softwares, including the operating system. For example, CPU utilization is the performance metric that can reflect the current workload of the processor. Nearly all the operating systems support these system events: Linux saves these values under the "proc" directory and Windows supply a set of APIs, called performance counter, to access these values. Here the performance counter is different with the hardware performance counters that will be talked about in the next section. A lot of works use them to build the power models for the hardware component, process and even program blocks, because these system events directly related with the usage of the hardware component and softwares.

Operating system, which is constituted by a set of system processes, consumes a large amount of power even when the system is idle, and it is the main reason that causes the unproportional of system usage and power dissipation. Several articles [41, 53, 61] review the traditional operating system design with energy as one of the foremost important considerations. Thus, understanding the power dissipation of these system processes is very important for the energy proportional operating system design. In [38], Li *et al.* estimate the power dissipation caused by the operating system. First, they find the power behaviors of three types of OS routines: interrupts, process and interprocess control, file system and miscellaneous services. These OS routines have different power behaviors. Some of them generate constant power dissipation and some others, such as process scheduling and file I/O, show higher stand deviation because they are largely dependent on system status. However, they find that the power of these OS routines have a linear relationship with instructions executed per cycle (IPC). They build the power model based on IPC as shown of equation 1.3. In this equation k_1 and k_0 are constants that get from linear regression step. Finally, they define the routine level based operating system power model as shown of equation 1.4.

$$P = k_1 \times IPC + k_0 \tag{1.3}$$

$$E_{OS} = \sum_i P_{OS\_routine,i} \times T_{OS\_routine,i} \tag{1.4}$$

In [18], Thanh Do *et al.* build process-level energy models. First, they build energy models for three main components: CPU, memory, and wireless network interface card (WNIC). The CPU energy model is based on system events such as the active time of the CPU, the time that the CPU worked on each frequency and frequency transition times. They assume the linear relationship between CPU frequency and power. This equation is shown as Equation 1.5, in which P_j denotes that power of the CPU when the frequency is j, n_k denotes the amount of times that transition k occurs and E_k is the corresponding energy consumption. The energy consumed by disk and WNIC are computed with the amount of data that operated by these devices. These two energy models are shown in Equation 1.6 and Equation 1.7. The disk read power (P_{read}), disk write power (P_{write}), WNIC read power (P_{send}) and WNIC write power (P_{read}) are assumed as a constant value. The energy of a process is calculated with the hardware resource used by each device in the last time interval, shown in 1.8. Here, U_{ij} is the usage of the application i on resource j, $E_{resourcej}$ is the amount of energy consumed by resource j, and $E_{interaction}$ is the amount of energy consumed by the application because of the interaction among system resources, in the time interval t. The energy profiling module is running at kernel space. This tool supply several APIs that other user space software can use, which returns the energy assumption of a process in a specified time interval.

$$E_{CPU} = \sum_i P_j t_j + \sum_i n_k T_k \tag{1.5}$$

$$E_{DISKi} = t_{readi} P_{read} + t_{writei} P_{write} \tag{1.6}$$

$$E_{NETi} = t_{sendi} P_{send} + t_{recvi} P_{recv} \tag{1.7}$$

$$E_{APPi} = \sum U_{ij} E_{resourcej} + E_{interaction} \tag{1.8}$$

Kansal *et al.* build the energy model for three main components, CPU, memory and disk, then they break down the energy into the virtual machine level based on the utilization of each component [36]. The CPU energy model they proposed is based on CPU utilization; the memory energy model used the number of LLC (last level cache) misses; and the disk energy model relies on the bytes of data that disk reads and writes. The following is these three energy models:

$$E_{cpu} = \alpha_{cpu} \mu_{cpu} + \gamma_{cpu} \tag{1.9}$$

$$E_{mem}(T) = \alpha_{mem} N_{LLCM}(T) + \gamma_{mem} \tag{1.10}$$

$$E_{disk} = \alpha_{rb} b_R + \alpha_{wb} b_W + \gamma_{disk} \tag{1.11}$$

In these three equations, μ_{cpu}, $N_{LLCM}(T)$, b_W and b_R, denotes CPU utilization, the number of LLC miss in time T, the amount of bytes write into the disk and the amount of bytes read from disk. α and γ parameters are constants that they get when training the energy model. The last two methods, even though build simple power models, only generate very low overhead, thus can be used on real systems to provide on-line energy information. Similar to Kansal *et al.*, Dhiman *et al.* also propose an on-line power prediction system for virtualized environments [17] . Instead of using linear regression, they utilize a Gaussian Mixture vector quantization based training and classification to find the correlation of measured events and the power.

SoftWatt, which models the CPU, memory hierarchy, and the low-power disk subsystem, is described in [24]. This tool is able to identify the power hot-spots in the system components as well as the power-hungry operating system services. Zedlewski *et al.* present Dempsy, a disk simulation environment that includes the accurate modeling of the disk power dissipation [59]. Dempsey attempts to accurately estimate the power of a specific disk stage, which includes seeking, rotation, reading, writing, and idle-periods, with a fine-grained model. Molaro *et al.* also analyze the possibility to create a disk driver power model based on the disk status stages [44].

1.3.1.2 PMC-based Power Model

Hardware performance counters are a group of special registers, which are designed to store the counts of hardware-related activities within computer systems. Compared with the system events supplied by the operating system or special softwares, hardware performance counters provide low-overhead access to a wealth of detailed performance information related to CPU's functional units, main memory and other components. These hardware performance events, such as L1/L2 cache miss times, could reflect the hardware activities. Thus, it is suitable for building the power models of these hardware components.

Even though a lot of works use PMCs to establish the power model, the basic process of these methods is about the same. Here is the general steps of generating the power model with PMC events.

1. Classify the device into several main functional units.

2. Choose a group of hardware performance events that may be related with the power dissipation of the device that we want to build the power model.

3. Collect the power of the device while different micro-benchmarks are executed. These micro-benchmarks stress one or several functional units of the device. In parallel with this process, we monitor all the performance counters we choose at the first step. Some early stage processes only supply a limited number of PMCs, thus we may need to repeat this process by run the same micro-benchmark monitor different PMC events.

4. Analyze the relationship of each PMC event we chose and the measured power with linear regression. This step finds correlation of PMC events and the power of each functional unit.

5. Select the PMC events that are most related with the functional unit's power to construct the unit-level power model. Usually, the power model of each unit is the product of a constant parameter and a value that reflect the active ratio of this unit, which is converted by the value of the PMC event. Some works omit the transfer and use the value of the PMC event directly.

6. Construct the device's power model by summarize each functional unit's power model and the static power of this device.

7. Find the value of the parameters in the model and exercise it with some micro-benchmarks.

To our knowledge, Bellosa, firstly, proposed the idea of using PMCs to estimate the power [5]. They run specific calibrate softwares, which stress one or several functional units, and use performance counters to monitor four types of hardware events, which are integer operations, floating-point operations, second-level address strobes and memory transactions, individually. Then they analyze the correlation of these events with the power of the component and find that these four events are tightly with the functional units that they stressed with the calibration softwares. Because the platform, a Pentium II 350 PC, they used only support two performance counters, they can only use sampling method to estimate the power of each thread and the system. Similar to Bellosa, Joseph *et al.* verify the correlation of more than ten CPU functional units and use several performance events that are most related with the CPU power to modeling the power of Pentium Pro processor [33].

In [16], Contreras *et al.* build power models for the Intel PXA255 processor and memory. The processor power model uses four hardware events, instructions executed, data dependencies, instruction cache miss and TLB (translation lookaside buffer) misses, and assume these counter values have the linear relationship with the processor power. They build the memory power model with three counters: Instruction Cache Miss, Data Cache Miss and Number of Data Dependencies. Bircher *et al.* find that the events have higher correlation with power are all IPC related [8].

Different with previous work that ignore the power dissipated by the small sub-units of processor, or consider it as static power, Isci *et al* build a more complicated power model for a Pentium 4 processor by considering much more CPU functional units [28]. We can not neglect these part of energy because it may accounts for nearly about 24% percent of the total CPU power [33]. Another difference with previous work is that they choose 22 fine granularity physical components that can be identifiable on a P4 die photo. For each of these physical components, they use an event or a group of events that can reflect the access amount of this physical component. For example, they use

IOQ Allocation, which count all bus transactions (reads, writes and prefetches) that are allocated in the IO Queue, and FSB (front side bus) data activity, which count all DataReaDY and DataBuSY events on the front side bus, to compute the bus control access rates. In [29], they list the equations that compute all the 22 physical components' access rate. They they estimate each physical component power use equation 1.12 and estimate the total power of CPU with equation 1.13.

$$\begin{aligned} Power(P_i) = AccessRate(C_i) \\ .ArchitectualScaling(C_i) \\ .MaxPower(C_i) \\ +NonGatedClockPower(C_i) \end{aligned} \tag{1.12}$$

$$TotalPower = \sum_{i=1}^{22} Power(C_i) + IdlePower \tag{1.13}$$

In equation 1.12, $AccessRateC_i$, $ArchitectualScaling(C_i)$, $MaxPower(C_i)$ and $NonGatedClockPower(C_i)$ denote the access rate, a scaling strategy that is based on micro-architectural and structural properties, the maximum power and the conditional clock power of physic component C_i. Here the maximum power and the conditional clock power are estimated based empirical value. For example, the initial maximum power are estimated based on area of this unit on the die. This estimation is reasonable because the area are related with the number of CMOS used by this unit. Then, the summation of all the physical components's power plus the idle power of the processor is the total power. Finally, they fine tune the parameters in the equations by running a group of training benchmarks, which exercise CPU differently, and compare the estimated power with power meters measured result.

Although previous efforts have already been able to accurately estimate the CPU power, they are not suitable for new on-chip multi-core processors (CMP) because part of the CPU functional unit, such as the last level cache, are shared between these cores. Also, new processors may supply different kinds of hardware performance events since the architecture of the processor changed rapidly. Following the previous works, Bertran *et al.* proposed a core level power models [6]. They categorize the processor components into three categories: the in-order engine, the memory subsystem and the our-of-order engine, based on extend that the PMCs can monitor their activities. For example, there are no PMC events to monitor the activity of some components in in-order engine category. Then select eight functional units, which include the front end component, the integer component, floating point component, the simple instruction & multiple data component, the branch prediction unit, the L1 cache, the L2 cache and the front side bus, to build the power model for a core. For the sharing components, such as L2 cache, the usage ratio will be computed individually. After choosing the components, they design a set of software micro-benchmarks that training different components and collect the maximum power information. Finally, they build the per core power model as equation 1.14 shows. In this equation, AR_i denotes the activity ratio of this component. The total CPU power is the summation of all the cores' power.

$$Power_{core} = (\sum_{i=1}^{i=compas} AR_i \times P_i) + Power_{static} \tag{1.14}$$

The previous works show that performance events can be directly used to build power model for CPU and memory, however, current performance counters do not directly supply useful events that reflect the activity of other devices. In [9], Bircher *et al.* show that processor related performance events are highly related with the power of other devices, such as memory, chipset, I/O and disk. Bircher *et al.* define chipset subsystem as processor interface chips that not within other subsystems, and define I/O subsystem as PCI (Peripheral Component Interconnect) busses and all devices attached to them. In able to establish the power model for these subsystems with processor related performance events, first they need to understand the correlation or say that how these events are propagated in the subsystem. Figure 1.5 shows the propagation of these performance events in all the subsystems they defined.

They select nine performance events based on the average error rate and comparison of the estimated and measured power traces. These events includes cycles, halted cycles, fetched uops, level 3 cache miss, TLB misses, DMA (Dynamic Memory Access) accesses, processor memory bus transactions, uncacheable accesses and interrupts. A more detail description can be found in [9]. Equation 1.15 shows the CPU power model they proposed. In this equation, the number 35.7, 9.25 and 4.31 is the maximum power dissipation of one CPU, the minimum power dissipation of one CPU and a constant value that reflect the relationship of the performance events and the real power.

$$\begin{aligned}\sum_{i=1}^{NumCPUs} & (9.25 + (35.7 - 9.25) \times PercentActive_i \\ & \qquad +4.31 \times \frac{FetchedUOPS_i}{Cycle})\end{aligned} \tag{1.15}$$

Usually, it is difficult to model the memory power, because memory accesses may happen both from the CPU-memory side and MEMORY-I/O side. The memory power model they proposed include two parts: power generated by CPU access and power generated by DMA. Power generated by CPU, as shown in Equation 1.16, is only related with the L3 cache misses and the cycle. Equation 1.17, in which MCycle is computed with Cycle, represent the other part of memory power. As we can see that it only related with bus transactions

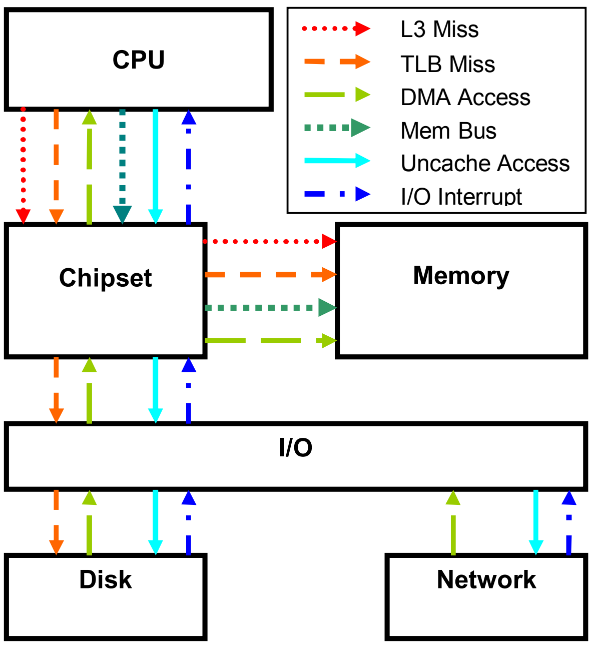


FIGURE 1.5: Propagation of performance events in the subsystems. (source [9])

and Cycle.

$$\sum_{i=1}^{NumCPUs}(28+\frac{L3LoadMisses_i}{Cycle}\times 3.43 \\ +\frac{L3LoadMisses_i^2}{Cycle}\times 7.66) \tag{1.16}$$

$$\sum_{i=1}^{NumCPUs}(29.2-\frac{BusTransictions_i}{MCycle}\times 50.1\times 10^{-4} \\ +\frac{BusTransictions_i^2}{MCycle}\times 813\times 10^{-8}) \tag{1.17}$$

The challenge of modeling disk power with performance events is because disk is logically far from CPU. Finally, they find that interrupts number and DMA accesses are most related with disk power. Equation 1.18 gives this power model. In the equation, the number 21.6 is the idle power of their disk.

$$\sum_{i=1}^{NumCPUs}(21.6+\frac{Interrupts_i}{Cycle}\times 10.6\times 10^{7} \\ +\frac{Interrupts_i^2}{Cycle}\times 11.1\times 10^{15} \\ +\frac{DMAAccess_i}{Cycle}\times 9.18 \\ +\frac{Interrupts_i^2}{Cycle}\times 45.4) \tag{1.18}$$

I/O subsystem are connected with many types of I/O devices, thus several performance events, such as DMA accesses, uncacheable accesses and interrupts, are related with the power of I/O subsystem. Although majority of I/O operations are caused by DMA access, they find that interrupts is more related with the power of I/O subsystem. Equation 1.19 shows the power model of I/O subsystem.

$$\sum_{i=1}^{NumCPUs}(21.6+\frac{Interrupts_i}{Cycle}\times 1.08\times 10^{8} \\ +\frac{Interrupts_i^2}{Cycle}\times 11.1\times 1.12^{9}) \tag{1.19}$$

1.3.2 Program Power Analysis

As we have mentioned before, power-aware hardware design is not enough to solve power problem, and higher-level power-aware strategies are even more important. In the future, both operation system and normal user space applications should be designed with considering energy efficiency. Several articles [31, 62] even proposed the concept of energy complexity, similar to time complexity and space complexity, as one of the metric that evaluating the quality of the algorithms. To understand the code-level power dissipation, we need to map the power to corresponding code block. With lower level program power characters, the program designer could find the power hungry areas or hotspots in the code. Furthermore, this information can be used as guidance for power-aware strategies. This is different with the power/performance tradeoffs, which only needs to synchronize the measured or estimated power to the counted performance metrics.

PowerScope [21], proposed by Flinn *et al.*, is one of the first tools on the mobile platform that profiling the energy consumption of applications. They could not only determine which fraction of the total system power is caused by a process, but also determine the energy consumption of the procedures within a process. Figure 1.6 shows the architecture of PowerScope. As we can see that, it includes three modules: energy monitor, system monitor and energy analyzer. The system monitor collects information such as the value PC register, process identifier (pid) and other information. For example, whether the system is currently processing an interrupt. The collected data is used to identify the executed program at a time point. To synchronize the system monitor and the energy monitor, they connect the multimeter's external trigger input and output to the parallel port of the profiling computer. By control the parallel port pin, they synchronize the data collection between the profiling machine and the data collection machine. The system monitor module on the profiling machine triggers the digital meter to sample the power, in this way they can synchronize the process events and measured power. Finally, the energy analyzer analyzes the raw data and generates the energy profile.

Different with PowerScope, which use time-driven method to sample, Chang *et al.* present an energy-driven method to help programmers evaluate the energy impact of their software design [13]. Energy-driven means they sample the data every energy quota, other than every time interval. They use the Itsy platform [54] we have described before to measure the power of the system. If the consumed energy comes to the energy quota, the Itsy platform will generate an interrupt to the profiling system. Then, the interrupt service will collect information that could identify the program block. Similar to PowerScope, they also analyze the program power profile off-line. The architecture of this method is shown in Figure 1.7.

As the further research of their earlier work [20], Ge *et al.* propose a program power analysis method called PowerPack [22] that is different with the last two methods [13, 21]. Different with the last two methods that analyze the power profile for mobile platforms, PowerPack is designed for desktop PC. The architecture of PowerPack is show in Figure 1.8. They run the profiled application, the system status profiler, a thread that controls the meters and a group of power reader threads on the same platform. PowerPack uses a special

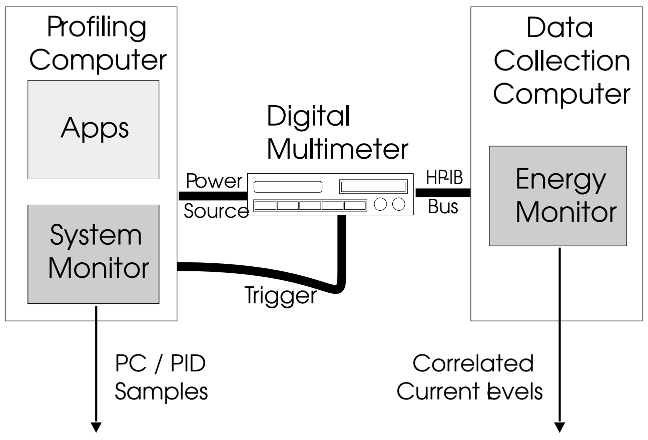


FIGURE 1.6: The architecture of PowerScope. (source [21])

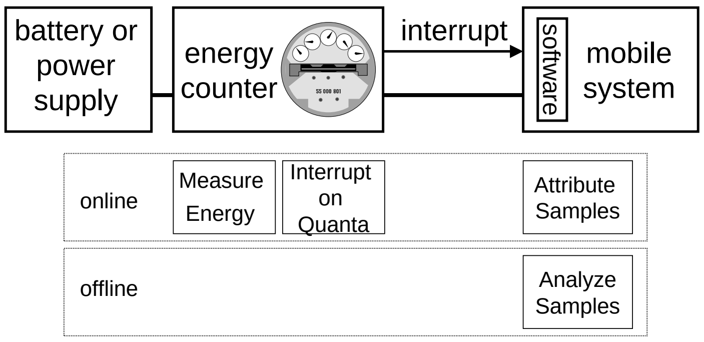


FIGURE 1.7: The architecture of event-driven method. (source [13])

Function	Description
pmeter_init	connect to meter control thread
pmeter_log	set power profile log file and options
pmeter_start_session	start a new profile session and label it
pmeter_end_session	stop current profile session
pmeter_finalize	disconnect from meter control thread

TABLE 1.1: The PowerPack power meter profile API (source [22])

Simulation	Hardware Measurement	Software Power Profiling
Wattch [11]	Itsy00 [54]	PowerScope [21]
SimplePower [58]	Joseph0106 [32]	Isci03 [28]
SoftWatt [24]	PowerPack [20, 22]	Li03 [38]
Orion [55]	PowerExecutive [1]	Chang03 [13]
SimWattch [15]	IMPI [27]	Lorch98 [40]
Dake94 [39]	Kamil08 [35]	Bellosa00 [5]
	DCEF [52]	Dempsey [59]
	Snowdon05 [51]	vEC [34]
		Powell09 [46]
		Bertran10 [6]
		Joseph0108 [33]
		Kansal10 [36]
		pTop [18]
		Dhiman10 [17]
		SoftWatt [24]
		Tempo [44]
		Contreras05 [16]
		Bircher [8, 9]
		SPAN [56]

TABLE 1.2: Classification of power profiling articles.

method that is different with the last two method to synchronize measured power to process information. Their method is used to find the power profile of predefined program code blocks. First, they implement a set of functions, as shown in Table 1.1, that will be called by the profiled applications before and after some critical code blocks. The executing of these functions then trigger the system status profiler and the meter control thread to sample the data. After that the power analyzer can analyze the collected data simultaneously. Finally, they propose a method to map the measured power into the application code and analyze the energy efficiency in a multi-core system.

Isci *et al.* developed an experimental framework to compare the control-flow based with the performance-monitoring-based power-phase detection techniques [30]. Their results show that both the control-flow and the performance statistics provide useful hints of the power phase behaviors.

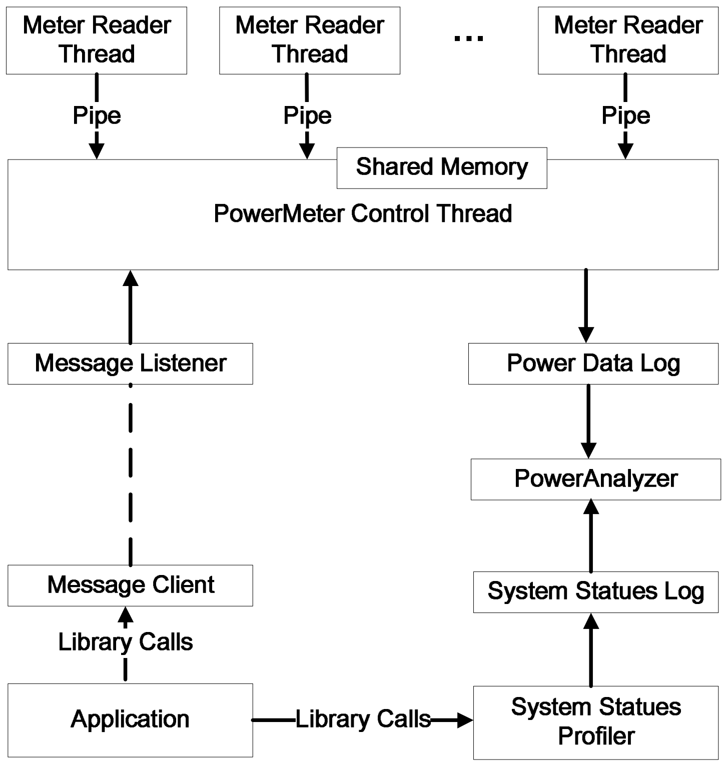


FIGURE 1.8: The architecture of PowerPack. (source [22])

Table 1.2 summarizes the classification of these previous publications.

1.4 Case Study

Power measuring and profiling are globally used in many areas on power/energy-aware system designs: making power/performance analysis, supplying directive power information for power aware strategies, and analyzing the power behaviors of programs, especially the hotspots, and so forth. In this section, we describe several case studies of using power measuring and profiling techniques.

1.4.1 Energy-aware Scheduling

Energy-aware scheduling is not only referred to scheduling algorithms that try to manage the system energy consumption, but also include other scheduling algorithms that try to manage power dissipation and component temperature. More specifically, energy-aware scheduling is scheduling methods that using system power characters to come to the following aims: conserving the energy consumption without decreasing the performance substantially [57], managing the heap power or temperature within a specified value [43, 47, 48], managing the energy consumption within a budget to extend the battery lifetime [61] and so on.

Based on task power profiles, Merkel *et al.* propose a method to scheduling tasks on a multiple processor system to maintain the power balance of these CPUs, and to reduce the need for throttling [43]. In this section, we describe their method in detail.

1.4.1.1 Task Energy Estimation

First, wee need to estimate the energy consumption of an individual task, because the energy-aware scheduling method makes decisions at the task level. This includes estimating how much energy a task spent in the last time unit and prediction the energy consumption in the next time unit. Although accuracy of the power model is important, a task-level power profiler must choose simple power models to decrease the overhead generated by the data collection module.

To estimate the energy consumption of a processor, they use performance counter to build a simple power model, that we have described in the last section. Then they determine the energy consumption of a task in the last time unit by reading these chosen event counters before and after this time unit. Except for the energy consumption of the last time unit, it is also critical to predict the underlining energy consumption in the next time interval. How-

Program	Maximum	Average
bash	19.0%	2.05%
bzip2	88.8%	5.45%
grep	84.3%	1.06%
sshd	18.3%	1.38%
openssl	63.2%	2.48%

TABLE 1.3: Change in power consumption during successive timeslices (source [43])

ever, estimating this is nearly inevitable because many factors may influence the power of the task. Merkel *et al.* assume the energy consumption of the last time interval equals to that of the last time interval, since they do a group of experiments and find that the energy consumption during two consecutive time intervals only changes with s very small percentage, normally below 6%. Table 1.3 shows their experiment result. We could see that although the difference may be very high in some circumstances, the average difference is very low. Thus, it is reasonable to use the last time unit's energy consumption as the prediction and make the scheduling decision.

To eliminate the task migration caused by misprediction, they use an exponential average function, shown as equation 1.20, to predict by using the task's past energy consumption. The weight p they used is based on the length of the time interval.

$$\overline{x_i} = p.x_i + (1-p).\overline{x_{i-1}} \tag{1.20}$$

1.4.1.2 The Design of the Scheduling Algorithm

Although task migration between CPUs can balance the temperature and workload of these CPUs, it may decrease the performance of the system cause this strategy is contradicted with processor affinity. Processor affinity is a kind of scheduling strategy, which scheduling a task to run on the CPU it has just run before, in this way the system may not need to reload the data and instructions of this task into the cache. Thus, task migration should be limited when designing an energy-aware scheduling.

They design two energy-aware scheduling strategies: energy balancing and hot task migration. Energy balancing is used on the circumstance that multiple tasks can be executed on one CPU. By rescheduling the execution of hot tasks and cool tasks, this strategy could balance the energy consumption on one CPU. Figure 1.9 shows the flow of this scheduling algorithm. In this way, further task migration may not be needed. However, this may not always the case. For the circumstances that only one task can be executed on the CPU, they use hot task migration, which try to migrate a hot task to another cool

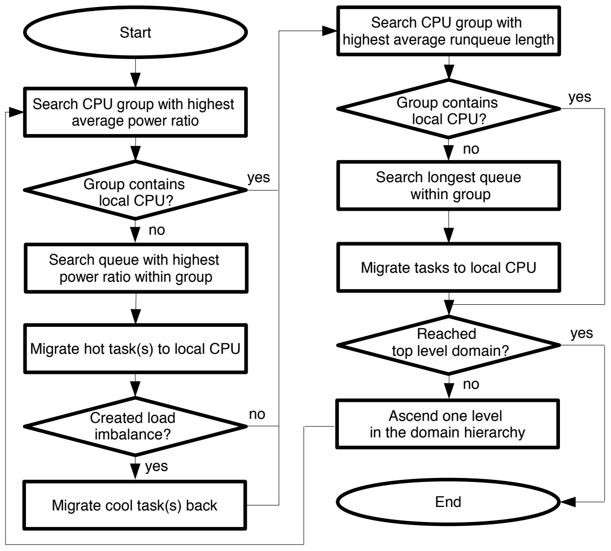

Start
Search CPU group with highest average power ratio
Group contains local CPU?
yes
no
Search queue with highest power ratio within group
Migrate hot task(s) to local CPU
Created load imbalance?
no
yes
Migrate cool task(s) back
Search CPU group with highest average runqueue length
Group contains local CPU?
yes
no
Search longest queue within group
Migrate tasks to local CPU
Reached top level domain?
yes
no
Ascend one level in the domain hierarchy
End

FIGURE 1.9: Energy and load balancing algorithm. (source [43])

CPU if the current CPU's temperature exceeds the limitation. Figure 1.10 shows the flow of this scheduling strategy.

1.4.1.3 Implementation & Evaluation

Merkel *et al.* implemented these two power-aware scheduling strategies on Linux 2.6.10 kernel. Furthermore, they implement the CPU energy estimation module and task energy profiling module on the same platform. The CPU's event counters are read on every task switch and at the end of each time interval. The values are transferred to energy value and save in a data structure.

To test the load balancing and temperature control, they run a group of programs on the system. First, they run these programs without using the power-aware strategies. The result is shown in Figure 1.11. Then they run the same group of programs with energy-aware strategies available on the same system. Figure 1.12 shows this result.

1.4.2 Software Power Phase Characterizing

Characterizing the power phase behavior of application has become more and more important, because the increasing of the complexity of the architecture and the global using of dynamic power-management strategies. For example, power phase characterizing can help the system optimize performance/power tradeoffs, and help programmer to identify critical execution

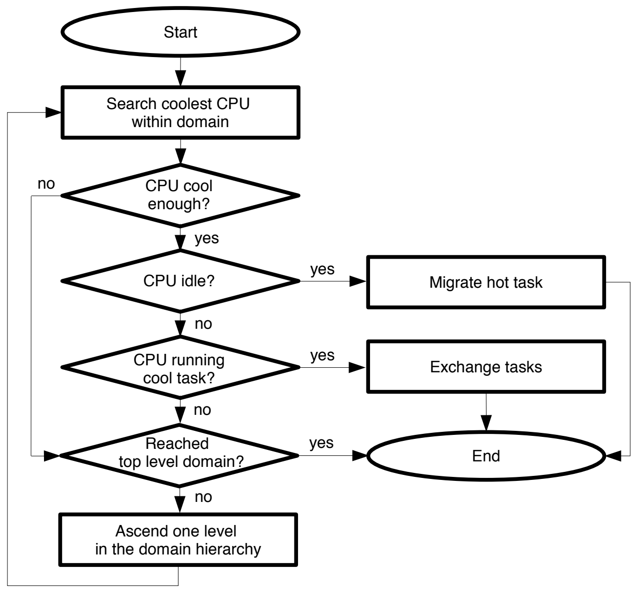


FIGURE 1.10: Hot task migration algorithm. (source [43])

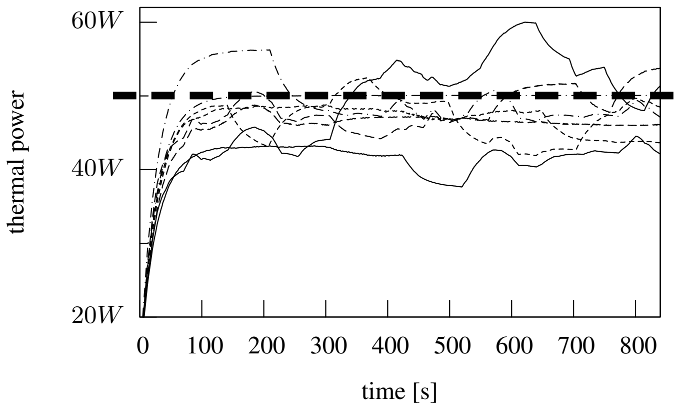


FIGURE 1.11: Thermal power of the eight CPUs with energy balancing disabled. (source [43])

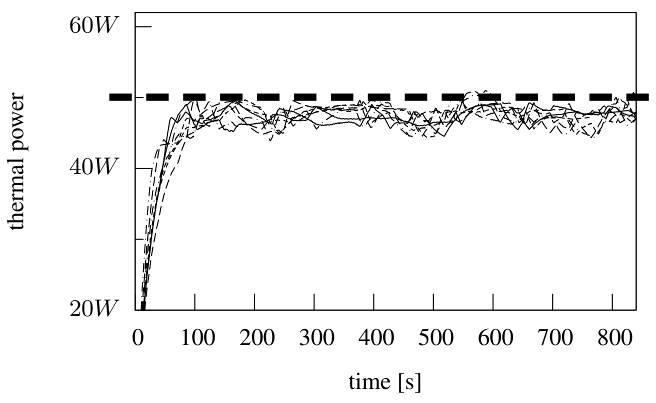


FIGURE 1.12: Thermal power of the eight CPUs with energy balancing enabled. (source [43])

areas that generate abnormal power dissipation. The basic aim of power phase characterizing is classifying software power behaviors into self similar operation regions. Isci *et al.* present a method for characterizing the running application's power phase in [30]. In this section, we describe their method in detail as a case study.

They use Pin [42] to collect synchronous information, such as system events, PMC events and measured power. Pin is a platform independent tool that design for software instrumentation, which is essential for applications to evaluate performance and to detect a bug. With the Pin API we can observe all the architectural state of a process, such as the contents of registers, memory, and control flow. The power information is required by measuring the current flow into the processor directly. The architecture of their method is shown in Figure 1.13.

The control-flow based application phase is tracked with the basic block vector (BBV) method [50], which maps executed PC address to the basic blocks of an application binary. Each item of BBV is corresponded to a specific block and saves a value that denotes how often this block is executed during the sampling period. Based on their past work [28], they select 15 PMC events that are more related with the CPU power to track the power phase. After that, the measured PMC events are converted into per-cycle rates, and saved in a data structure similar to BBV as a 15 dimensional vector. Finally, the analyzer cluster the sampling result saved in the BBV and the PMC vector into power phases with different clustering algorithms, such as first pivot clustering and agglomerative clustering.

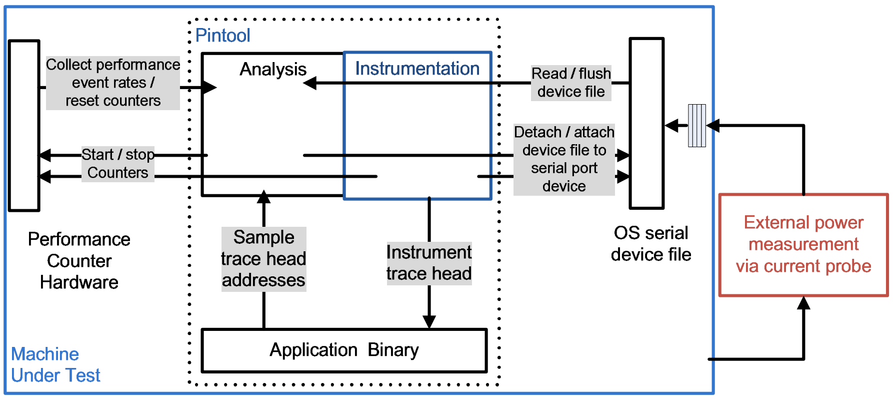

Pintool
Analysis
Instrumentation
Collect performance event rates / reset counters
Start / stop Counters
Read / flush device file
Detach / attach device file to serial port device
Performance Counter Hardware
Sample trace head addresses
Instrument trace head
OS serial device file
External power measurement via current probe
Application Binary
Machine Under Test

FIGURE 1.13: Experimental setup for power phase analysis with Pin. (source [30])

1.4.3 SPAN: a Realtime Software Power Analyzer

Section 1.3.2 presents several techniques of analyzing software power. These works using different strategies to synchronize the hardware measured power with the software execution information. In this section, we describe a realtime power analyzer, called SPAN [56], which could find the power dissipation at the function-block level. Different with other works, SPAN uses PMC events to estimate the power of the processor then analyze how this power dissipation is related to the current program's function blocks.

1.4.3.1 Power Estimation

Similar to other PMC-based power model, SPAN also use this strategy to estimate the CPU power. To get a high adaptability, SPAN only use IPC and CPU frequency to build the power model. A bunch of previous articles [38, 8] have proven the high correlation of IPC related PMC events with the power.

Assuming that a CPU supports n frequencies, f_i, $i = 1, 2, 3...n$, $P(f_i)$ is the power of CPU for each frequency f_i. And, $P(t_j, f_i)$ is the CPU power when executing benchmark t_j at frequency f_i and $IPC(t_i, f_i)$ is the corresponding IPC. After running a group of micro-benchmarks, they calculate $P(f_i)$ and $IPC(f_i)$ as the median of $P(t_j, f_i)$ and $IPC(t_i, f_i)$ individually. After that, they compute $\Delta P(t_j, f_i)$ as the difference between $P(f_i)$ and $P(t_i, f_i)$ for each training benchmark, shown as Equation 1.21. Similarly, they calculate ΔIPC (t_j, f_i) as the IPC difference of training benchmark t_i to the median value, shown as Equation 1.22.

$$\Delta P(t_j, f_i) = P(t_i, f_i) - P(f_i) \tag{1.21}$$

$$\Delta IPC(t_j, f_i) = IPC(t_i, f_i) - IPC(f_i) \tag{1.22}$$

Targeting on predicting $\Delta P(t_j, f_i)$, they use $\Delta IPC(t_j, f_i)$ as model input to derive linear regression parameters, $P_{inct}(f_i)$ and $P_{\Delta}(f_i)$ with Equation 1.23. The final predicted power dissipation is showing in Equation 1.24, in which $P(f_i)$ dominates the calculated value.

$$\Delta P(t_j, f_i)_{pret} = P_{inct}(f_i) + P_{\Delta}(f_i) * \Delta IPC(t_i, f_i) \tag{1.23}$$

$$P(t_j, f_i)_{pret} = \Delta P(t_j, f_i)_{pret} + P(f_i) \tag{1.24}$$

The modeling of multiple cores is based on the assumption that each core has similar power behavior. Therefore, they apply the single core model to each core in the system. The equation for the total power dissipation is shown in Equation 1.25, in which a_j is the target benchmark, $\Delta P(a_j, f_i, k)_{pret}$ is generated at per core level because different cores might have the varied value of $\Delta IPC(t_i, f_i, k)$.

$$P(a_j, f_i)_{pret\_total} = \sum_{k=1}^{k=cores} (\Delta P(a_j, f_i, k)_{pret} + P(f_i)) \tag{1.25}$$

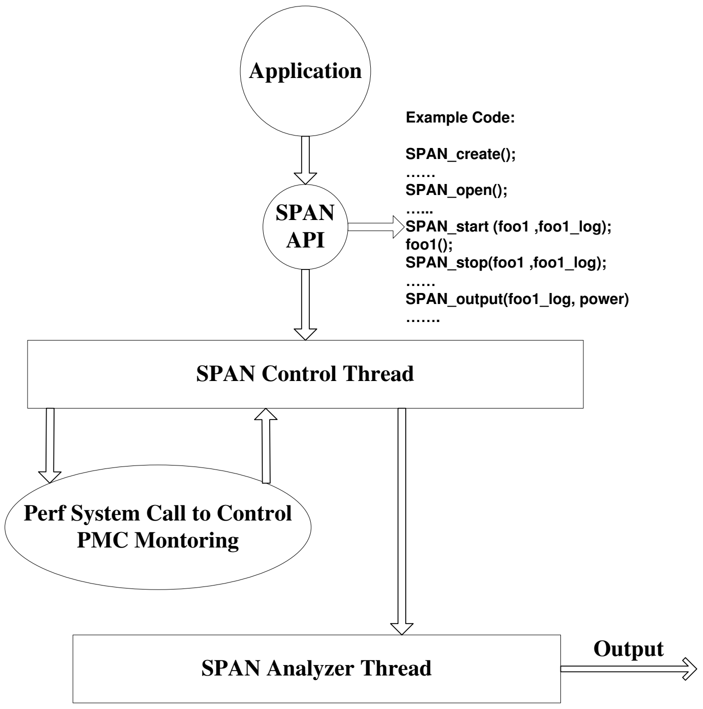
1.4.3.2 The Design and Implementation of SPAN

In able to synchronize program executing information to estimated power, they design a group of APIs for applications. Table 1.4 lists some of these APIs. Developers need to call these APIs at some critical area of their code. The basic flow of the SPAN tool is illustrated in Figure 1.14(a). The two inputs of SPAN are the application information and PMC counter values. At the application level, the application information and the estimation control APIs are passed to the control thread through the designed SPAN APIs. Finally, the SPAN outputs a figure of estimated power dissipation represented by different colors distinguishing different application functions, such as Figure 1.14(b) shows.

1.4.3.3 Evaluation

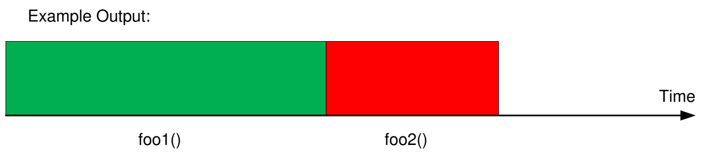
In this article, they mainly evaluate two aspects of the SPAN, the overhead and the responsiveness. They use two benchmarks to do the evaluation. One is the FT benchmark from NAS parallel benchmark suite. Another is a synthetic benchmark that we designed with a combination of integer operation, PI calculation, prime calculation, and bubble sort. First, they measured the execution with and without the SPAN instrumentation for ten times each. The differences of execution time are within 1% on average.

To evaluate the responsiveness of the power model, they compare the continuous measured and estimated values. From Figure 1.15, it is easy to observe that the estimated power is closely related to the measured power dissipation at the overall shape. Furthermore, the Figure marked the corresponding benchmark functions. The first iteration of benchmark FT mainly consists



```
SPAN_create();
......
SPAN_open();
......
SPAN_start (foo1 ,foo1_log);
foo1();
SPAN_stop(foo1 ,foo1_log);
......
SPAN_output(foo1_log, power)
.......
```

(a) The flow chart of SPAN.



(b) The example output of SPAN

FIGURE 1.14: Desgin of SPAN.

APIs	Description
`span_create()`	Prepare a power model profile which records basic parameters
`span_open()`	Initialize a SPAN control thread and targeting PMCs
`span_start(char* func, char* log)`	Record the targeting application function and specify the log file name
`span_stop(char* func, char* log)`	Stop the power estimation for a specified application function
`span_pause()`	Temporally stop reading PMCs
`span_continue()`	Resume reading PMCs
`span_change_rate(int freq)`	Shift the estimation rate, basically this methods control the PMC sampling rate
`span_change_model(float* model, File* model)`	Modify the model parameters in the model file according to the platform
`span_close()`	Close the opened PMCs and SPAN control thread
`span_output(char* log, FILE* power)`	Invoke SPAN analyzer thread and produce the detailed power estimation information with respective to the profiled functions to the destination file

TABLE 1.4: SPAN APIs.

of two functions, $compute\_initial\_conditions()$ and $fft()$. The rest iterations follow the same procedure, which can be clearly observed from Figure 1.15, but the estimations present a certain level of delay due to the rapid function changes in the source code. Moreover, in Figure 1.16, they deliberately insert $sleep()$ function between each sub benchmark in the synthetic workload in order to distinguish each one of them easily. The achieved error rate is as low as 2.34% for the two benchmarks on average.

1.4.4 Cinder

Cinder [49] is a power-aware operating system that designed for mobile devices. Similar with EcoSystem [61], Cinder also tries to manage battery energy as one kind of resources. In order to control energy consumpption, Roy *et al.* proposed two abstract definitions in the Cinder operating system, *reserves* and *taps*, which store and distribute energy for application use. This section describes how Cinder was designed and implemented.

1.4.4.1 The design of Cinder

Cinder is implemented based on HiStar kernel [60], which is a secure operating system. They implemented *reserves* and *taps* as two new fundamental kernel objects.

A *reserves* shows the rights to use a given quantity of energy. Cinder allocates a *reserves* for each application based on their power dissipation information, which is estimated by the power profiling module. This module was implemented by using standard device-level accounting and modeling [61]. Energy consumed by a program will be deducted from its reserve. If the reserve of an program does not have enough energy, the execution of this program will be halted. Energy consumption estimation is very important for reserve allocation.

Among a program, reserve could be delegated and subdivided by threads.

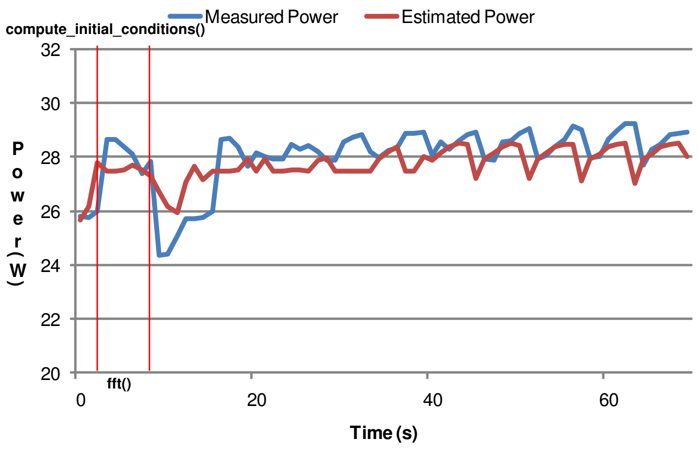


FIGURE 1.15: The FT benchmark with SPAN instrumentation. (source [56])

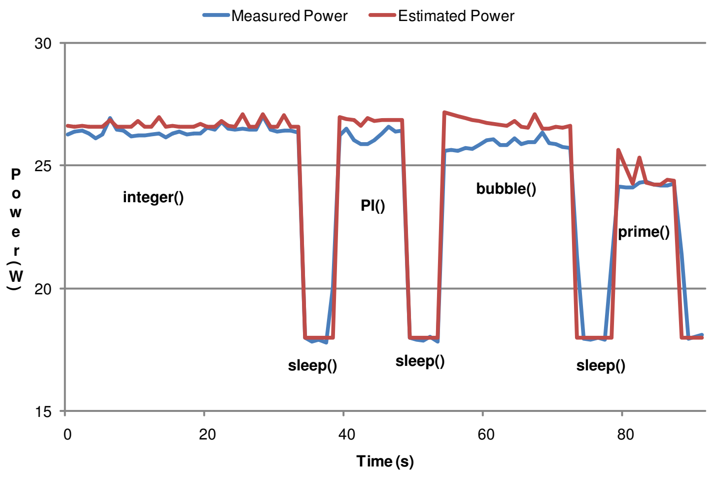


FIGURE 1.16: The synthetic benchmark with SPAN instrumentation. (source [56])

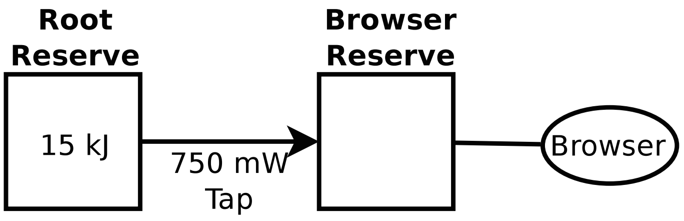


FIGURE 1.17: A 15 kJ battery, or root reserve, connected to a reserve via a tap. (source [49])

For example, an application, which has a reserve of 1000mJ, could split 200mJ of reserve to one of its thread. When the application stop to run, the reserve of which will be recycled.

A tap represents the amount of energy that transfered from one reserve to the other one in the unit time. They use tap to control the maximum energy consumption rate. An application reserve with a 1mJ/s tap connected to the battery reserve, which is the root reserve of all the applications, means that this application cannot consume more than 1mJ energy in a second. Through taps they could control the power dissipation of the whole system and the battery lifetime. Figure 1.17 show an example of how reserves and taps work. In this example, the web browser reserve draws energy from the root reserve (battery) through a 750mW/s tap.

Different with EcoSystem, Cinder adds a new function called cooperation, which means several applications collaborate to fulfill a task, such as radio power up event, that need to consume a large amount of energy. This is done by using a public reserve, each each application who want to use the service splits part of energy by adding a tap to this public reserve.

1.4.4.2 Evaluation

The result of on Figure 1.18 shows the differences of using cooperation and without cooperation. First, the figure of uncoperation shows that the radio nearlly active all the time. This is mainly cause by those short radio usage. The figure of cooperation shows that during the active period, the radio usually sleep for a while. This is because none of an application has enough energy to activate the radio. In this way, the applications will wait till there is enough energy. This result shows Cinder could control energy consumption in the right way. Also, the second result is obviously more energy efficient than the first method.

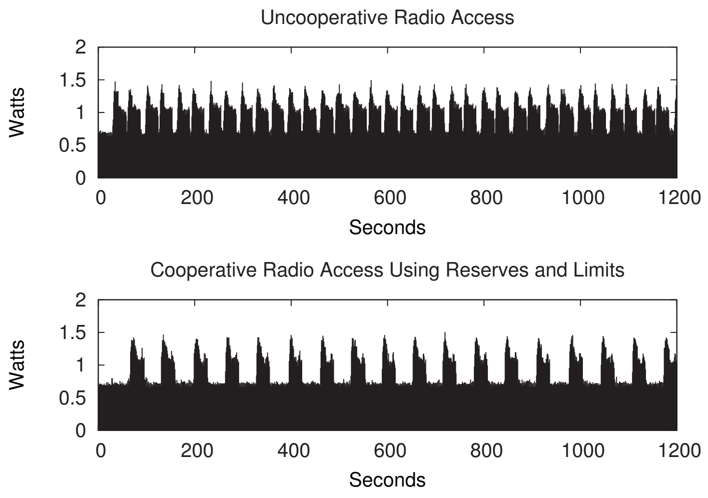


FIGURE 1.18: The comparison of using cooperation and without cooperation. (source [49])

1.5 Summary & Future Work

In this survey, we provide an introduction to the power measuring and profiling techniques that are globally used in power aware system design area. By summarizing the direct power measurement method and software-based power estimation techniques, and the description of several case studies, we show researchers how these methods could be used in their future research.

Power measuring and profiling have already been studied extensively. However, more investigations are needed in the following two areas: improving power profiling techniques and using these strategies in power-aware system design. As we have mentioned before, accuracy is not the only requirement, simplicity and adaptability are about equally important. In the future, the operating system should supply a module to do the power profiling and supply configurable accuracy. In addition, as the fast changing of hardware architectures, new methods should adapt to these new architectures. For example, as the number of cores on a single chip keeps increasing, on-chip network fabrics become one of the main power dissipation resources. Thus, future research needs to consider this unit and reevaluate the power indicators that are currently used.

Currently, power measurements and profiling has been used for making power/performance trade-offs during hardware designs, supply basic information for power-aware strategies and so on. Exploiting how to make use of power profiling approaches to design energy efficient software is also an interesting direction, which is highly related to the research activities in the operating systems and software engineering field.

Bibliography

[1] Ibm powerexecutive. http://www-03.ibm.com/systems/management/director/about/director52/extensions/powerexec.html.

[2] Watts up. https://www.wattsupmeters.com.

[3] Ishfaq Ahmad, Sanjay Ranka, and Samee Ullah Khan. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –6, apr. 2008.

[4] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, December 2007.

[5] Frank Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, EW 9, pages 37–42, New York, NY, USA, 2000. ACM.

[6] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 147–158. ACM Press, 2010.

[7] Suparna Bhattacharya, Karthick Rajamani, K. Gopinath, and Manish Gupta. The interplay of software bloat, hardware energy proportionality and system bottlenecks. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 1:1–1:5, New York, NY, USA, 2011. ACM.

[8] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *ISLPED 05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 275–280. ACM Press, 2005.

[9] W. Lloyd Bircher and Lizy K. John. Complete system power estimation: A trickle-down approach based on performance events. *Performance Analysis of Systems and Software, IEEE International Symmposium on*, 0:158–168, 2007.

[10] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. *The case for power management in web servers*, pages 261–289. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[11] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94, New York, NY, USA, 2000. ACM.

[12] John Byrne, Jichuan Chang, Kevin T. Lim, Laura Ramirez, and Parthasarathy Ranganathan. Power-efficient networking for balanced system designs: early experiences with pcie. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 3:1–3:5, New York, NY, USA, 2011. ACM.

[13] Fay Chang, Keith Farkas, and Parthasarathy Ranganathan. Energy-driven statistical sampling: Detecting software hotspots. In Babak Falsafi and T. Vijaykumar, editors, *Power-Aware Computer Systems*, volume 2325 of *Lecture Notes in Computer Science*, pages 105–108. Springer Berlin/Heidelberg, 2003.

[14] Jianmin Chen, Bin Li, Ying Zhang, Lu Peng, and Jih kwon Peir. Statistical gpu power analysis using tree-based methods. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1 –6, july 2011.

[15] Jianwei Chen, Michel Dubois, and Per Stenström. Simwattch: Integrating complete-system and user-level performance and power simulators. *IEEE Micro*, 27(4):34–48, 2007.

[16] Gilberto Contreras and Margaret Martonosi. Power prediction for intel xscale processors using performance monitoring unit events. In *Proceedings of IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 221–226, 2005.

[17] Gaurav Dhiman, Kresimir Mihic, and Tajana Rosing. A system for online power prediction in virtualized environments using gaussian mixture models. In *Proceedings of the 47th ACM IEEE Design Automation Conference*, pages 807–812. ACM Press, 2010.

[18] Thanh Do, Suhib Rawshdeh, and Weisong Shi. ptop: A process-level power profiling tool. In *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09)*, oct 2009.

[19] R. Duan, Mingsong Bi, and C. Gniady. Exploring memory energy optimizations in smartphones. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1 –8, july 2011.

[20] Xizhou Feng, Rong Ge, and Kirk W. Cameron. Power and energy profiling of scientific applications on distributed systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, IPDPS '05, pages 34–, Washington, DC, USA, 2005. IEEE Computer Society.

[21] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[22] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):658–671, 2010.

[23] Ravi A. Giri and Anand Vanchi. Increasing data center efficiency with server power measurements. Technical report, Intel Information Technology, 2010.

[24] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, N. Vijaykrishnan, Mahmut Kandemir, Tao Li, and Lizy Kurian John. Using complete machine simulation for software power estimation: The softwatt approach. In *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, page 141, Washington, DC, USA, 2002. IEEE Computer Society.

[25] L.P. Hewlett-Packard Development Company. Service processor users guide. Technical report, Hewlett-Packard Development Company, L.P., 2004.

[26] Timo Hönig, Christopher Eibel, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Seep: exploiting symbolic execution for energy-aware programming. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 4:1–4:5, New York, NY, USA, 2011. ACM.

[27] Intelligent platform management interface. http://www.intel.com/design/servers/ipmi/index.htm.

[28] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93, Washington, DC, USA, 2003. IEEE Computer Society.

[29] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. Technical report, Princeton University Electrical Eng. Dept., September 2003.

[30] Canturk Isci and Margaret Martonosi. Phase characterization for power: evaluating control-flow-based and event-counter-based techniques. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 121 – 132, feb. 2006.

[31] R. Jain, D. Molnar, and Z. Ramzan. Towards a model of energy complexity for algorithms [mobile wireless applications]. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1884 – 1890 Vol. 3, march 2005.

[32] Russ Joseph, David Brooks, and Margaret Martonosi. Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs. In *In Workshop on Complexity Effectice Design WCED, held in conjunction with ISCA-28. Jun 2001*, June 2001.

[33] Russ Joseph and Margaret Martonosi. Run-time power estimation in high performance microprocessors. In *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, pages 135–140, New York, NY, USA, 2001. ACM.

[34] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *PASTE '01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 28–31, New York, NY, USA, 2001. ACM.

[35] Shoaib Kamil, John Shalf, and Erich Strohmaier. Power efficiency in high performance computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –8, Apr. 2008.

[36] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[37] Samee Ullah Khan and Ishfaq Ahmad. A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. *Parallel and Distributed Systems, IEEE Transactions on*, 20(3):346 –360, mar. 2009.

[38] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, 2003.

[39] Dake Liu and C. Svensson. Power consumption estimation in cmos vlsi chips. *Solid-State Circuits, IEEE Journal of*, 29(6):663 –670, jun. 1994.

[40] Jacob R. Lorch and Alan Jay Smith. Apple macintosh's energy consumption. *IEEE Micro*, 18(6):54–63, 1998.

[41] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Power-aware operating systems for interactive systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 10(2):119–134, 2002.

[42] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40:190–200, June 2005.

[43] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev.*, 40:403–414, April 2006.

[44] Donald Molaro, Hannes Payer, and Damien Le Moal. Tempo: Disk drive power consumption characterization and modeling. In *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on*, pages 246 –250, may. 2009.

[45] Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34:52–58, 2001.

[46] Michael D. Powell, Arijit Biswas, Joel S. Emer, Shubhendu S. Mukherjee, Basit R. Sheikh, and Shrirang Yardi. Camp: A technique to estimate per-structure power at run-time using a few simple parameters. In *IEEE 15th International Symposium on High Performance Computer Architecture, 2009. HPCA*, pages 289–300, 2009.

[47] Michael D. Powell, Mohamed Gomaa, and T. N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. In *In Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 260–270, 2004.

[48] Erven Rohou and Michael D. Smith. Dynamically managing processor temperature and power. In *In 2nd Workshop on Feedback-Directed Optimization*, 1999.

[49] Arjun Roy, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 139–152, New York, NY, USA, 2011. ACM.

[50] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 45–57, New York, NY, USA, 2002. ACM.

[51] David C. Snowdon, Stefan M. Petters, and Gernot Heiser. Power measurement as the basis for power management. In *IN 2005 WS OPERAT. SYSTEM PLATFORMS FOR EMBEDDED REAL-TIME APPLICATIONS*, 2005.

[52] Intel Information Technology. Data center energy efficiency with intel power management technologies. Technical report, Intel Information Technology, Feb. 2010.

[53] Amin Vahdat, Alvin Lebeck, and Carla Schlatter Ellis. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, EW 9, pages 31–36, New York, NY, USA, 2000. ACM.

[54] Marc A. Viredaz, Marc A. Viredaz, Deborah A. Wallach, and Deborah A. Wallach. Power evaluation of itsy version 2.3. Technical report, Compaq, Western Research Laboratory, 2000.

[55] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. Orion: a power-performance simulator for interconnection networks. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 294–305, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[56] Shinan Wang, Hui Chen, and Weisong Shi. Span: A software power analyzer for multicore computer systems. *Elsevier Sustainable Computing: Informatics and Systems*, page In press, 2011.

[57] Andreas Weissel and Frank Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '02, pages 238–246, New York, NY, USA, 2002. ACM.

[58] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *DAC '00: Proceedings of the 37th Annual Design Automation Conference*, pages 340–345, New York, NY, USA, 2000. ACM.

[59] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[60] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in histar. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 19–19, Berkeley, CA, USA, 2006. USENIX Association.

[61] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: managing energy as a first class operating system resource. *SIGPLAN Not.*, 37(10):123–132, 2002.

[62] Kostas Zotos, Andreas Litke, Er Chatzigeorgiou, Spyros Nikolaidis, and George Stephanides. Energy complexity of software in embedded systems. *ACIT - Automation, Control, and Applications*, 2005.