Frank Bellosa, Simon Kellner,
Martin Waitz, Andreas Weißel

Event-Driven Energy Accounting for Dynamic Thermal Management

Technischer Bericht TR-I4-03-02 ◆ Juli 2003

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

**Friedrich-Alexander-Universität
Erlangen-Nürnberg**

TECHNISCHE FAKULTÄT

Universität Erlangen-Nürnberg • Informatik 4 • Martensstr. 1 • 91058 Erlangen
Tel.: 09131.85.27277 • Fax: 09131.85.28732
E-Mail: i4@informatik.uni-erlangen.de • URL: http://www4.informatik.uni-erlangen.de

Event-Driven Energy Accounting for Dynamic Thermal Management

Frank Bellosa   Simon Kellner   Martin Waitz   Andreas Weissel

University of Erlangen
Department of Computer Science 4 (Operating Systems)

{bellosa, kellner, waitz, weissel}@cs.fau.de

ABSTRACT

With increasing clock speed and level of integration in today's processors, memories, and I/O-controllers, power dissipation is becoming a definitive concern of system design. Control-theoretic techniques have proven to manage the heat dissipation and temperature starting from the level of functional blocks within the processor up to the level of complete systems, so that a thermal emergency will never be reached. However application-, user- or service-specific requirements had to be neglected.

In this work we investigate dynamic thermal management with respect to the demands of individual applications, users or services. We present an event-driven approach to determine on-the-fly the energy consumption on a fine grained level and describe a model to estimate the temperature without the need for measurement. With this power and thermal model—combined with the well-known facility of resource containers—it is possible to throttle the execution of individual tasks according to their energy-specific characteristics and the thermal requirements of the system. In addition to throttling we investigate a modified process scheduler which allots CPU time according to the power contribution of each task to the current temperature level of the processor.

Experiments using a Pentium 4 architecture running a modified Linux show that a given temperature limit for the CPU will not be exceeded while tasks are scheduled according to their energy consumption.

1. Introduction

To meet the insatiable demand for high performance hardware, components with increased gate count and clock frequency were developed. The improvements in manufacturing processes could not keep the power increase at a reasonable level. Thus elaborate and costly package cooling technologies have to be applied. The heat removal mechanism is designed according to the power specification of the components. There exist two design alternatives:

- Cooling technology is designed for *maximum power consumption* in a conservative approach to guarantee that a pre-determined temperature threshold is never exceeded even when running unrealistic workloads with excessive thermal demands.
- Heat removal is designed for *typical sustained power across realistic workloads*. A trigger mechanism is provided to respond with a throttling of activity in order to guarantee a reliable operation of the device. This *dynamic thermal management* (DTM) reduces the cost for cooling by uniformly throttling the system without respect to the importance of the currently running task.

The major shortcoming of existing dynamic thermal management schemes is the neglect of application-, user- or service-specific requirements. The reason is not so much a lack of fast acting response mechanisms but missing online information about the originator of a specific hardware activation and the amount of energy consumed by that activity.

Our approach to dynamic thermal management provides task-specific throttling according to energy-specific characteristics of the task, according to the performance demands of inidividual applications, users or services, and of course according to the thermal requirements of the hardware.

In this work, we present an event-driven energy-estimation model that employs event-monitoring counters to determine on-the-fly the actual power consumption and who has used the power in the system. With the specification of the cooling system (thermal resistance and capacitance), the temperature can be estimated without the need for measurement and used to trigger task-specific throttling. We apply the abstraction of resource containers [3] to account the consumed energy to an *energy principal* and to throttle according to the attributes of the so called *Energy Containers*. Additionally we present a CPU scheduler which identifies and penalizes "hot" processes by reducing their time slices.

Two target application spaces benefit from dynamic thermal management: in the server market, cooling facilities play a significant role in the overall power consumption and costs. Furthermore, cooling facilities are often overprovisioned in order to cope with a cooling unit failure. In the laptop market, dynamic thermal management could make fans obsolete or at least limit the power consumption used for cooling and, as a consequence, achieve longer battery lifetime.

Power and temperature measurements of a Pentium 4 system running synthetic tests as well as real-world applications and benchmarks demonstrate that event-driven dynamic thermal management can handle energy budgets of applications and services while keeping the temperature below a critical limit by constraining the CPU activity.

Section 2 reviews related work. We detail our approach to event-driven energy accounting and dynamic thermal management in section 3. Further we describe our implementation in section 3.4 and present an evaluation in section 4. Section 5 discusses future work and section 6 summarizes our results.

2. Related Work

Power and power density are becoming a major challenge in system design. Not only the power density on the chip level is rising exponentially [12], but also the problems for infrastructure level power supply and cooling [4]. The widening gap between maximum power and typical active power [26] allows two thermal design alternatives:

(1) Heat spreading and cooling is designed for the worst case of sustained maximum power to prevent thermal emergency even under unrealistic conditions.

(2) Thermal design assumes moderate average power thus reducing the cost for packaging and cooling. However, this approach requires dynamic thermal management to influence switching activity whenever a critical temperature limit is reached.

Dynamic thermal management can be divided in three categories:

- *Request management*
  Energy consumption is the consequence of serving a request. Request throttling and load (re-)distribution influence the energy consumption and thermal load of a single system or a cluster. Typically this approach is recommended for internet data centers working on many requests of short service time [23, 18, 19, 10].
- *Direct feedback-driven reduction of chip activity*
  Temperature sensors or on-chip activity monitors determine the thermal state of the chip and initiate a reduction of activity of individual units or the whole chip by reducing their execution rate. This approach was successfully applied to microprocessors by a feedback-driven reduction of the clock frequency or a throttling of the instruction cache [11, 22, 8, 24].
- *Task level throttling*
  CPU intensive tasks are said to be “hot” when they use more than a specific CPU activity over a period of time. When temperature reaches a critical level, hot tasks are candidates for throttling. In this way the system is idling and the CPU spends more time in the low-power state, so the temperature is decreased [21].
  In contrast to direct feedback-driven activity reduction, task-specific throttling does not affect necessarily the performance of important activities like interrupt processing and the execution of tasks that do not contribute significantly to the power consumption of the system.

Performance monitoring counters have proved to offer valuable information in the field of performance analysis [1] and cache-affinity scheduling in multiprocessors [6, 28]. Now they become more and more attractive in the area of power management: the power/performance characteristics of a running thread can be determined on-line by reading of event counters [7, 5]. According to the thread’s patterns the scheduler can find the optimal thread-specific clock-speed in a time-sharing environment to save energy with just minor performance penalties [27].

Precise off-line power estimation is a prerequisite for making architectural decisions and for the design of energy-aware software. Power simulators [9, 25, 13] disclose, how much energy was consumed by a specific component as consequence of the execution of a certain piece of code. Furthermore architectural-level power simulators are useful to calculate thermal plots of the processor die like the Pentium 4 [12]. Thermal plots are essential for the placement of a temperature sensor supporting feedback-driven thermal management.

On the fly power approximation using performance-monitoring counters was quite inaccurate in the beginning [5, 7] as the number of most CPU architectures allowed to measure just two power-relevant events at a time. For long-running user-applications with constant execution behavior, time-multiplexing of several events to the two counters seems to be sufficient for power analysis [16, 17]. However, multiplexing is not suitable for a general purpose run-time power estimation to determine the thermal status of a device.

Managing energy as a first class resource and sharing this resource among competing tasks according to user preferences was introduced in *ECOSystem* [30]. The *Currentcy* model [29] allows to allocate and account energy, and to enforce energy limits. ECOSystem/Currentcy is very similar to our resource container infrastructure and could easily be adapted to support dynamic thermal management.

3. Event-Driven Dynamic Thermal Management

Knowledge of the thermal status of the processor is an indispensable requirement for dynamic thermal management. Chip sets which allow the reading of a thermal diode embedded in modern processors cannot be used for fine-grained management because they don’t allow a correlation between the originator of power consumption and the effect of heating. Furthermore, our experiments revealed that reading the thermal diodes of a typical Pentium4 board imposes significant overhead on the system. It takes 5.5 ms to retrieve the current temperature level via the system management bus (SMBus).

Our approach is based on the performance counters in today's processors to clearly identify “hot” processes, to estimate the processor's power consumption, and to amply determine the temperature of the chip given that the ambient temperature is either constant or can be measured occasionally. Contrary to the measurement approach described above, access to the performance counters is very fast, allowing a process-specific update on energy consumption during every timer interrupt.

Currently, our implementation is constricted to thermal management of the processor, because the rest of the system architecture is not covered by any energy-specific monitoring counters. The presented thermal model is not intended to compete with dedicated power simulators. However it should provide sufficient accuracy to account on-the-fly CPU energy to an energy principal, to determine the thermal status of the processor and to support appropriate throttling mechanisms (e.g. by placing the CPU in HLT state until an interrupt occurs).

The event-driven power and thermal model is combined with the well-known facility of resource containers [3] to throttle the execution of individual tasks according to their energy-specific characteristics, their service requirements and the thermal demands of the system. We call this operating system abstraction *Energy Containers.* The root-container (reflecting the sum of all processes’ energy consumption) is utilized to estimate an upper limit on the processor temperature. In this way a trigger for the system is provided to begin throttling processes. “Hot” processes can be identified using energy consumption information stored in their resource containers.

For measurements we used a P4 2GHz motherboard (ASUS P4B266E, DDR RAM) instrumented with four–terminal precision resistors attached between the board and the 3.3V, 5V and 12V power supply. The voltage drop at the sense resistors was measured with an A/D-converter with up to 40000 samples per second and with a resolution of 256 steps.

3.1 From Events to Energy

The increasing complexity of modern processors (superscalar architecture, out of order execution, branch prediction, ...) demands a more elaborate procedure to estimate on-the-fly the power consumption. While it was sufficient for former architectures like Pentium II to calculate the percentage of CPU activity [21], we registered a wide variation of the active power consumption between 30 W and 51 W for the P4 architecture running a compute intensive task. We measured the power and energy consumption of a set of test programs structured in three groups (see bars in figure 1):

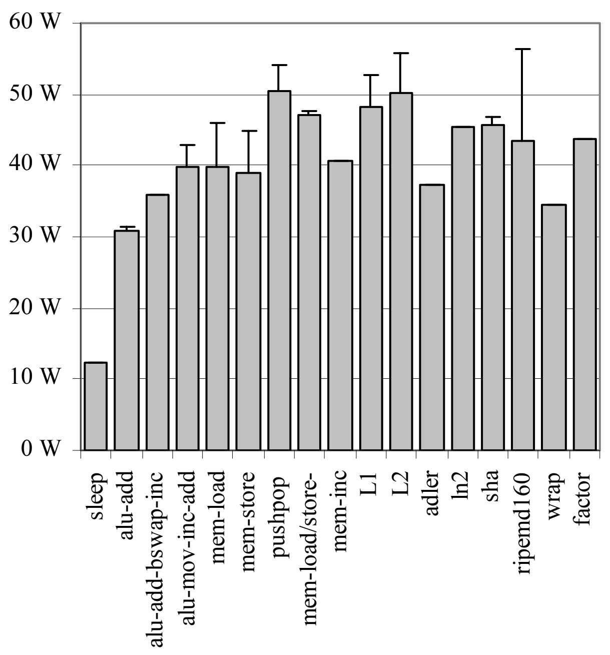

Figure 1: Measured and estimated power consumption

- ALU: programs which operate entirely on registers using ALU instructions like `addc`, `bswap`, `xor`, ...
- MEM: programs which operate on registers and memory (including L1/L2 caches)
- Micro benchmarks which perform various algorithms (`checksum`, `factor`, `heron`, `SHA-1`, `RIPEMD-160`, ...).

Because there are high-power tasks that need about 70% more power than low-power tasks, CPU cycles are no longer a clear indicator for energy consumption.

The next step is to use more processor-internal information provided by the performance counters on-line. Modern processors feature much more performance counters than their predecessors. In particular, the Pentium 4 architecture provides 18 performance counters which can be used simultaneously.

Our approach to energy estimation is to correlate a processor-internal event to an amount of energy. As events being monitored correspond to specific activities, this correlation has linear characteristics. Therefore, we select several events which can be counted simultaneously and use a linear combination of these event counts to estimate the processor's energy consumption.

The event selection was done manually. For each set of events (containing n events), m test programs were run and their consumption recorded.

The data gained from such a test consists of:

- energy consumption for m processes $\bar{e}^T = (e_1, e_2, \ldots, e_m)$
- n performance counter values for each of the m processes: $A = [a_{i,j}]$ $(1 \le i \le m, 1 \le j \le n)$

The problem is to find a vector $\bar{x}^T = (x_1, x_2, \ldots, x_n)$ with $\|A \cdot \bar{x} - \bar{e}\|_2$ minimal and $A \cdot \bar{x} - \bar{e} \ge 0$ so that an underestimation of the energy will not be accepted.

The computation of the coefficients was done using *dqed*, a *netlib* FORTRAN subroutine which tries to find a vector $\bar{x}$ for which the vector $(g_1(\bar{x}), g_2(\bar{x}), \ldots, g_m(\bar{x}))$ has minimum length, with linear constraints on $\bar{x}$. This routine was set up so that a linear combination of events and coefficients was at least as large as the measured energy consumption.

We found quite promising correlations between energy consumption and integer ALU operations, load-/store operations and cache-references. For complex floating point instructions, MMX, SSE, and SSE2 operations our quest for a set of events failed because of a lack of meaningful events. Although these internal events are known and are used in INTEL's architectural-level power simulator ALPS [12], they cannot be counted with the performance monitoring infrastructure of the Pentium 4. Therefore we focused on integer applications to demonstrate the viability of our approach. A further restriction is the fact that first- and second-level cache misses cannot be counted simultaneously, although both are highly relevant events for energy estimation. Most applications show a low 2nd-level cache miss rate, so we decided to ignore the power contribution of 2nd-level cache misses. For some memory intensive applications this can lead to an underestimation of energy consumption of up to 20% (see annotation in section 4.1 and figure 4d).

The final set of events and corresponding coefficients did well for most of the test programs (see table 1). The worst energy estimation is 30% too high, while the estimation was less than 1% wrong for 11 of 25 test programs (see error bars for major categories of test programs in figure 1). The contribution of the time stamp counter constitutes the basic power consumption of the halted processor. Additional energy has to be dissipated if the CPU is active (unhalted cycles) and for each micro-operation fed into the pipeline. Branches contribute differently to the power budget depending on their predictability. Whenever a data memory address is referenced, the memory management unit and the first-level cache become active. The memory order buffer can contribute significantly to the power consumption though this event cannot be seen at a high rate.

3.2 Energy Containers

To manage energy as a first class resource we apply the abstraction of resource containers [3]. These *Energy Containers* are a specialized form of resource containers that can account energy with respect to client-server relations. When a machine is running under energy pressure, processes are throttled according to the limits of the energy containers.

event	weight [nJ]	maximum rate (events / cycle)	power contribution [Watt]
time stamp counter	6.17	1.0000	12.33
unhalted cycles	7.12	1.0000	14.23
uop queue writes	4.75	2.8430	26.99
retired branches	0.56	0.4738	0.53
mispred branches	340.46	0.0024	1.62
mem retired	1.73	1.1083	3.84
mob load replay	29.96	0.4165	24.95
ld miss 1L retired	13.55	0.2548	6.91

Table 1: set of events and their power contributions

Energy consumption of all activities like running processes or I/O requests are accounted to hierarchical energy containers. Energy accounted to an energy container is also accounted to its parent container. Hence, the root container indicates the total energy consumption of the system. If an accountable device is idle, its energy consumption is accounted to the container of the idle task.

The association between processes and containers can be dynamically established by special system calls to reflect changes in the workload of the processes. It is also possible to precisely account the energy consumption of a server to a client on a per-request basis. While a server is reading a new request from a file descriptor, an implicit update of its energy container binding is triggered.

Local clients can attach their resource container to a file descriptor (e.g, pipe, socket). For remote clients a packet filter can classify the packet carrying a request and bind the socket to an energy container depending on the address range from which the client request is coming. This approach propagates resource bindings from clients to server applications.

An energy container is used to control power consumption storing the used energy as well as a limit. We do not limit the amount of energy, but the rate of energy consumption. Thus, time is split up in epochs and a container has an energy limit per epoch. This limit is refreshed according to the current energy policy of the machine. Every activity that consumes energy reduces the available energy of some container. As energy containers are just a special kind of resource containers, they can account for multiple resources. A task is allowed to run if all resources are available (e.g., energy and network bandwidth). Our implementation currently accounts CPU time and energy consumption.

The operating system stops all activities that do not have enough energy in their energy container and enters low-power states to reduce power dissipation. By putting the CPU into a low-power state (e.g., HLT-state) for a short duration of time, it is possible to modulate the processor power consumption. Further potential throttling mechanisms are discussed in [8].

Due to the hierarchical structure of energy containers, there a control loop of one container affecting all containers in the sub-tree. The top-level resource container controls any energy consuming activity in the complete system. By changing the amount of energy that is refreshed in this container, system-wide power consumption can be managed according to thermal requirements.

To implement an energy management policy that copes with different situations ranging from no limit to extreme energy pressure, resource limits of containers can be scaled as the top-level limit changes. That way, a given percentage is always available for the corresponding process. An application of this feature is a packet filter using our energy-aware socket extensions. The filter automatically configures the resource containers of a server process and assigns different energy shares to requests from different client address ranges.

To protect server processes from clients with a very low energy quantum, they always retain their original resource container as backup. As long as the client's container provides enough energy, its budget is used and energy is accounted to the client. If the client runs out of energy, the server, working on behalf of the client, would have to wait until the client's container is refreshed. This would lead to a situation of priority inversion if other clients which have not exhausted their energy budget would have to wait for the server, too. Thus, we modified the resource binding mechanism: if the client's container runs out of resources, the server can be configured to fall back on his original container. Other solutions to the problem of priority inversion exist, but are outside the scope of this paper and will not be presented here.

3.3 From Energy to Temperature

With the processor's energy input known, we are able to estimate the processor temperature by looking at the thermal characteristics of the heat sink. The heat sink's energy input consists only of the energy consumed by the processor, and can be formulated as

$$cm\Delta T = \Delta Q = \int_{t_1}^{t_1+\Delta t} P(t)dt$$

c:	constant
m:	mass of heat sink
P:	CPU power usage
Δt:	elapsed time
ΔT:	heat sink's temperature increase
ΔQ:	difference of inner energy

which is transformed into

$$dT = \frac{1}{cm}Pdt = c_1 Pdt. \qquad (3.3.1)$$

The energy output of the heat sink is primarily due to convection and can be formulated as

$$\Delta Q = \frac{\alpha}{r}\cdot(T-T_0)\cdot t = cm\Delta T$$

r:	thermal resistance
α:	constant
T_0:	ambient temperature

which is transformed into *Newton's Law of Cooling*:

$$dT = -c_2(T-T_0)dt. \qquad (3.3.2)$$

Energy output by heat radiation does not have to be considered because the temperature is quit low (< 60° celsius) and the aluminium surface has a low radiation emitting factor.
Together, these two formulas are used as an approach to estimate the processor temperature:

$$dT = [c_1 P - c_2(T-T_0)]dt. \qquad (3.3.3)$$

Solving this differential equation yields

$$T(t) = \frac{-\tilde{c}}{c_2}\cdot e^{-c_2 t} + \frac{c_1}{c_2}\cdot P + T_0. \qquad (3.3.4)$$

To find values for c_1, c_2 and T_0, we conducted two experiments:

To determine c_2, the raise in processor temperature on a sudden constant power consumption and a sudden reduction to HLT power was measured with the thermal diode on the processor die.

Our thermal model resulting in the differential equation simplifies the real thermal conditions as it assumes a single heat sink interfaced with the chip without thermal resistance. However there is the impact of interface material like grease and phase-change films [26] and the thermal effects of heat spreaders. We found a simple approach to come close to the complex thermal model of several heat spreading components. We assume two constant values for c_2: $c_{2,up}$ for increasing temperature, and $c_{2,down}$ for decreasing temperature for modelling the overlay of the characteristics of interface material, heat spreader and heat sink in case of rising and falling temperatures: $c_{2,up} > c_{2,down}$ and $c_{2,up} \approx c_2 \approx c_{2,down}$.

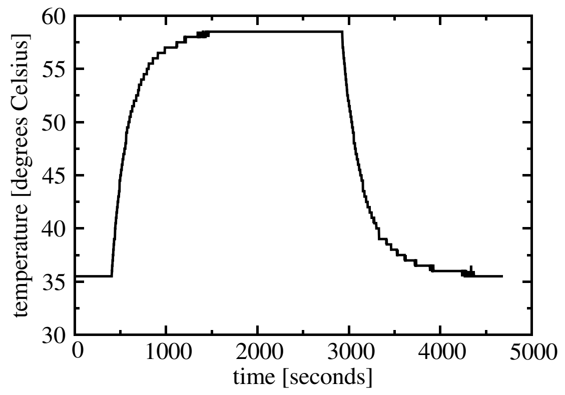

Figure 2: Temperature raise and decay due to constant power

With this approach, the estimated temperature is above the measured temperature in all of the test cases.

For c_1 and T_0, we measured the static temperatures and power consumption of the test programs and interpolated the resulting points with a quadratic function

$$T_s(P) = a_2P^2 + a_1P + a_0 \tag{3.3.5}$$

which has to be above the curve measurements. Otherwise there could be a program for which the real static temperature is higher than the estimated.

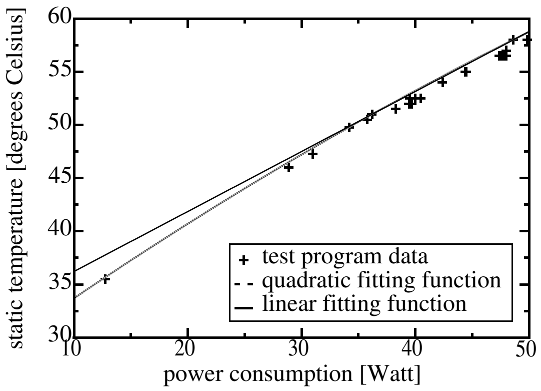

Figure 3: Estimating temperature for static power

The values for c_1 and T_0 are computed from a tangent to $T_s(P)$ in the point $(P, T_s(P))$ with P being the current power consumption.

This results in the following differential equation to estimate the processor's change in temperature:

$$dT = [(a_2P + a_1)P + a_0 - T]c_2dt \tag{3.3.6}$$

with $c_2 = c_{2,up}$ if the last computed $dT \geq 0$, else $c_2 = c_{2,down}$.

Once having measured a representative of the target systems, the four parameters just have to be loaded into the temperature estimation software. This estimator only needs information about the ambient temperature and a continuous flow of power values that can be determined with the help of the event-monitoring counters (see section 3.1).

3.4 Implementation of Thermal Management

With Energy Containers, the kernel used in this work has the infrastructure for accounting and controlling energy consumption of processes and the entire machine. As a result, the temperature estimation and control could easily be implemented in user-space facilitating the use of floating point operations.

The resource container facility features simultaneous energy limits on different time-slices (128 ms and 1024 ms per default). Our approach to temperature control is to compute an energy limit for each time-slice for the whole computer (= root container), based on the current estimated temperature and the temperature limit. By limiting the root container's power consumption, the change in the processor's temperature (specified in equation (3.3.6)) will never result in an overrun of the critical temperature:

$$[(a_2P + a_1)P + a_0 - T]c_2dt \leq T_{limit} - T. \tag{3.4.1}$$

Formula (3.4.1) forms the quadratic inequation

$$a_2P^2 + a_1P + a_0 - T \leq \frac{T_{limit} - T}{c_2dt}. \tag{3.4.2}$$

Because $a_2 < 0$ the solution of this inequation is

$$P \leq \frac{-a_1}{2a_2} - \sqrt{\frac{1}{a_2}\left(\frac{T_{limit} - T}{c_2dt} - a_0 + T + \frac{a_1^2}{4a_2}\right)}. \tag{3.4.3}$$

We extended the utility *mbmon*, which reads the health monitoring chip set and displays the measured temperature, with the temperature estimator and the code to limit the root resource container. This eases calibration of the temperature estimation procedure. The energy consumption necessary for the temperature estimator is read from the root resource container. To prevent a deadlock, the mbmon-process is accounted, but never throttled.

Small errors in the temperature estimation mechanism or errors due to changing ambient temperature will accumulate over time. Me measured an error of 3º–5º C over a period of 24 hours. In order to prevent such deviations the estimated temperature is periodically adjusted to the measured temperature. For this re-calibration a period of 10 to 20 minutes is sufficient.

In order to examine the effects of energy- or temperature-aware process scheduling, we modified the allotment strategy for CPU time of the Linux scheduler. Originally, time slices are computed using the static priorities—the nice-levels—of the processes. We implemented a scheduler which computes time slices according to the relative power consumption of the process compared to the power consumption of the root container. This relation reflects the contribution of the process to the current power dissipation and, furthermore, to the current temperature level of the CPU. Additionally, the priority computation—the decision which process will run next—is based on the relative power consumption. With this approach "hot" processes are disadvantaged by the scheduler.

To sum up, we are able to identify hot processes using energy containers. We present two means to deal with them: first, limiting the power consumption of the attached containers automatically throttles hot processes as they spend their power budget faster than the others. Second, a power-based process scheduler can allot longer time slices to energy-efficient processes. While the second approach does not waste CPU time, throttling is needed to facilitate thermal management.

4. Evaluation

To get an impression of the applicability of our power and thermal models to the "real world" we performed measurements of different applications and application benchmarks.

We attached each test application to a newly created resource container. Thus we were able to isolate the power consumption of the application from the rest of the system.

Using mbmon we recorded the following information:

- the power consumption (derived from the power model) accounted to the application resource container and to the root resource container
- the temperature (derived from the thermal model) of the whole system
- the measured temperature

In addition to that we determined the energy consumption of each test run by recording the accounted energy consumption of the application and root resource containers before and after the test run.

We evaluated our power and thermal model with following applications and benchmarks:

1. perl benchmark
2. Linux 2.5.64 kernel build with gcc
3. caffeine 2.5 benchmark.
4. jvm98 1.03 benchmark
5. Mozilla 1.0.0: Browsing an on-line news magazine
6. MiBench 1.0: An embedded benchmark suite [14]

4.1 Evaluation of the Accuracy of Estimates

4.1.1 Computed vs. measured power consumption

Figure 4 shows the power consumption accounted to the applications, to the whole system (root container) and the measured power consumption. Table 2 shows the estimation errors of the energy consumption for each application and benchmark.

application or benchmark	estimation error of energy consumption
perl benchmark	4.95%
Linux 2.5 kernel build	4.16%
caffeine benchmark	6.09%
jvm98 benchmark	2.20%
Mozilla	-0.56%
MiBench	-1.73%

Table 2: estimation error of energy consumption (positive values: estimation too high)

For real-world applications the energy estimator seems to be quite accurate with an inconsiderable difference between the measured power consumption and the accounting information of the root container (< 10% error). For interactive applications, which are blocked most of the time, the difference between the energy estimation of the root container and the application container is the idle-power (about 13 W) that is accounted to the idle thread.

The jvm98-benchmark contains an interesting example of a program for which the energy estimation model is incorrect: the energy estimation of the database part of the benchmark is *lower* than the measured power consumption (s. figure 4d, time range 25–45 s). A measurement of this routine with various sets of events reveals the difference to the test programs: its number of 2nd level cache misses is higher than any other program in the test set by orders of magnitude. Unfortunately, the Pentium 4 performance counter architecture does not allow a distiction between the 1st and 2nd level cache misses when they are counted simultaneously. As stated in section 3.1, we decided to omit this event in favor of the 1st level cache misses.

4.1.2 Computed vs. measured temperature

For all real-world applications the temperature estimator is within the accuracy of the temperature measurement (< 1° C). The worst-case scenario is an energy estimation error of 30% resulting in an error in temperature estimation of 7° celsius. So the error in temperature estimation is always the consequence of an error in energy estimation. The thermal model presented in section 3.3 did not show any shortcomings.

4.2 Enforcing Energy Limits

The effect of throttling can be illustrated with a set of CPU intensive threads sharing an energy container with a share of 50% relative to the root container. We ran *Hourglass* [20], a tool to visualize scheduling behavior. Figure 5 shows the assignment of energy budgets in

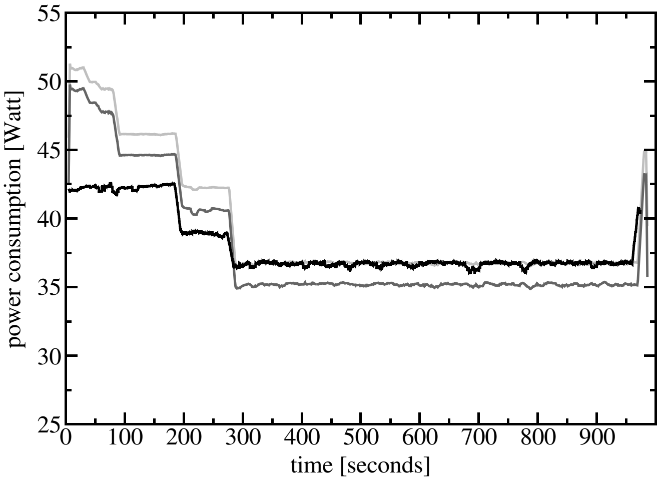

Figure 4a: perl benchmark

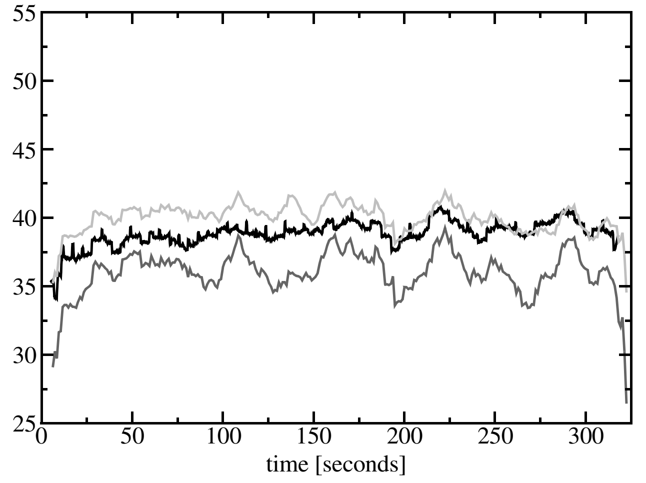

Figure 4b: Linux 2.5 kernel build

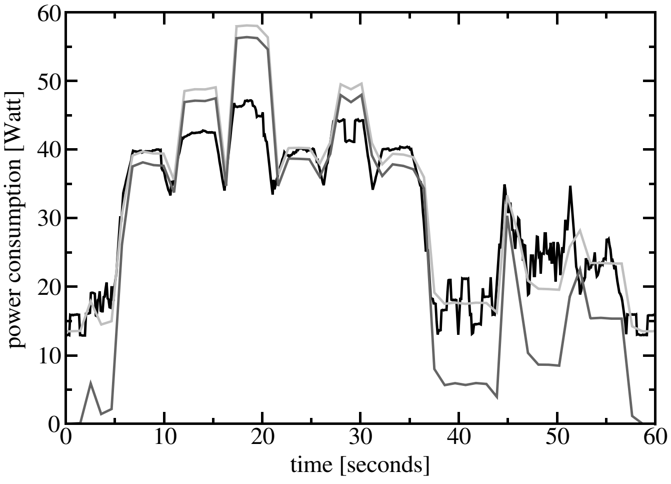

Figure 4c: caffeine benchmark

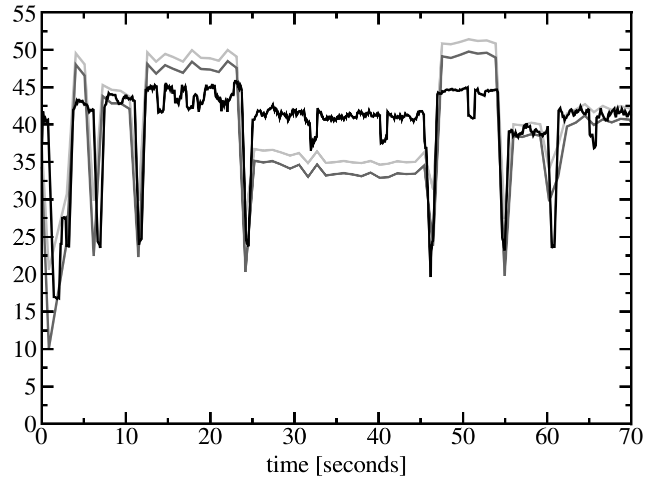

Figure 4d: jvm98 benchmark

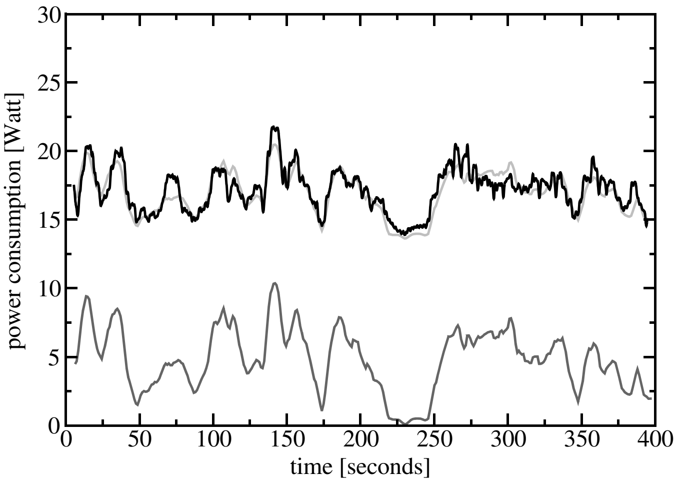

Figure 4e: Mozilla

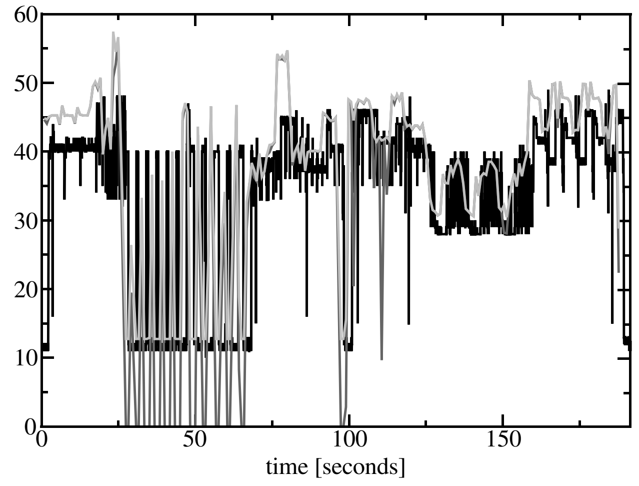

Figure 4f: MiBench

▬ estimated power consumption (total) ▬ estimated power consumption (application) ▬ measured power consumption epochs of 128 milliseconds to an energy container shared by the four threads of Hourglass.

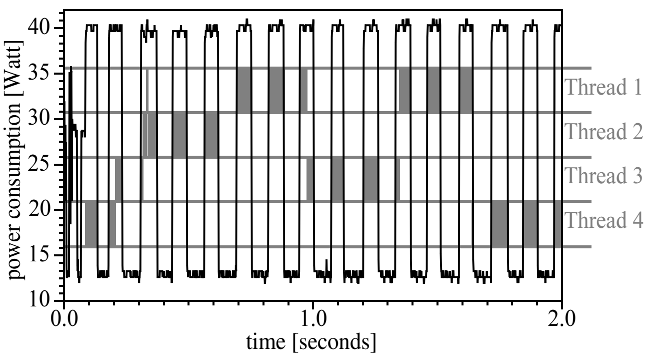

Figure 5: Throttling of 4 threads sharing one resource container with a 50% limit (128ms epochs).

4.3 Enforcing Temperature Limits

Figure 6 shows a test run of the apache web server httpd. Two clients repeatedly send http "GET"-requests to the target machine in order to execute a cgi program (one of the test programs from section 3.1). The shaded area reflects the temperature limit which is set to 50º celsius. Mbmon wakes up periodically and computes the current temperature. The defined temperature level is enforced by setting an appropriate energy limit on the root container. As long as the computed temperature is below the target temperature, no energy limit is set. As can be seen in figure 6 the temperature is rising up to 50º celsius. At time stamp 190 the energy consumption of the root container is limited and the system is throttled in order to stay below the target temperature.

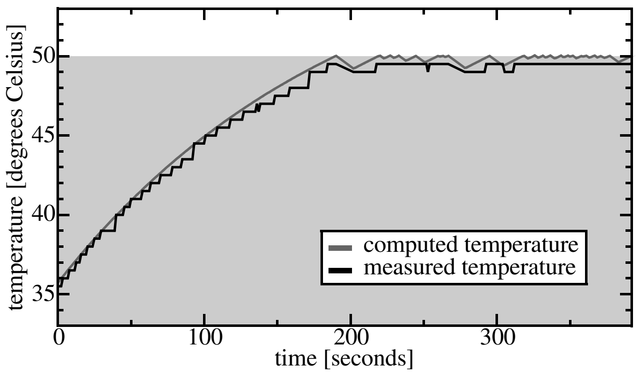

Figure 6: Throttling when reaching a limit at 50º celsius

Figure 7 shows another run of httpd. Initially no temperature limit is defined. Around time stamp 148s the limit is set to 50º celsius (again, the shaded area represents the allowed temperature level). This could be necessary if e.g. a cooling unit in a server cluster fails so that the cluster nodes have to be kept below a certain critical temperature. In our test, the thermal management needs about 110 seconds to reduce the temperature to the new limit.

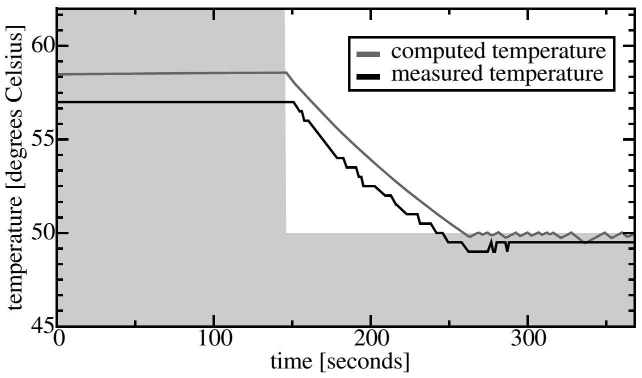

Figure 7: Cool-down with throttling

The proof of concept is a web server accepting requests from two different classes of clients. When a critical temperature limit of 50º celsius is reached, client #1 should be preferred and should get a share of 80% of the allowed total power, while client #2 is just allowed to consume 20% of the remaining power. The power shares can be specified using user space tools which adjust the energy limits of the corresponding resource containers. Figure 8 shows the power consumption of the free running apache tasks working on behalf of the two classes of clients before and after reaching the predefined limit of 50º celsius. The root container reflects the sum of both client containers plus the power consumption of the halted CPU accounted to the idle thread.

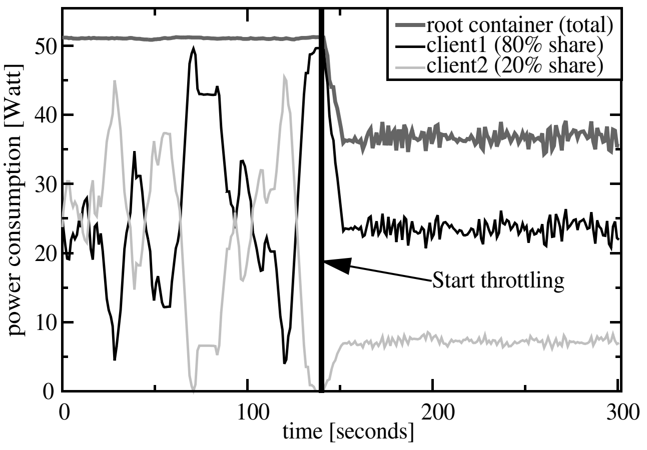

Figure 8: Throttling at 50º according to energy shares

4.4 Energy/Temperature Scheduling

To evaluate our modified scheduler we ran two of the test programs presented in section 3.1: *pushpop*, which shows a very high power consumption compared to the other tests, and *alu-add*, which operates very energy-efficient. These test programs are both compute-intensive and each of them receives, when run on the original Linux scheduler with equal priorities, exactly 50% of the CPU time (as expected). Figure 9 shows the estimated power consumptions. On the left side, the results running the original scheduler are displayed: the contributions of the two processes to the total power consumption are 27 W or 64% for pushpop and 15 W or 36% for alu-add. The energy scheduler changes the assignments for CPU time to 39% and 61% respectively. As can be seen on the right side of the figure, both programs consume the same amount of power. As a consequence of the shorter time slices for pushpop, the overall power consumption is also reduced (by 3W). This example clearly demonstrates that event-driven energy estimation determines hot processes accurately and makes energy-driven scheduling feasible.

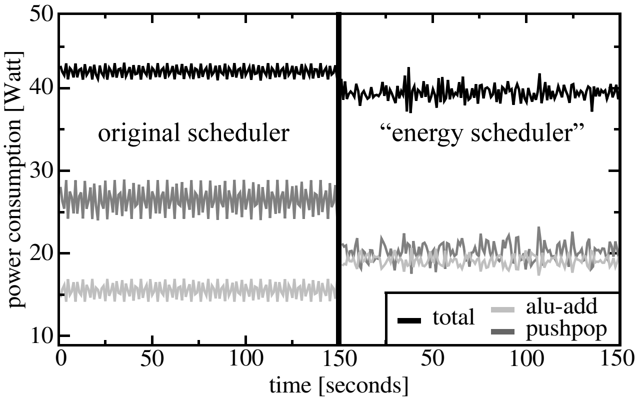

Figure 9: Comparison of different schedulers

4.5 Overhead

4.5.1 Overhead of Resource Container Infrastructure

Reading of the event-monitoring counters is done in the timer interrupt (1000 times per second) or when a task is blocking. The context switching times in Linux 2.5 with energy container support is increased by 49% (5.9 µs) due to algorithmic overhead and the time for reading the event counters. However for a typical scenario like kernel compiling we registered an overall performance loss of less than 1% (the time a kernel compile run needs on the original kernel compared to our modified kernel).

4.5.2 Overhead of Temperature Estimation

Estimating the temperature takes about 4.85 µs with a standard error of 0.843 because of a varying number of cache misses. Setting new limits to the root container requires 12.37 µs with a standard error of 1.537. The overhead for temperature estimation can be neglected because this procedure is typically executed 1-10 times per second. Furthermore, the overhead is by orders of magnitude smaller compared to reading the temperature sensors of the motherboard (which takes about 5.5 ms).

5. Future Directions

There is a multiplicity of interesting opportunities for research in operating systems and computer architecture. Currently we extend our approach to propagate energy accounting information in a server cluster by sending them piggyback in IPv6 extension headers. In this way energy within a cluster will be accounted to an *energy principal (=cluster reserve [2])*. Having accomplished cluster wide energy accounting, the next step is to throttle power consumption according to thermal demands of individual machines and the complete cluster environment and corresponding to quality of service requirements.

If the processor architecture allows a rapid change in clock frequency, task-specific frequency scaling is a further step to moderate the thermal load. Performance monitoring counters will provide the essential information for the power-performance trade-off. The thermal model has to be enhanced to deal with variable speed. Not only the number of events is relevant, but also the clock speed at which the events happen.

The architectural placement of counters and the types of countable events in today's computer architectures are devoted to performance profiling. In a hardware-/software co-design project we investigate the benefit of energy-monitoring counters (ECMs). In contrast to performance-monitoring counters, EMCs cover all energy relevant events. Furthermore the reading of these counters by the operating system is as fast as reading a processor register. The resulting low-overhead has to be paid by a loss in accuracy because we allow a delayed propagation of events to counters. By relaxing the timely resolution of the counters, we accelerate energy accounting and reduce the overhead in energy for managing the energy consumption.

Memory is becoming more and more a target for power management and energy accounting. According to precise energy estimation models for memory [15] we want to develop elaborate energy models for the use in operating systems that employ counters connected with the memory modules. These memory EMCs not only count read and write request but also the number of cycles the clock of the individual memory banks is enabled, and the number of cycles during which the rows in the individual memory banks are open.

6. Conclusions

Events on all architectural abstraction layers are used to simulate the energy consumption of computer systems. Therefore event-monitoring counters should be the adequate source of information for on-the-fly energy accouing. Restricting our validation to integer applications we could demonstrate the benefit of performance monitoring counters for an estimation of energy consumption and for managing the processor temperature. With architectural support for energy monitoring counters, we expect comprehensive event-driven energy accounting to form the basis of coming thermal management strategies.

References

[1] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S.-T. Leung, R. Sites, M. Vandervoorde, C. Waldspurger, and W. Weihl. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems*, 15(4), November 1997.

[2] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *Measurement and Modeling of Computer Systems*, pages 90–101, 2000.

[3] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation OSDI'99*, February 1999.

[4] Christian Belady. Cooling and power consideration for semiconductors into the next century. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*, August 2001.

[5] F. Bellosa. The case for event-driven energy accounting. Technical Report TR-I4-01-07, University of Erlangen, Department of Computer Science, June 2001.

[6] F. Bellosa and M. Steckermeier. The performance implications of locality information usage in shared-memory multiprocessors. *Journal of Parallel and Distributed Computing*, 37(1):1–2, August 1996.

[7] Frank Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, September 2000.

[8] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings Of The Seventh International Symposium On High-Performance Computer Architecture (HPCA'01)*, January 2001.

[9] D. Brooks, Vivek Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings Of The 27th Annual International Symposium on Computer Architecture ISCA-27*, June 2000.

[10] Jeff Chase, Darrell Anderson, Prachi Thakur, and Amin Vahdat. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles SOSP'01*, October 2001.

[11] Christos J. Georgiou, Thor A. Larsen, and Eugen Schenfeld. Variable chip-clocking mechanism. United States Patent 5,189,314, February 1993.

[12] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, 2001. Q1 issue.

[13] S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of The Seventh International Symposium On High-Performance Computer Architecture (HPCA'02)*, February 2002.

[14] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*, December 2001.

[15] Jeff Janzen. Calculating memory system power for DDR SDRAM. *Designline*, 10(2), 2001.

[16] Russ Joseph and M. Martonosi. Run-time power estimation in high-performance microprocessors. In *The International Symposium on Low Power Electronics and Design ISLPED'01*, August 2001.

[17] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam. vEC: virtual energy counters. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering PASTE'01*, June 2001.

[18] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In Luca Benini, Mahmut Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2002.

[19] Ram Rajamony, Mootaz Elnozahy, and Mike Kistler. Energy-efficient server clusters. In *Proceedings of the Second Workshop on Power Aware Computing Systems*, February 2002.

[20] John Regehr. Inferring scheduling behavior with hourglass. In *Proceedings of the 2002 USENIX Annual Technical Conference, FREENIX Track*, June 2002.

[21] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *Proceedings of the 2nd Workshop on Feedback-Directed Optimization*, November 1999.

[22] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez. Thermal management system for high performance PowerPC microprocessors. In *Proceedings of IEEE Compcon'97 Digest of Papers*, February 1997.

[23] Ratnesh K. Sharma, Cullen E. Bash, Chandrakant D. Pateland, Richard J. Friedrich, and Jeffrey S. Chase. Balance of power: Dynamic thermal management for internet data centers. Technical Report HPL-2003-5, HP Labs, February 2003.

[24] Kevin Skadron, Tarek Abdelzaher, and Mircea R. Stan. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *Proceedings Of The Seventh International Symposium On High-Performance Computer Architecture (HPCA'02)*, January 2002.

[25] Phillip Stanley-Marbell and Michael Hsiao. Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*, August 2001.

[26] Ram Viswanath, Vijay Wakharkar, Abhay Watwe, and Vassou Lebonheur. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, 2000. Q3 issue.

[27] Andreas Weissel and Frank Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems CASES'02*, October 2002.

[28] Boris Weissman. 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*, October 1998.

[29] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. Currentcy: Unifying policies for resource management. In *Proceedings of the USENIX 2003 Annual Technical Conference*, June 2003.

[30] Heng Zeng, Xiaobo Fan, Carla Ellis, Alvin Lebeck, and Amin Vahdat. Ecosystem: Managing energy as a first class operating system resource. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'02*, October 2002.