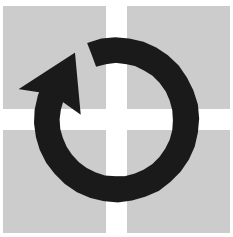
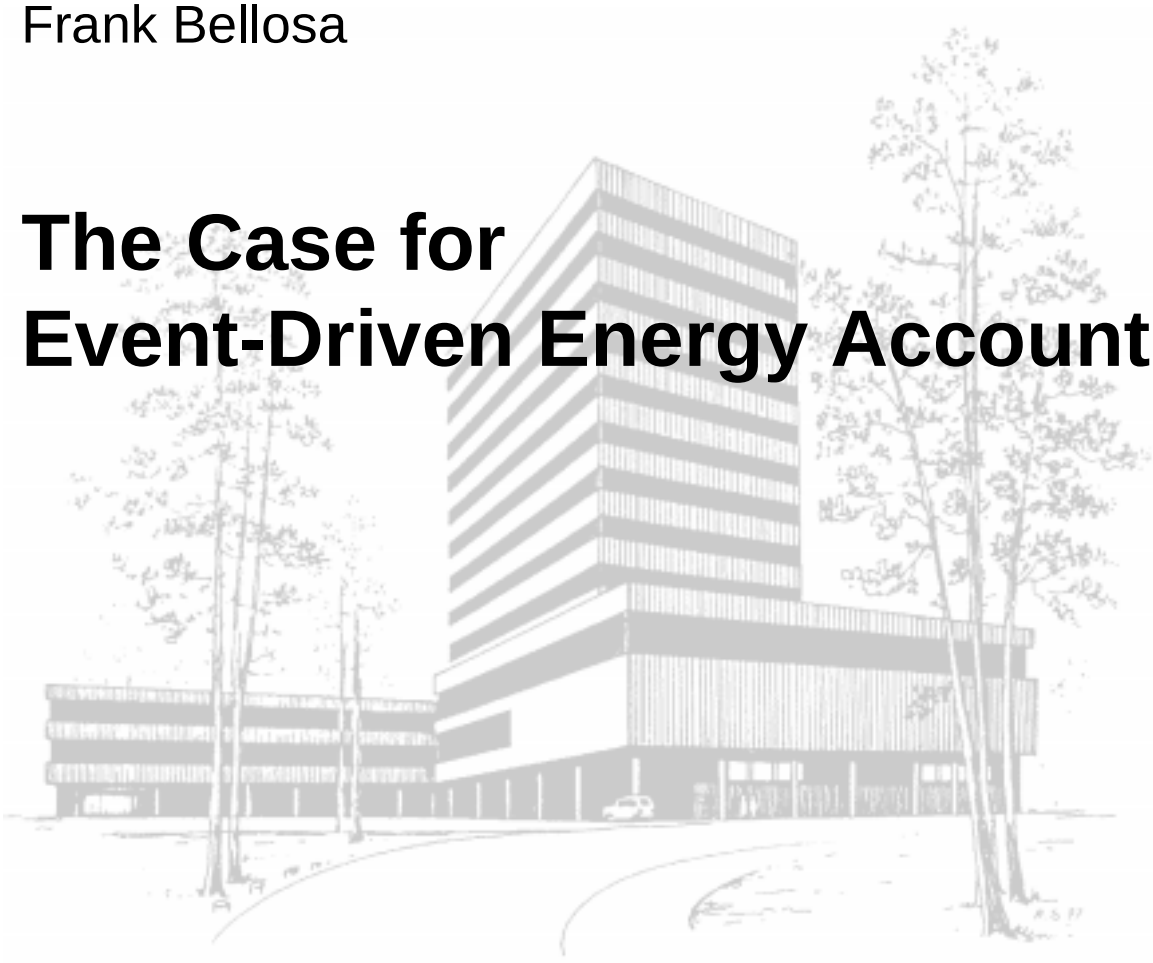


Department of Computer Science 4
Distributed Systems and Operating Systems

Frank Bellosa



The Case for Event-Driven Energy Accounting

Technical Report TR-I4-01-07

Friedrich-Alexander-Universität Erlangen-Nürnberg
TECHNISCHE FAKULTÄT

The Case for Event–Driven Energy Accounting

Frank Bellosa*

Department of Computer Science (Operating Systems), University of Erlangen,
Martensstr. 1, 91058 Erlangen, Germany
bellosa@cs.fau.de

Energy management requires a precise knowledge of the patterns of energy use. Not only knowing where the energy has been spent is important but also knowing who was responsible for the use of energy.
The resolution of power measurement equipment like current meters or smart battery interface is neither sufficient to identify the hardware unit consuming the energy nor to identify the originator (*energy principal*) of a specific hardware activation. Our measurements have also demonstrated that timing data, the source of information used in current operating systems, is not adequate to estimate the use energy consumption, because the power depends on the patterns for the use of specific functional units.

To investigate energy usage patterns we strongly encourage the use of embedded hardware monitors (e.g., processor performance counters) that have also proven to offer valuable information in the field of performance analysis. We use information about active hardware units (e.g., instruction decoder, memory management unit, cache-/memory-interface) gathered by event counters to establish a precise and energy principal-specific energy accounting.
By showing the correlation of events and energy values we can provide the necessary information for energy-aware scheduling policies that go far beyond the capabilities of the state of the art power management standards like ACPI. While the aspect of energy saving by driving the system at its optimal operation point is important in mobile devices, the aspect of client- and service-specific throttling is vital in data-centers where servers have to provide mission critical services even when running under an emergency power supply in times of (rolling) black outs.
An implementation of the proposed event-driven energy accounting using a Linux/x86 system and extensive measurements prove the concept.

1 Introduction

Without energy the processing and transport of data is impossible. Nonetheless the measurement, accounting, and management of energy has been widely unattended in the field of systems research. With the emergence of portable and wireless devices and with the energy crisis affecting data centers and server-farms in many parts of the United States we are suddenly facing a rising awareness for the topic of energy management.

This paper contributes to this awareness and initiates a new phase in the evolution of OS power management. To better classify our approach we describe the previous phases:

(1) The operating system analyzes the active and idle times of a device (e.g., CPU, hard disk or display) and makes assumptions about the future use of the device. Based on these assumptions the, OS decides when a HW-component should switch its power-state to a low-power mode. These power-modes differ in their energy savings, the time to switch the power state, and the energy that is necessary to perform this change of state. These simple heuristic policies only affect the behavior of applications by the latency to switch to an active mode or the performance in a low-performance active mode. Because they affect the behavior of devices on a system wide-scope, these policies resemble centrally-planned economies that neglect the individual demands and characteristics of the objects (e.g., applications serving a request) affected by the decisions. These policies have no knowledge of the usage patterns and demands of a specific application. Consequently, they cannot optimize their strategy such that some low-priority applications suffer from higher latencies while other high-priority tasks are not significantly affected. An example of this type of power management is found in Windows2000 [21] using ACPI [16].

(2) Having several HW-components available that can fulfill the same service offers a further chance to influence the energy consumption. By focusing the load on a subset of the components (load consolidation), we generate idle components that can be set into a low-power state.
Especially in clustered environments with nearly stateless systems which process short term requests, this approach of energy saving is effective and can easily be realized by an energy-aware load distribution mechanisms [8]. A typical representative of these environments are web-server farms with diskless compute nodes and network-attached storage devices. Here the compute nodes are the targets for load consolidation. However, this approach does not apply to complex services consisting of several interdependent sub-services which built up some state (e.g., file-locking, transactions) while answering request-sequences.
We see a consolidation technique that has been successfully applied to fail-safe, hot-swappable computer architectures for more than 20 years as a good candidate for power-savings in server systems. In these systems, built as shared-memory multiprocessor or single-chip multiprocessor architectures, tasks can easily be moved to another CPU due to the common memory. In this way we

* Frank Bellosa was on sabbatical as an academic visitor of the SawMill group at the IBM Watson Laboratory from November 2000 to April 2001.
The implementation and measurements were supported by the IBM Austin Research Laboratory .

can consolidate the load and shutdown some of the processors even when complex state in the common memory has been established.

(3) The last step in the evolution of power management is a renouncement from a system centric approach to an energy management, that rules according to the individual characteristics of applications, the demands of specific requests and the energy related status of the whole system. According to our economic analogy of (1) this approach resembles more a social market economy.
Because the operating system has the complete knowledge where the energy has been spent, who was responsible for the use of energy and what are the individual demands, power management can find a trade-off between energy consumption and quality of service demands. Examples are the task-specific control of the CPU speed and voltage under consideration of the task's quality-of-service demands (a potential scenario for the Intel XScale architecture [14]), the delay of certain requests to reduce the number of energy-expensive power-up state-transitions of HW-components (see [26] of IBM ARL) or the throttling of specific activities with respect to quality-of-service/environmental/cooling/ energy-supply demands (see section 4).

This paper of the OS-research group of the University of Erlangen belongs to the third phase in power management research. Section 2 describes the energy-related behavior of hardware-components like processor-core, caches, main-memory and I/O devices and motivates our approach for event -driven energy accounting called Joule Watchers that was introduced in [4].

Section 3 details the Joule Watchers energy accounting system and addresses the fallacies and pitfalls we encountered. We identify the limitations of the approach and sketch architectural improvements that increase the fidelity of Joule Watchers. The first results of an implementation on an x86 architecture are presented.).

In Section 4 we propose some on-line scheduling policies that rely on the precise energy accounting of Joule Watchers. The benefit of one of the policies is demonstrated by an implementation and energy measurements with a high-resolution data-acquisition system. Here we can show that the power of a server system can be throttled so that the average power within a time window of 50 milliseconds will never exceed a given threshold. This type of throttling might become vital for data centers which have to provide 24x7 services even when running under an emergency power supply.
We summarize with a description of some future projects of our group.

2 Energy-Related Characteristics of System Components

2.1 Processor/Memory/Interconnection Technology

In semiconductor technology, energy is (very simplified) used whenever current is flowing due to leakage or due to loading/ deloading of capacitors triggered by transistor switch operations. The leakage current depends on static parameters like time, voltage and properties of the semiconductors. In addition to these static parameters the dynamic energy consumption depends on the switching frequency of the transistors and the size of the capacitors.
If we want to identify those parts of a system architecture which contribute significantly to the total energy consumption, we have to look at those parts containing most of the capacitors and those with the highest switching frequencies:

- The processor core executing algorithmic, logical, or control flow operations consumes energy depending on the switching activity within the essential functional units. As the basic activity of the CPU core proceeds in clock cycles, we expect some relation between energy consumption and clock frequency. However, a major part of the activity depends on the type of instructions and their operands. Therefore, we have to keep an eye on the activity of each functional unit.
- If a memory management unit (MMU) is used in a computer architecture for reasons of mapping and protection, the MMU will contribute significantly to the energy consumption as it built up from full associative memory that is accessed whenever memory is referenced. Therefore, the energy consumption might depend on the memory-reference patterns of the executed software.
- Caches contribute to static energy consumption depending on their size but also to dynamic energy consumption depending on the frequency of cache references and the associativity of the cache indexing algorithm. The higher the associativity, the more cache-tags have to be compared at each cache reference.
- Main memory built out of dynamic memory cells (DRAM) is build up of capacitors to store information. We expect it to contribute to the static energy consumption depending on the size of memory.
  For each memory request. the data has to be transferred from the addressed memory cell to an intermediate buffer within the memory chip before it is transferred over the interconnection network (e.g., a bus system) to the requesting device (e.g., the CPU or an DMA capable I/O device). Therefore, we see a dynamic energy consumption of DRAM as well.
  Advanced memory modules (e.g., RDRAM) also offer several low-power states which differ in the latency to re-activate the memory module again. In a low-power state some parts of the address-logic and the bus connectors of the module are shut down. This saving in passive energy consumption has to be paid by a higher access latency.
- The interconnection network (e.g., the bus system) contributes mainly to the dynamic energy consumption as the capac-

itance of the bus-lines has to be loaded/unloaded at each bus cycle. Therefore, we expect an energy consumption which is related to the activity level of the interconnect.

We have measured the energy consumption of a Pentium III 866 system with 1 GB RAM to determine the variation in power of the processor and the memory. During all tests the processor was always busy. We ran several simple loops that involve different functional units. An arithmetic test doing arithmetic operations (additions) with all operands in register. We ran two tests to investigate the influence of predicted and misspredicted branches on the energy consumption. One of the test heavily triggers the memory management unit, by steadily saving/restoring the registers to/from the stack. Finally we ran three tests which write data to the caches and the memory.

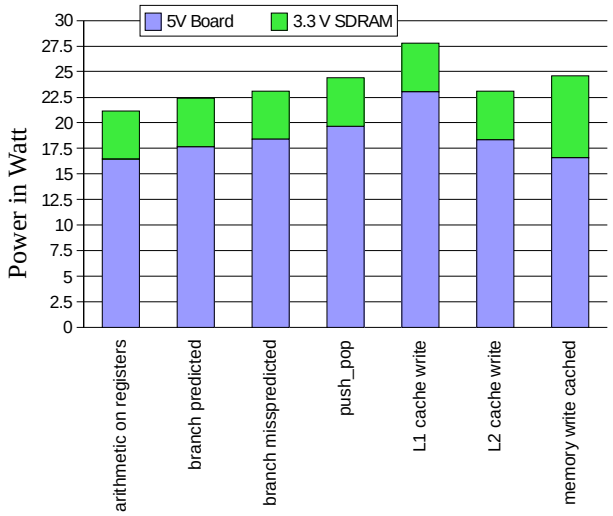


FIG. 1 : PC Board (PIII 866, 1 GB) Power Breakdown

The PC system shows a variation in the power at the 5V voltage, that is converted to 1.65V to supply the CPU, from 16.43 Watt up to 23.05 Watt (see Figure 1). If the main memory is involved heavily, the 3.3V power which supplies the SDRAM rises from 4.73 W to 8.03 W.

At the first glance we see, that the energy consumption of a system depends on more parameters than just the time. The power of the processor and the memory varies by more than 50%.

If the power management solely relies on timing information to make decisions, the power estimation can be wrong by 40%. The knowledge of the involved HW-components is essential to determine the power of a system.

2.2 Variable Clock Speed and Variable Voltage

A genuine saving in energy is possible with processors that offer the feature of tunable clock speed to reduce the power of the processor. The CPU speed and a dynamically adapted supply voltage change the energy performance [28, 11, 17, 24]. Neglecting the impact of external effects on CPU performance (e.g., memory latencies, bus contention) the energy per operation remains constant when reducing the frequency at a fixed voltage. However, the energy per operation is proportional to s^2, if frequency and voltage is changed by a factor s within the allowable limits of operation.

Particularly the academic results concerning variable voltage design have been applied to latest products (Intel XScale, Intel Mobile III [15], Transmeta Crusoe [27], AMD K6+ with PowerNow). According to this assumption the processor runs more efficiently at the lowest possible speed and voltage. However, this assumption does not hold, if we look at main memory, which is driven by a fixed voltage and that benefits concerning energy efficiency from requests within the same address range, because each memory reference involves a complete fetch of a memory row into a row-buffer. If data from the same row is referenced within a short period of time the memory operation is more energy efficient. Therefore, memory benefits from a fast processors in terms of energy consumption, because a faster processor can issue more memory operations. To demonstrate these effects, we show some data that has been gathered with a StrongARM SA-1100 system, which was designed to run at variable speed and voltage [25]. If we look at the power-curves of the variable speed processor running a bandwidth test (see Figure 2), we see that the energy per megabyte is the lower, the slower the CPU is running. On the other side. the energy of the memory to copy data is higher if the CPU is running slowly.

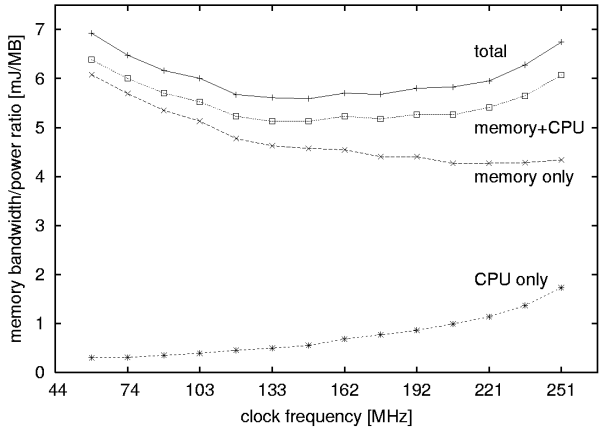


FIG. 2 : Energy breakdown for memory read (SA-1100 with 32 MB of EDO RAM) Measurements of the University of Delft [25]

In conclusion we can say that there is an optimal operation point near 142 MHz where the energy of the whole system, running the memory bandwidth test, is minimal. That means, that depending on the memory reference patterns of the application, the optimal operation point is between the minimal frequency and 142 MHz.

Consequently the optimal frequency for minimal energy consumption of a variable speed/voltage CPU can only be determined, if the memory reference characteristics of the application is known.

2.3 I/O Technology

Most of the I/O devices have an energy characteristics that differ from the above-mentioned HW-components. Besides active energy consumption which depends mainly of the number of I/O requests, there is a static part which depends on the power state of the device.

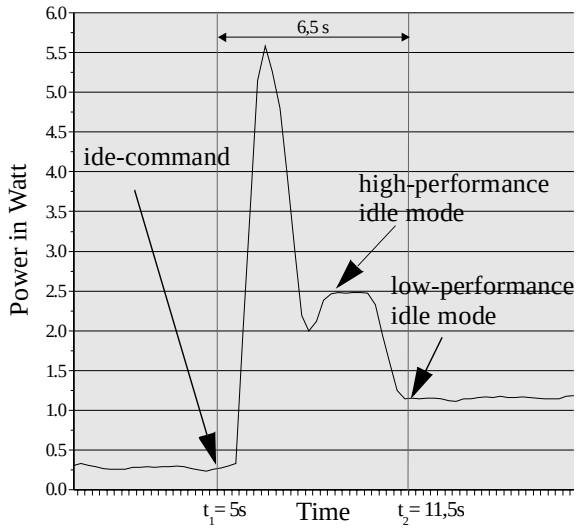


FIG. 3 : Power consumption of an IBM DTNA-22160 disk spinning up.

In addition to the increased latency to access a device and the energy costs to reactivate the device (e.g. spin-up of a hard-disk see Figure 3), there is an additional penalty in energy consumption which is neglected in most of the simulation-based research project of the past.

Not only to wake-up a device from a low-power state, but also to switch a device to a low-power state can consume energy.

One example is the head-lock mechanism of a hard-disk, that has to grip and lock the disk-head after the disk has spun down (see Figure 4).

An other example for power-down energy consumption is the protocol between a wireless network adaptor and the base station to inform the base station that it has to buffer incoming packets, while the network adaptor is powered-down for some seconds.

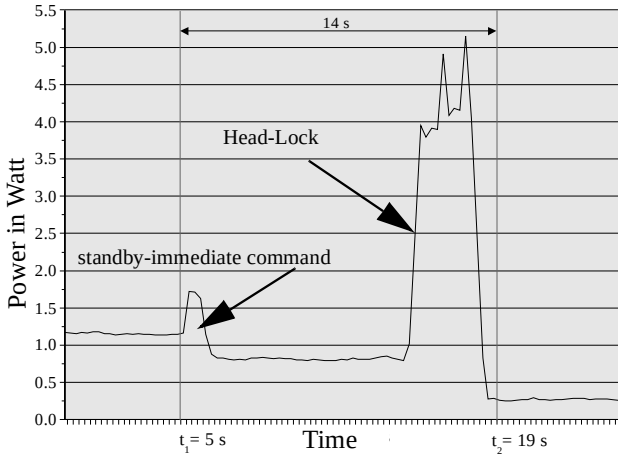


FIG. 4 : Power consumption of an IBM DTNA-22160 disk spinning down.

2.4 Unpredictability of Applications

The previous subsections made clear that we have to know which usage patterns for the various functional units and HW-components a certain software is featuring.

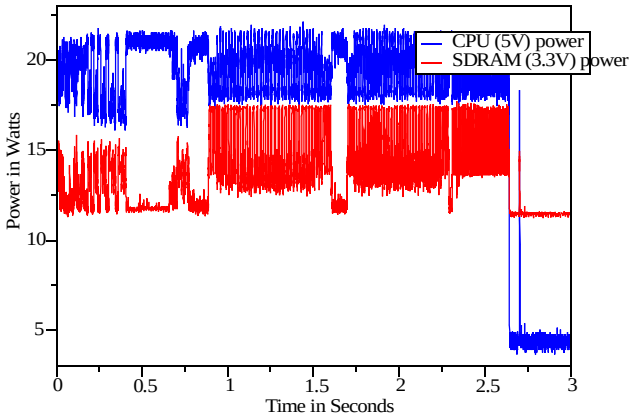


FIG. 5 : Ghostscript: Postscript -> PCL

It would be pleasing if we could characterize an application in advance to give the power management a hint concerning the application's energy characteristics. A first approach was the measurement and classification of applications and parts of applications according to energy-usage patterns [6, 9]. For simple loops with a very regular behavior, this seems to be possible, but a short look (see Figure 5 and 6) at the energy profile of an single real-world application like a postscript interpreter or Acrobat reader eliminates any hope that energy profiles can be predefined in general.

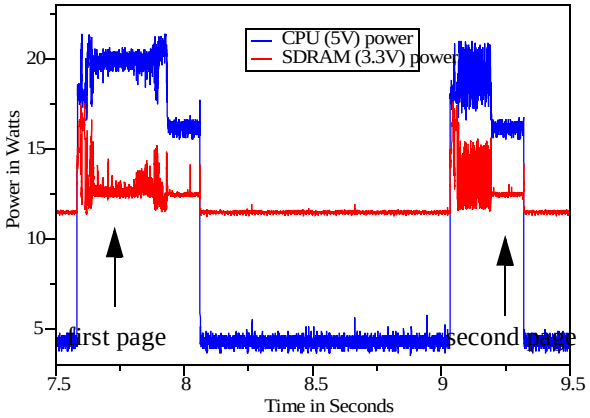


FIG. 6 : AcrobatReader: every page shows different behavior.

The 5V plot for the CPU and the 3.3V power plot for the memory shows, that for a ghostscript interpreter that converts a postscript file into PCL printer code, phases of high CPU demand, change with phases of medium CPU demands but high memory demands. The length, type and number of these phases depends on the content of the postscript file to be converted. The same observation can be done with the Acrobat reader, where the time and energy-curve to present a new page of the PDF-file depends on the content of the page, especially on the fact, how many complex pictures were embedded.

Consequently a predefinition of an application's energy profile is not possible in general.

2.5 The need for on-line energy accounting

As the predefinition of an applications energy profile is not known in advance we need some on-line measurements. By employing a digital multimeter [10] an accurate power analysis is nearly impossible due to the inertia of the voltage regulator associated to the processor and the inertia of the multimeter's measurement unit.

A novel approach to on-line energy profiling should consider the power-related effects of each functional unit without significant influence on the execution of the target system. In section 2.1 we have shown, that the energy consumption depends on the number of activations and the individual characteristics of the HW-components. Our approach to on-line energy accounting uses counters embedded in the target hardware to register events that imply the consumption of a certain amount of energy. **We will show that counter values strongly correlate to a specific energy consumption, so we have found a cheap and easy methodology for power consumption monitoring.**

As activity counters are not found in almost all of the I/O devices and the power-state of these devices can change due to internal device policies, the operating system has no chance to count precisely the number of state transitions. For example many hard disks switch automatically after a time-out of several hundred millisecond from a high-performance idle-mode to a low-performance idle-mode without any notification to the operating system.

Therefore, we recommend to embed some counters to I/O devices like hard disks and wireless network adaptors to support the OS in counting energy expensive transitions of the device's power state.

As event counters are not yet available in I/O devices we will focus in the next sections on the use of event counters found in advanced processor architectures. However, the results are not limited to processors and memory-components. They can be transferred to the field of I/O as well.

2.6 The need for resource containers

Previous work in the field of dynamic power management [12, 24, 28] focus on adjusting the power management parameters in intervals. This approach is motivated by deadline driven real-time strategies with a predefined workload but it does not apply to interactive devices running unknown - maybe dynamically loaded - software, or server systems with workloads that depend on the input data of the client.

Therefore, a novel approach has to support thread-specific or request-specific energy accounting because the energy-specific properties are not known in advance.

Our first approach was system support for thread-specific energy accounting [4] to account and control each thread individually in the time-sharing operating system Linux. Later process specific energy accounting has been successfully applied to the quality-of-service aware operating system Nemesis [23].

At the implementation of process-specific energy accounting in power management (see also section 4.3) we realized that the notion of a thread of control as a resource principal is not adequate for a server thread consuming the resource "energy" while answering requests from different clients. E.g., for a web server of a company, one energy principal might be responsible for the energy used to answer internal requests, while another energy principal is responsible for external requests. With different resource principals for the same group of threads you can differentiate between different operation modes. While a thread might run in the most energy efficient way for answering external requests with lower latency demands, the same thread will run in an energy inefficient way to answer critical internal requests. By supporting the concept of resource containers [2] we can account and schedule a thread according to the container it is acting for.

The abstraction of resource containers separates the notion of a scheduling entity from that of a resource principal. Applied to energy it enables fine-grained energy management in client-server environments allows the partitioning of the system in energy domains with different power management policies.

3 *Joule Watcher* Energy Accounting

The principle of Joule Watcher follows the observation that many of the events which are countable by performance monitoring counters correlate to some activity of the functional units of a processor. An activation of a functional unit should show a characteristic energy consumption. Now the challenge is to find appropriate events and to correlate them to energy values.

3.1 Measurement Methodology

Our initial target environment is a simple Pentium III 866 MHz PC running a Linux 2.4.0 operating system. Context switch routines and kernel data structures are modified to hold the values of the two available performance-monitoring event counters. These counters are realized in the P6 family as registers and can be configured to count one of several events. The accumulated counter values can be accessed though the /proc-file system. To ease the implementation we base on the Performance API (PAPI) from the University of Tennessee [7, 22] that already offers interfaces to configure and read the event counters.

To find the correlation of events and energy values, synthetic micro-benchmarks trigger events of a certain type and frequency for several seconds, while a data acquisition system from National Instruments (DAQ SCXI-1102C) measures the voltage at 1mΩ and 10mΩ 4-pin sense resistors attached to the power supply for the voltages of 5V and 3.3V.

In this way, we measure both the current and the voltage for the two supply voltages and we can determine the power by multiplying both values. While sampling at 20 kHz we get a resolution of 24.5 mW at 5V and 1.6 mW at 3.3V. To synchronize the sampling with special kernel events, we trigger the data acquisition system with rising edges sent by the parallel port of our target system. The data acquisition, streaming to disk, data exploration and integration is done with LabVIEW. The plots are generated with the GRACE tool from the Weizmann Institute, Israel.

We configure the counters to register specific events, run the micro benchmarks and measure the power consumption. So we

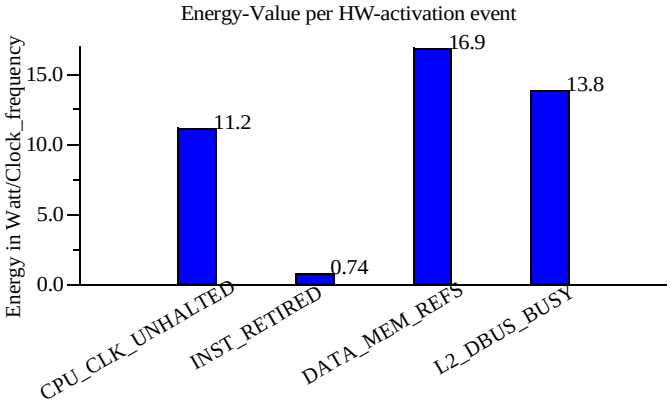


FIG. 7 : Energy values of PIII 866 PGA370 with 256 KB on-die L2-cache.

get a set of equations with event numbers $E_{i,j}$ and total power values P_i.

$$\begin{bmatrix} E_{1,1} & E_{1,2} & \dots & E_{1,n} \\ E_{2,1} & E_{2,2} & \dots & E_{2,n} \\ \dots & \dots & \dots & \dots \\ E_{n,1} & E_{n,2} & \dots & E_{n,n} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \dots \\ V_n \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ \dots \\ P_n \end{bmatrix} \quad (1)$$

Solving this set of equations gives us energy values $V_1..V_n$.

There are some events which contribute significantly to the energy consumption. The main events are the clock cycles, the retired instructions, the memory references leaving the CPU core and the number of cycles during which the back-side bus, conneting the CPU with the second level cache, was busy.

In Figure 7 we have expressed the energy values in relation to the clock-frequency of the CPU in order to make them independent of the CPU speed. An energy value of 16.9 means, that it contributes to a power of 16.9 Watt, if the event happens at the same frequency as the clock speed. If the event just happens every 4th clock cycle, it contributed with 16.9/4 Watts to the power of the processor.

Type of event	Maximum occurrence in% of clock cycles	Energy value	Maximum power contribution
CLK_UNHALTED	100%	11.2	11.2 W
INST_RETIRED	250%	0.74	1.85 W
DATA_MEM_REFS	49%/ 78%	16.9/10.1	8.28 W
L2_DBUS_BUSY	25%	13.8	3.45 W

3.2 Fallacies and Pitfalls

If an event involves the activation of several functional units, we see a high variation in energy consumption, if not all of these units are involved every time, the event happens.

Our first approach was to use several training applications to trigger events and determine the weights using the least-squares error. This methodology is exactly the same proposed in [20]. The training applications (micro benchmarks, but also GNU tools like gcc, make, and ghostscript) were generated by the gcc compiler. The results were very promising and showed small errors for the estimation of other applications. When we added some floating point intensive code (like mpg123) we received negative energy values for the floating point events. As we cannot produce energy by using floating-point instructions, there had to be some “flaw” in our methodology.

We wrote some assembler routines, which trigger specific events with the highest possible frequency and which try to trigger as few types of other events as possible. The first micro benchmarks just makes calculations on registers, another micro benchmark reads/writes data to the first-level cache, then we produce first-level cache misses to trigger the second-level cache and so on. At the end we got a lower-triangle matrix like in equation (2).

$$\begin{bmatrix} E_{1,1} & & & \\ E_{2,1} & E_{2,2} & & \\ \dots & \dots & \dots & \\ E_{n,1} & E_{n,2} & \dots & E_{n,n} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ \dots \\ V_n \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ \dots \\ P_n \end{bmatrix} \quad (2)$$

We built several sets of microbenchmarks. Each set was sufficient to solve the set of equations. We did not compute the energy values using least-squares errors.

Our pitfall was that we assumed that whenever an address is calculated by the MMU to write data, the data is really written to the first-level cache. This assumption does not hold. The store buffer of the CPU acts like a write-back, level-zero cache. If data is read shortly after it was written and the same memory address is later-on the target for a write operation, it seems to be that the data is never transferred to the first-level cache. A write/reclaim to the store buffer consumes less energy than a write/read to the first-level cache. The energy consumption of an application performing frequent push-pop operations to the cache or applications which use the cache very intensively for holding intermediate results, as it typically does x86 floating point code because of the small number of floating point registers, consume therefore less energy for data memory references, because just the MMU is just involved, but not the first-level cache. However, applications working intensively on a small working set which fits into the first-level cache, but not in the small store buffer consume significantly more energy. The difference makes up a factor of 3 in energy consumption per event and can contribute to an error of up to 33% for some code.

Therefore, we recommend solving the set of equations directly with multiple sets of training applications, where training application which triggers the same event should be in different training sets. This procedure does not lead to more precise energy values, but covers up weaknesses in the selection of events or architectural features that are not covered by the available events as we have shown in our example with the data memory references.

3.3 Accuracy of Event-Driven Energy Estimations

With the energy values we derived from the training applications, we estimated the energy consumption of real-world applications, which are mostly compiler generated and do not show up extreme behavior like some assembler routines.

We calculated the energy consumption for all applications just on the basis of 2 events. The Pentium III CPU can count two independent events. However, we have to switch to another event type between application runs, if we want to count more than 2 events. For those applications, where the run of the applications can be repeated exactly, we modified the event types between the runs and counted up 5 different relevant events.
As the behavior of real-world applications in contrast to numerical loops, varies heavily over time, random sampling of the counters with different configured event types [1] does not apply to on-line energy profiling, because the results have to be evaluated within short time (scheduling cycles see section 4) and the application behavior is not constant (see subsection 2.4)

Application	Error in 5V (CPU) using 2 Counters	Error in 5V (CPU) using 5 Counters
ghostscript	-4.77%	4.74%
acroread	8.39%	3.47%
netscape	7.99%	0.08%
mpg123	11.4%	0.09%
make&gcc	-2.05%	3.46
apache 100	5.5%	
apache 200	1.32%	
apache 300	3.46%	
apache 400	-6.6%	

The error-values show the correlation between the extra measurements and the estimation by using event counters. With just the available 2 counter found in the Pentium III hardware, we come close to 10% to the measurements. If more counters would be available, an error of less than 5% is achievable.

The concept of event-driven energy accounting makes an energy estimation possible that comes very close to the real energy consumption. With 2 counters an error of less than 10% is found for compiler generated code. The accuracy is just limited by the number of counters and not by the principle itself.

3.4 Event Correlation for SDRAM

For the main memory we found out that the energy per main memory transaction depends on the frequency of the memory requests. The more we reference the main memory, the more energy it consumes, but also the less energy is necessary per memory transaction. This observation corresponds to the measurements from the university of Delft, which reported that the energy per transferred megabyte depends on the speed of the CPU and, therefore, on the reference frequency as well (see subsection 2.2).

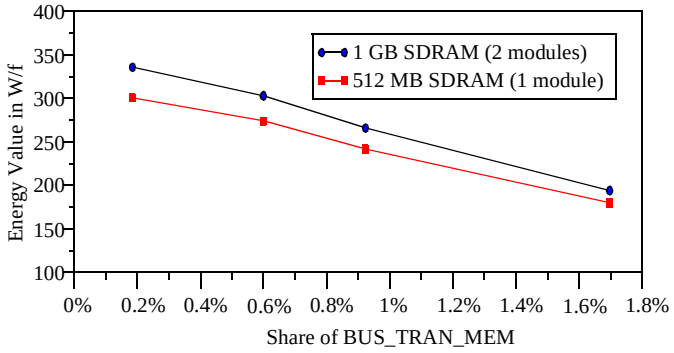


FIG. 8 : Energy values for SDRAM in relation to the transaction frequency.

Beside the static energy consumption of 4.72 Watt per GB of memory (512 MB modules) we see a linear correlation between the energy per memory transaction that we can count by an event counter and the frequency, these transactions happen (see Figure 8).The frequency is here expressed in relation to the clock frequency. A 1% share of memory bus transaction means, that at 1% of the clock cycles we see a memory transaction. By periodically (e.g., in the timer-interrupt handler) calculating the transaction frequency, we estimate the energy values per memory transaction and can then estimate the dynamic energy consumption of the main memory which could reach up to 3.3 Watt.

Applying this model to real-world applications leads to a very precise estimation of the energy consumption with an error of less than 3%.

Application	Error or 3.3 V (SDRAM)
ghostscript	-2.76%
mpg123	1.19%
make&gcc	0.83%

The accuracy of the energy estimation for memory depends on the energy model we apply and should be provided by the designer of the memory architecture. For SDRAM we found a model that estimates the energy with an error of less than 3% by just requiring a single counter.

4 Energy-Aware Scheduling

To demonstrate the benefits of an event-triggered energy accounting, we propose energy-aware scheduling strategies that improve the battery capacity in mobile systems, reduce the energy consumption, and throttle the average power in workstations and servers.

4.1 Improving the usable battery capacity

The operating system can improve the usable battery capacity and consequently, the active life-time of a mobile device by a battery-friendly operation mode. The capacity of a battery is dominated by two factors: the load power and the intermittence of discharge. The charge capacity is the total amount of energy a battery can deliver when discharged at a constant current, called a 1C discharge rate, over a defined period (normally 1 hour)

[19]. The discharge rate has a non-linear impact on the total amount of power a battery can deliver. A typical lithium-ion battery has the following characteristics [13]:

A high discharge rate (e.g., >4C) can lower the usable battery capacity to 70%-80%. Additionally, the usable battery capacity can be shortened by an intermittent load. Duty cycles of 25% can reduce the usable capacity by 40% compared to a continuous load with the same average power [19].

Discharge Rate	Usable Battery Capacity (Normalized to 1C Discharge Rate)
C/5	107%
C/2	104%
1C	100%
2C	94%
4C	86%

Measurement of a typical browser application (netscape) shows an extremely intermittent energy profile which is caused by the fast processing of the arriving data and the high latency of the network which implies a halt of the processor (see Figure 9).

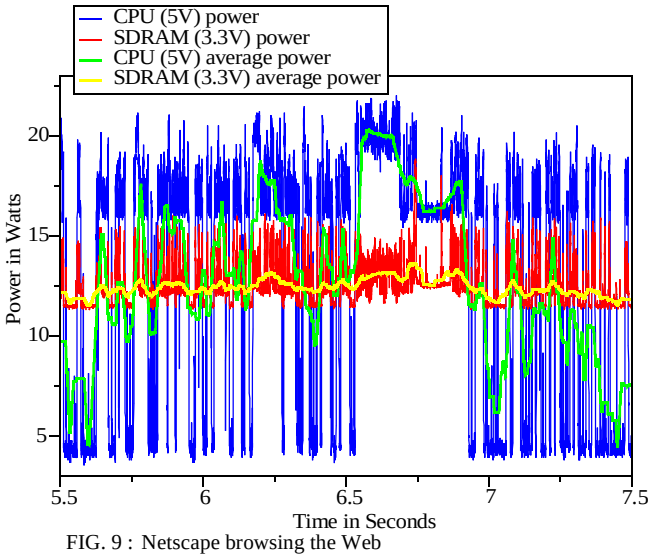


FIG. 9 : Netscape browsing the Web

Assuming a multithreaded browser that can hide network latency by the processing of the data to be displayed, the most battery-friendly execution mode would be a processor that is running so slow that it never has to stop and wait for incoming data. Consequently there are no abrupt changes in the energy profile which are caused by halting and restarting the CPU.
A CPU with variable clock speed and voltage like Intel XScale can be slowed down by the scheduler in order to establish a steady load of a modest level which optimizes the usable battery capacity. The energy accounting has to be aware of the influence of discharge rate and load intermittence on the battery capacity and has to find the best trade-off between battery efficiency and energy efficiency of the consuming devices (see next point).

4.2 Reduction of Energy Consumption

While the motivation for energy reduction is obvious for battery-powered systems, servers are more and more becoming a target for efforts in energy savings due to the rising cost of energy used by computer systems and for energy used by the cooling system. Also becoming important is the cost of floor-space used for racks which can only be populated sparsely with servers to guarantee sufficient air-flow for cooling. On the other side, a denser server population requires a more expensive cooling system.

4.2.1 Affinity scheduling

If the operating system monitors a high number of cache misses after restarting a thread (compulsory misses) it should extend the time-slice of the thread to reduce the number of restarts. Additionally, scheduling strategies can respect the cache affinity of individual threads [5, 29] and improve the cache reuse of threads that use shared memory segments [3] in order to avoid bus transactions and CPU stall cycles due to cache misses. In addition to the improvement in performance all those memory-conscious scheduling strategies mitigate the power consumption of a system.

Looking at the power estimations for current embedded-core generations like XScale we see a significant increase in the amount the memory contributes to the system power.

HW-Component	Peak Power Intel XScale
CPU Core	0.9 W@800MHz 0.45 W@600MHz 0.04 W@150MHz
Memory 32 MB	0.4 W

Event counters which register cache-misses (like the two counters found in the Intel XScale architecture) support memory-conscious scheduling strategies which improve energy efficiency.

4.2.2 Finding the optimal operation point

A high number of main memory references (e.g., bus transactions related to main memory) and a low number of instructions indicates that the speed of execution is dominated by the main memory latency. Without noticeable performance degradation the scheduler can improve the energy efficiency by throttling the clock speed of the processor working on behalf of a latency-bound application. By this means the clock frequency is kept at such a low level that applications run close to their optimal operation point (see section 2.2 and Figure 2) Application measurements have demonstrated [18] that this operation point is application- and parameter-dependant.
Counters that register cache-misses and events related to the processor-internal execution of instructions provide information for an on-line characterization of the currently running applications. This information is essential to run the system at its optimal frequency for minimal energy consumption.

4.3 Throttling of Average System Power

4.3.1 Motivation

Not only the energy efficiency of a system is of importance, but also the temporal distribution of the energy consumption.

- With workstations a high energy consumption becomes annoying if the noise of fans influences the work environment. Therefore, many users ask for a system with passive cooling but low-power components that operate continuously with passive cooling offer the desired performance only at a high price, that is normally justified just for mobile systems.
- The higher the power requirements of a cluster of servers in a high-availability 24x7 environment the higher the cost of installation and maintenance of an uninterruptible power supply consisting of batteries, converters and generators.

Both scenarios have the fact in common that costs arise because we have to care for the rare case of peak energy consumption.
In a workstation environment high computing power is only used at short bursts. Most of the time the system is idle and does not need a noisy cooling system.
For high-available server systems the uninterruptible power supply has to be projected for the uncommon case of peak client load in times of power outages.

4.3.2 Principle of throttling

Our approach to power throttling reduces the costs in both environments. For workstations we employ affordable standard components that normally rely on active cooling like fans. But we throttle the system activity if the average energy use exceeds the predefined power-dissipation capacity of the passive cooling. In servers we throttle the system in the case of a power outage. By this procedure the batteries and the generator of the uninterruptible power supply (UPS) can be configured in much smaller dimensions.

The *Joule Watcher* energy accounting offers information about the energy use with regards to individual threads as well as the whole system. If the average system-energy exceeds the rate that guarantees a safe operation, the CPU is halted by the execution of the HLT-instruction. By halting the CPU for a short period of time the average system-energy diminishes. At the next interrupt another thread is chosen if the average system-energy low enough so that the system is allowed to run again. Threads are ranked according to their priority and energy consumption in the history. Highly interactive threads still can run, but low-priority threads consuming a lot of energy become throttled so that the system can dissipate enough heat or stays below the power capacity of the UPS. Threads that have waited a long time will be preferred because their average power use decays.
We have implemented this policy on a Linux-2.4.0 system. The basic operation of throttling is demonstrated on a compute intensive loop which would consume 17W without throttling. By halting the CPU about twenty times per second we keep the average power below the given limit of 10 W (see figure 10).

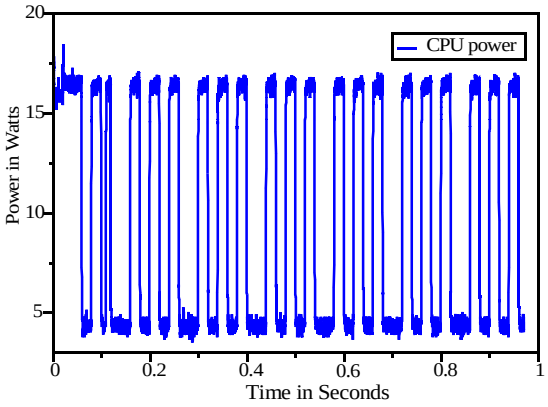


FIG. 10 :Throttling of a 17W applications down to 10 W average power

4.4 Throttling of an apache server

Our scenario for a practical deployment of throttling is an apache web server under load generated by the httperf load generator (see Figure 11). The four plots show the energy profile of an apache which has to establish 100, 200, 300 and 400 connections per seconds and has to serve 5 identical requests at each connection. With the minor load of 100, 200 and 300 connections per second, you can even identify each single request. Because each connection means a cold start for the cache, it is clear to us that the first request consumes more than the remaining 4 requests. However, we can not yet explain the effect that the second requests still consumes more than the third.The average power of an 50ms window rises from 7 Watts to 9 Watts, 11 Watts and finally to more than 15 Watts.

Deploying the Joule Watchers energy accounting together with the power throttling scheduler we see that the server stays below the configured 10W limit (see Figure 12).

While we are very satisfied with the effect of throttling which guarantees a safe operation of a server system even under an emergency power supply, we note that the energy efficiency suffers from our simple throttling approach. While the energy per answered request stays constant in the unthrottled system, we need 25% more energy per request in a intensively throttled system. The reason is the scheduling strategy, that prefers runnable threads that are blocked for longer time. If a thread has to stop, because the average system-power threshold is exceeded, the same thread is not chosen for restart. This leads to many more compulsory cache misses than in an unthrottled system.

Application	unthrottled	CPU Power	Energy per Request	throttled	CPU Power	Energy per Request
httperf 100	500	7.27	0.00574 J	500	7.11	0.00542 J
httperf 200	1000	9.47	0.00507 J	940	9.71	0.00564 J
httperf 300	1445	11.98	0.00524 J	790	9.93	0.007 J
httperf 400	1991	15.03	0.005339 J	832	10.03	0.0067 J

Assuming precise energy accounting, throttling of a system by halting the CPU is a viable approach to keep the average system power below a threshold. However, we have to focus more on energy efficiency so that the system does not suffer from energy-related side-effects of cache-inefficiency.

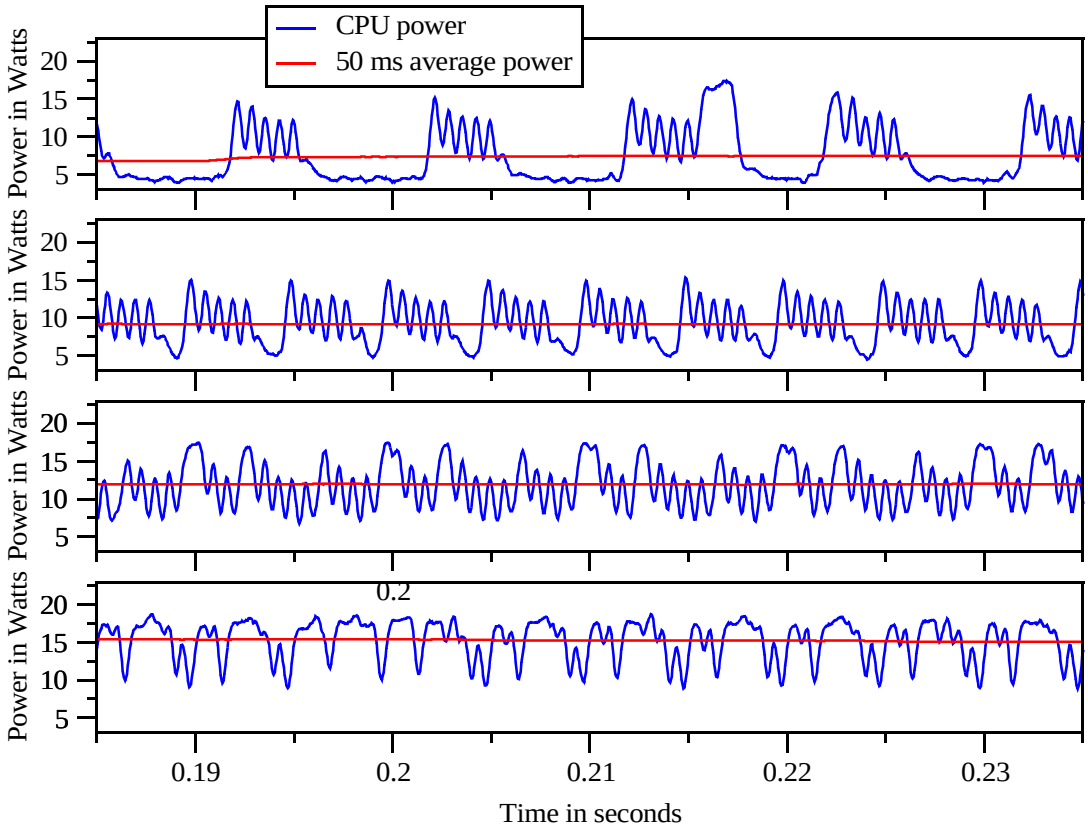


FIG. 11 :Unthrottled Apache (100-400 con/s * 5 requests)

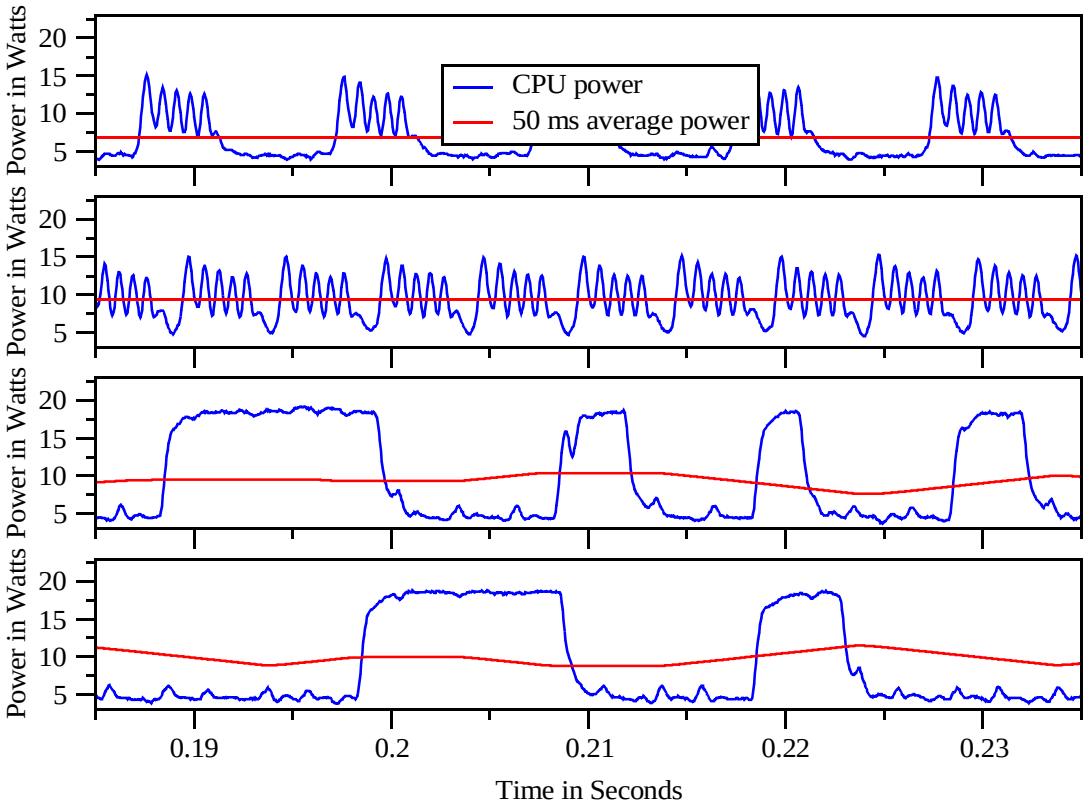


FIG. 12 :Throttled Apache to 10 Watt Limit (100, 200,300, 400 connection_attempts/s * 5 requests)

5 Conclusion

The more the operating system knows what is going on inside the hardware the more it can adapt the execution of threads to the needs of the user. With the emergence of power-sensitive devices, the operating system scheduler has to move from a CPU-centric approach to activity control of all power related components. Event-driven energy accounting and billing to a resource principal is a promising approach to reach this goal.

Our approach to event-driven energy accounting has proved to estimate the thread-specific energy consumption with high accuracy and without any overhead. The current implementation can only use a small number of counters that were intended originally for performance profiling. If the operating system technology is ready to deal with a variety of counters in all locations of the hardware it is just a small step to embed new counters which are exclusively devoted to energy accounting.

Future work will focus on an expansion of the concept of resource containers to distributed and micro-kernel based systems. We want to improve the efficiency of our throttling technique and prove this approach in a data-center environment. Finally, we will improve the energy efficiency by an on-line determination of the optimal operation point as soon as we have a variable speed/voltage systems offering event counter available.

We expect thread-specific speed settings in combination with event-driven energy accounting to become an essential element of future operating systems for power-sensitive devices

Acknowledgements

First, I would like to thank Trent Jaeger, my manager at IBM T.J. Watson, for establishing the valuable contacts within IBM research and to thank the SawMill team for providing a warm and friendly research atmosphere.
Special thanks to Chandler McDowell and Bishop Brock at IBM ARL for their help and advise in power analysis.

References

[1] ANDERSON, J., BERC, L., DEAN, J., GHEMAWAT, S., HENZINGER, M., LEUNG, S.-T., SITES, R., VANDERVOORDE, M., WALDSPURGER, C., AND WEIHL, W. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems 15*, 4 (Nov 1997).

[2] BANGA, G., DRUSCHEL, P., AND MOGUL, J. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation OSDI'1999* (Feb 1999).

[3] BELLOSA, F. Follow-on scheduling: Using tlb information to reduce cache misses. In *Proceedings of the 16th Symposium on Operating Systems Principles SOSP'97, Work in Progress Session* (Oct 1997).

[4] BELLOSA, F. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop* (Sep 2000).

[5] BELLOSA, F., AND STECKERMEIER, M. The performance implications of locality information usage in shared-memory multiprocessors. *Journal of Parallel and Distributed Computing 37*, 1 (Aug. 1996), 1–2.

[6] BENINI, L., BOGLIOLO, A., CAVALLUCCI, S., AND RICCO, B. Monitoring system activity of os-directed dynamic power managament. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'98* (1998).

[7] BROWNE, S., DONGARRA, J., GARNER, N., LONDON, K., AND MUCCI, P. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of the Conference on Supercomputing SC'2000* (Nov 2000).

[8] CHASE, J., AND DOYLE, R. Balance of power: Energy management for server clusters. In *Proceedings of the Eighth Workshop on Hot Topic in Operating Systems HotOS'2001* (May 2001).

[9] ELLIS, C. The case for higher level power management. In *Proceedings of the Seventh Workshop on Hot Topic in Operating Systems HotOS'1999* (Mar 1999).

[10] FLINN, J., AND SATYANARAYANAN, M. Energy-aware adaption for mobile applications. In *Proceedings of the 17th Symposium on Operating Systems Principles SOSP'99* (Dec 1999).

[11] GOVIL, K., CHAN, E., AND WASSERMANN, H. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *Proceedings of the 1st Conference on Mobile Computing and Networking MOBICOM'95* (Mar 1995). also as technical report TR-95-017, ICSI Berkeley, Apr. 1995.

[12] HONG, I., POTKONJAK, M., AND SRIVASTAVA, M. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proceedings of the International Conference on Computer-Aided Design ICCAD'98* (Nov 1998).

[13] INTEL. *Mobile Power Guidelines 2000 Rev 1.0*, Dec 1998.

[14] INTEL. *Intel 80200 Processor based on Intel XScale Microarchitecture*, Nov 2000.

[15] INTEL. *Intel SpeedStep Technology*, Jan 2000.

[16] INTEL, AND ANF TOSHIBA, M. *Advanced Configuration and Power Interface Specification 1.0b*, Feb 1999.

[17] MARTIN, T., AND SIEWIOREK, D. A power metric for mobile systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'96* (1996).

[18] MARTIN, T., AND SIEWIOREK, D. The impact of battery capacity and memory bandwidth on cpu speed-setting: a case study. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'99* (Aug 1999).

[19] MARTIN, T. L. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1999.

[20] MARTONOSI, M. Power-performance modeling, analyis and validation. Tutorial at the HPCA'2001, Jan 2001.

[21] MICROSOFT. Windows power management:instant pc availability and energy savings. White Paper, March 2001.

[22] MUCCI, P. The performance api papi. White Paper of the Univeristy of Tennessee, http://icl.cs.utk.edu/projects/papi/ , March 2001.

[23] NEUGEBAUER, R., AND MCAULEY, D. Energy is just another resource: Energy accounting and energy pricing in the nemesis os. In *Proceedings of the Eighth Workshop on Hot Topic in Operating Systems HotOS'2001* (May 2001).

[24] PERING, T., AND BRODERSON, R. Energy efficient voltage scheduling for real-time operating systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session* (Jun 1998).

[25] POUWELSE, J., LANGENDOEN, K., AND SIPS, H. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the International Symposium on Mobile Multimedia Systems & Applications MMSA'2000* (November 2000).

[26] RAJAMONY, R., BOHRER, P., BROCK, B., ELNOZAHY, E., KELLER, T., AND LEFURGY, M. K. C. The case for power management in web servers. Tech. rep., IBM Austin Research Laboratory, Nov 2000.

[27] TRANSMETA. *The Technology behind Crusoe Processors*, Jan 2000.

[28] WEISER, M., WELCH, B., DEMERS, A., AND SHENKER, S. Scheduling for reduced cpu energy. In *Proceedings of the First Symposium on Operating System Design and Implementation OSDI'94* (Nov 1994).

[29] WEISSMAN, B. Performance counters and state sharing annotations: a unified approach to thread locality. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'98* (Oct 1998).