Ereignisgesteuerte Temperaturregelung in Betriebssystemen

Studienarbeit im Fach Informatik

vorgelegt von
Simon Kellner
geboren am 27. Oktober 1978 in Roding

Institut für Informatik,
Lehrstuhl für verteilte Systeme und Betreibssysteme,
Friedrich Alexander Universität Erlangen-Nürnberg

Betreuer:	Dr. Ing. Frank Bellosa
	Dipl.-Inf. Andreas Weißel
	Prof. Dr. Wolfgang Schröder-Preikschat

Beginn der Arbeit:	01. Dezember 2002
Abgabedatum:	30. April 2003

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 30. April 2003

Event-driven temperature control in operating systems

Student Thesis

by

Simon Kellner

born October 27th, 1978, in Roding

Department of Computer Science,
Distributed Systems and Operating Systems,
University of Erlangen-Nürnberg

Advisors:	Dr. Ing. Frank Bellosa
	Dipl.-Inf. Andreas Weißel
	Prof. Dr. Wolfgang Schröder-Preikschat

Begin:	December 1st, 2002
Submission:	April 30th, 2003

Abstract

The development of processors in recent years with its astonishing increase in speed and complexity leads to problems in regard to power consumption and cooling. With shorter clock cycles and increased complexity a modern CPU has to consume far more energy than its simpler predecessors. The processors are reaching the limit of the current standard CPU-cooling method: fan and heat-sink.

From a physical point of view the processor takes electric energy and transforms it to heat which has to be diverted from the processor. Here the development of processors becomes visible: from the early CPUs up to some types of the i486 wich did not need any real heat sink to cool it up to the current Pentium 4 and graphic accelerators where an additional fan is necessary just to increase the heat sink's efficiency. The upcoming discussion on better cooling solutions like heat pipes, water cooling and so on also indicates a growing need for thermal processor design.

It is getting difficult in air conditioned server farms with racks containing several computers on very small space to increase the density of processors. The air conditioning required when using current CPUs is challenging, so limits on power consumption per rack are introduced to prevent an overload of the installed air conditioning units.

Modern processors and motherboards allow the measurement of the CPU temperature. This is a prerequisite of temperature control, either in hardware like the Pentium 4 or various software methods. All of these methods have drawbacks in either affecting all processes simultaneously or not being applicable to current processors.

This work provides a process-specific approach to temperature control utilizing performance counters to account the energy consumption of processes, which can be used to compute the temperature of the processor.

This thermal model allows the computation of a dynamical power consumption limit for the machine, the enforcement of which keeps the processor temperature lower than a set limit. Combined with the facility of resource containers this approach features flexible task specific throttling while maintaining a temperature limit.

Contents

Chapter 1

Introduction

Increasing complexity in current processors, especially “desktop”-processors, has its drawbacks in form of increasing power consumption. As a result, processors are reaching the limit of today’s standard CPU-cooling solution. This calls for more elaborate and expensive cooling methods and for a more thermal oriented processor design.

For the latter there exist two alternatives: Either heat spreading and cooling are designed to handle the unrealistic case of maximum sustained power consumption. Or they are designed to handle the average expected workload allowing less expensive packaging and cooling solutions. In the latter case an additional mechanism called *dynamic thermal management* has to prevent sustained high workloads.

1.1 Related Work

There are various different approaches to dynamic thermal management working at different levels of granularity. They can be divided into three categories:

Request management relies on the assumption that energy consuption is primarily caused by requests sent to a computer, like load-balanced web-servers at Internet data centers. Temperature control of the cluster and a single machine is done via throttling of the request rate and distribution of requests [15, 11, 12, 5].

Direct feedback-driven reduction of chip activity is implemented, for example, in the Pentium 4 [9] (called “Thermal Monitoring”). Here a thermal diode embedded in the processor can trigger a throttling mechanism on reaching a fixed temperature. Throttling is done by skipping clock cycles and affects all processes running on the processor equally, as well as interrupt latency [6, 14, 4, 17].

Task level throttling as described in [13] only throttles “hot” processes, that is, processes which consume more than a certain percentage of CPU-activity per time-interval. It is a reactive system which relies on periodical measurement of processor temperature to trigger the selective throttling mechanism.

1.2 Contribution of this Work

This work presents an approach in the category of task level throttling: The usage of performance counters allows an energy consumption estimation at process level. In combination with a thermal model the overall energy consumption of the processor and knowledge of the ambient temperature is sufficient to estimate the processor temperature. Reversal of the thermal model and a given CPU temperature limit yield a dynamic power consumption limit for the processor. This limit can be enforced using a specific form of resource containers, energy containers, which allow fine grained control over the throttling of processes based on the information on task level energy consumption.

In contrast to the request management method this work’s approach is applicable to a single computer. However, no major difficulties are expected in applying this method to a cluster of machines.

The differences betweeen this work and the thermal monitoring method are the granularity level and the usage of energy consumption estimation instead of direct measurement. Temperature control in hardware lacks user control, and CPU temperature measurement with software is rather slow with current chips and does not provide information on the source of power consumption due to the large measurement intervall.

Being in the same category, there are less differences between the dynamic thermal management presented in [13] and the event-driven temperature control in this work. However, the definition of “hot” processes has to be adjusted as power input and CPU-activity are no longer directly correlated. This is shown by the test programs used in this work. The approach taken in this work is also a bit more proactive by computing energy limits for the next time interval and features more flexibility on process throttling thanks to the concept of resource containers.

Chapter 2

Energy Estimation

Energy estimation methods of most devices use information about how long a device has been in a certain state, associating a constant power consumption with each state. With modern processors this scheme is not applicable. Power consumption running several test programs ranges from 30W to 50W with a CPU-activity of 100% each.

Therefore more information has to be taken into account. Such information is provided by performance counters which are able to count various processor-internal events. Compared to its predecessors, the Pentium 4 processor features an extensive set of events which can be monitored with performance counters (PCs).

2.1 Approach

The basic idea is to find a subset of events that refer in some way to the energy consumption of the processor. Additionally it should be possible to monitor this event subset simultaneously, to allow an on-the-fly estimation of energy consumption.

One problem is obviously the way in which the events refer to energy consumption. In this work a simple linear approach is used. This choice was made mainly for practical reasons: Since such linear problems are well understood, there already exist many programs which solve a variety of these problems. Therefore it is sufficient to formulate this linear problem and find a program to solve it.

Besides, a linear approach is quite plausible. If the occurence of an event indicates some amount of energy consumption directly, this correlation has certain linear characteristics. Even dependencies can be taken into account: If event *a* induces event *b*, and "*b* with *a*" and "*b* without *a*" indicate different amounts of consumed energy, the correlation's linear properties remain.

The other problem lies within the performance counter architecture itself. For performance reasons Intel decided to place the counters near the various units which can be monitored. Thus, an event can only be counted in a counter lying next to the unit responsible for this event.

This results in 4 groups of PCs and events (3 groups with 4 PCs and 1 group with 6 PCs), meaning that no more than 4 or 6 interesting events can be monitored simultaneously per group, reducing the number of possible event-PC assignments. Since the “interesting” events are not distributed evenly among the 4 groups, this grouping is problematic. Additionally, for each event in a group there are only 2 or 3 PCs that can be used to count this event. There exist some special events with even more specific requirements reducing the choice of events further.

2.2 Test Configuration

2.2.1 Event Counting

For event counting the programs `brink` and `abyss` [18] were used in the beginning of this work. `brink` is a rather large Perl-script which takes a well formed XML description of the Pentium 4’s performance counting architecture and a description of an experiment consisting of a set of test programs and a set of events, again in well formed XML. It then assigns the events to PCs and passes this information to the `abyss` program which takes care of PC initialization and event measurement of one test program. Both programs have a disadvantage, though.

The event assignment method of brink is inconvenient, so it was patched to produce far more possible assignments than the original implementation, although not in every case, especially with some tagging events.

The measurement method used by abyss includes the startup code from the dynamic linker in the measurement, which is not insignificant in case of programs running only for 1-3 seconds. In later tests, the PC setup and measurement was done within the test programs themselves, including the setup and measurement code using a preprocessor. Included in this code are instructions to trigger a signal for the A/D-converter to synchronize the beginning and end of measurement.

2.2.2 Energy Consumption Measurement

The motherboard of the Pentium 4-computer (ASUS P4B266E) uses the 12V-sources from the power supply for the processor exclusively. By introducing an accurate low measurement resistor between the power supply and the processor (figure 2.1), measurement of the current power consumption of the processor (P_{CPU}) becomes possible:

$$\begin{aligned} U_{R_1} + U_{CPU} &= 12V \\ I &= \frac{U_{R_1}}{R_1} \\ P_{CPU} &= (12V - U_{R_1})\frac{U_{R_1}}{R_1} \approx \frac{12V \cdot U_{R_1}}{R_1} \end{aligned}$$

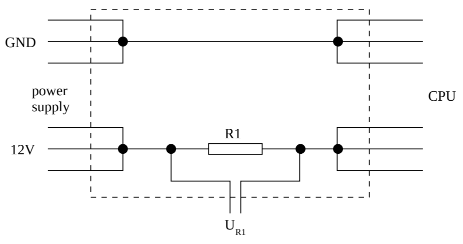

Figure 2.1: The measurement resistor

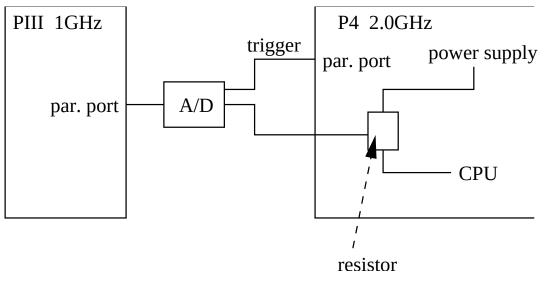

Figure 2.2: The measurement assembly

The A/D-converter is able to generate 5000 samples per second, retrieved by a second computer running at 1 GHz. A device driver in the second computer attaches to each sample the current value of the time-stamp counter.

The energy consumption of a test program is computed by integrating numerically over all power samples with a set trigger bit.

2.3 Test Programs

Due to the limitations of the performance counter architecture it is unlikely that a general energy model can be found quickly. In this work, the focus was on “normal” programs: C-programs that contain little or no architectural dependent code which is often found in highly optimized programs. The test programs consisted of three groups:

- ALU — programs which operate entirely on registers and use only ALU-instructions like `adc`, `bswap`, `xor`, . . .
- MEM — programs which operate on registers and memory (including caches, read/write instructions)
- other programs which perform various algorithms for a few seconds (checksum, factor, heron, SHA-1, RIPEMD-160, . . .)

As stated before, the programs take care of proper initialization of the counters, measurement of the events and of triggering the A/D-converter.

2.4 Methods of Analysis

The selection of an event subset which correlates to energy consumption was done manually. For each of these subsets (containing n events), the m test programs were run and their energy consumption recorded. The data gained from such a test consists of:

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

The remaining problem is to find a vector $\vec{x}^T = (x_1, \ldots, x_n)$ with these properties:

$$A \cdot \vec{x} - \vec{e} \geq 0, \qquad ||A \cdot \vec{x} - \vec{e}|| \quad \text{minimal}$$

In other words, a scalar multiplication of a performance counter vector (PCV) with a vector $\vec{x}$ should be at least as large as the energy consumption of the corresponding process, but not much larger.

Based on the accuracy of each estimation, the subset of events was extended or reduced. It was reduced in cases where the accuracy without an event was better than with or if the corresponding coefficient x_i was not stable in another test with the same event subset. The idea behind the last criterium is to remove events the analysis programs tended to use as "dummies": events that don't have anything to do with energy consumption but the analysis program uses anyway to produce a better result.

Indications of such "dummy events" are: a corresponding coefficient which is exceptionally larger than the others, and fluctuation between two similar tests (which produce slightly different energy and performance counter values).

This procedure was started with the set {`time stamp counter`,`global power events`}. `time stamp counter` is not a real performance event. It was included because a halted Pentium 4 does not trigger performance events, but consumes approximately 12.5 Watts in this state. Additionally, measurement of this event does not allocate precious performance counter resources. `global power events` is the most basic event related to energy: it counts the number of clock cycles the CPU has been active.

2.4.1 Analysis with `lrslib`

What is left to describe are the analysis programs used. The first one, `lrslib` [2], is included here because it was used for a long time and helped compose a first working event subset.

`lrslib` is a library which solves linear programming problems. The description of these problems is nearly identical to our problem here which is why `lrslib` was the first program used in this work for analysis. `lrslib` was chosen in particular because it can be linked against the GNU mp library (`gmp` [7]) for arbitrary precicion arithmetics.

Given a (m,n)-matrix A', a vector $\vec{b}$ and a *linear* function $g(x_1,\ldots,x_n)$, `lrslib` finds the minimum of g on the set of points $\vec{x} \in \left\{\vec{x} \in \mathbb{R}^n \mid A' \cdot \vec{x} - \vec{b} \geq 0\right\}$. The only difference to our problem is the restriction on g to be linear.

With m test programs and n events, the input to `lrslib` was constructed as follows:

- $\vec{b} \leftarrow \begin{pmatrix} \vec{e} \\ -(1+\beta)\vec{e} \end{pmatrix}$
- $(2m,n)$-matrix $A' = \left[a'_{i,j}\right]$ with $a'_{i,j} = \begin{cases} a_{i,j} & \text{, if } i \leq m \\ -a_{(i-m),j} & \text{, if } i > m \end{cases}$
- $g(x_1,\ldots,x_n) = \sum_i x_i$

The duplication of $\vec{e}$ and A and the parameter β need some explanation: A' and $\vec{b}$ form a vector of $2m$ equations. The purpose of the first m equations is to establish the measured energy values as a lower bound for energy estimations. The other equations make sure the estimation is not too high. More precisely, the relative error of the estimation does not exceed β.

No attention was paid to g, because with a minimum β the search space for a possible minimum of g becomes very small, and vectors $\vec{x}$ obtained with different functions for g do not differ much.

The analysis program was run several times with $\beta_0 = 0.5$ as a starting value. The minimum β was found using binary search. While not being the optimal analysis program for this problem, the values obtained from it are quite satisfying.

2.4.2 Analysis with dqed

The program used to find the final set of events and coefficients is `dqed` [8], a FORTRAN routine found at netlib [1], a large collection of mathematical software. It does a good job trying to solve the following problem:

For functions $g_1,\ldots,g_m : \mathbb{R}^n \to \mathbb{R}$ (not necessarily linear), find a vector $\vec{x}$ minimizing the length of the vector $(g_1(x_1),\ldots,g_m(x_m))^T$. Additional linear and slightly non-linear constraints can be placed on the vector $\vec{x}$, like $2x_1 - 3x_3 \leq 5$.

The advantage of this method over `lrslib` is the minimization of a vector length, that is, the sum of squares, rather than only the sum of the elements. This ensures an approximately equal fitting to the m test programs.

The input for `dqed` was computed as follows:

$$\begin{array}{ll} g_i(\vec{x}) = \sum_{j=1}^{n} a_{i,j} x_j - e_i & (1 \leq i \leq m) \\ \text{constraints:} \quad g_i(\vec{x}) \geq 0 & (1 \leq i \leq m) \end{array}$$

The analysis program was run several times, with randomized starting values for $\vec{x}$, providing a little bit more accuracy than a fixed starting point. The resulting final events and their corresponding coefficients (weights) are:

event	weight [nJ]	maximum rate (events per cycle)
time stamp counter	6.17	1.0000
unhalted cycles	7.12	1.0000
μop queue writes	4.75	2.8430
retired branches	0.56	0.4738
mispred. branches	340.46	0.0024
mem retired	1.73	1.1083
mob load replay	29.96	0.4165
ld miss 1L retired	13.55	0.2548

2.5 Evaluation

With both analysis methods, the accuracy of the estimation differed with the programs. The 25 test programs finally could be partitioned into several groups:

- 8 programs with nearly perfect estimations (approx. 10^{-3}% relative error),
- 3 programs with very good estimations (approx. 1% relative error),
- 9 programs with a relative error below 10%,
- 4 programs with a relative error below 20%, and
- 1 program with a relative error of 29%.

The last program shows that a linear combination of performance counter events is not generally applicable to energy estimation with current restrictions on event counting.

Figure 2.3 shows the real power consumption of 17 test programs, the error bars indicate the estimated power consumption. Some test programs were omitted because of similarities to the shown programs.

The energy estimator was also tested with a set of real-world applications, consisting of:

- 3 benchmarks: perl-bench, jvm98 and caffeine 2.5,
- 2 interactive programs: Mozilla 1.0.0 and OpenOffice 1.0.2, and
- a compilation of the Linux 2.5.64 kernel.

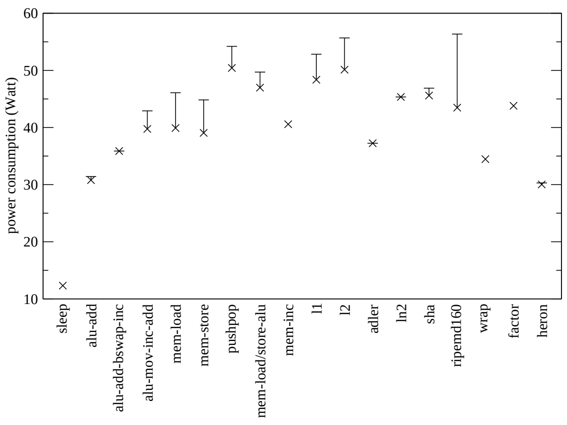

Figure 2.3: Overall relative errors of energy estimation of the test programs

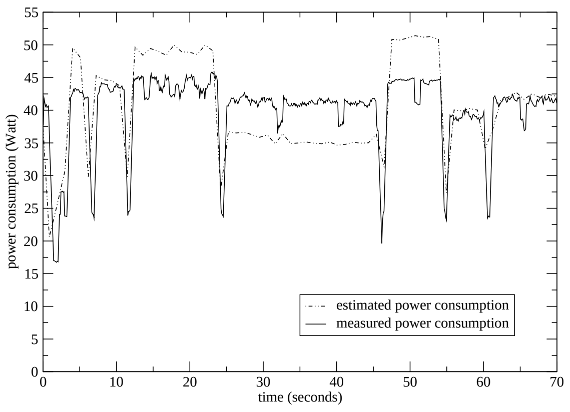

Figure 2.4: Estimated and real power consumption of jvm98

The resulting energy overestimation is:

application	estimation error of energy consumption
perl-bench	4.95%
Linux 2.5.64 kernel build	4.16%
jvm98 1.03	2.20%
mozilla 1.0.0	-0.56%
OpenOffice 1.0.2	-0.69%
caffeine 2.5	6.09%

The test shows that this energy estimation procedure could be applicable to real-world programs if the selection of simultaneously countable events was less restrictive. A closer look on the run of jvm98 (figure 2.4) illustrates this. jvm98 consists of several sub-benchmarks, and one of them, the database benchmark, consumes considerable more energy than estimated.

A measurement of this benchmark with various sets of events reveals the difference to the test programs: its number of second-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 distinction between the first- and second-level cache misses when they are counted simultaneously. Since the correlation of the second-level cache misses to the energy consumption of the test programs is only marginal, this event was omitted in favor of the first-level cache misses.

Chapter 3

Temperature Estimation

In this chapter the attention turns from energy consumption estimation towards temperature estimation. The goal is to compute the CPU temperature using data from the energy estimation. An additional restriction on this computed temperature value is to never be less than the actual processor temperature.

The CPU is treated as a black box with energy input and output. The energy input consists only of the electrical energy being consumed. For energy output there are several ways available to the CPU: heat radiation and convection.

3.1 Approach

Heat radiation is not considered in this work, because its effects become noticable with high temperatures only, whereas the temperature of the CPU in this work ranges from 20°C to 60°C. Additionally the aluminium heat sink was not designed to emit a large part of heat energy via radiation. The remaining methods of energy transportation are: The input of electrical energy and the output of heat energy in form of convection.

The energy input can be formulated as $dT = (cM)^{-1} \cdot dQ$, and the two constants cM are then merged into c_1. The energy output is basically formulated as $dT = \kappa A(T - T_0)dt$, where T_0 is the temperature of the air surrounding the heat sink and κ is a material-dependent constant. Again, the constats κA are merged into another constant c_2. Together, these two equations form the basic approach to temperature estimation taken in this work:

$$dT = [c_1 P - c_2(T - T_0)]\,dt \tag{3.1}$$

The ambient temperature T_0 is assumed to be constant throughout the thermal models. This is a rather annoying problem in a non air-conditioned room which is subject to temperature fluctuations. Therefore every thermal model provides a parameter for adjusting to the current ambient temperature.

3.2 Test Configuration

The only requirement for creating a temperature estimation model for a CPU is the measurement of both power consumption and temperature of the CPU. For power consumption measurement, the resistor from section 2.2.2 was used.

The temperature was read from the CPU's internal temperature-diode via the motherboard's health monitoring chip. The allowed resolution of this chip regarding processor temperature is only 0.5°C. To increase the temperature range of the CPU, the heat sink's fan was slowed down, allowing the CPU to reach 60°C.

`lm_sensors` [10], a program to read data from this type of chip, detects a wrong chip for which it has no support, due to lack of documentation from the manufacturer. A similar program, `mbmon` [16] was able to read from this chip, but only after a kernel module from `lm_sensors` had activated it. This incident reveals a situation where temperature estimation with performance counters could be needed: if the chipset is undocumented, deactivated or damaged.

Later `mbmon` was extended by the temperature estimation model, providing synchronized pairs of real and estimated temperature values. As a normal userspace program, it also has the ability to use floating point arithmetic.

3.3 Models

3.3.1 Initial Model

The first temperature model tested is the one described in equation 3.1. It is included here because it shows the steps necessary to find the model parameters and it is the common ancestor of all the other models. The yet unknown values c_1, c_2 and T_0 are treated as constants:

$$dT = [c_1 P - c_2(T - T_0)]\,dt \qquad c_1,\, c_2,\, T_0 \text{ const.} \tag{3.2}$$

The variables P and dt are available via the energy estimator from the previous chapter and the time stamp counter of the Pentium 4. T can be initialized with any value and adjusts itself to the real temperature if the constants and the estimated input power are correct.

Equation 3.2 is a simple form of a differential equation and is easily solved:

$$dT = [c_1 P - c_2(T - T_0)]\,dt$$

$$\begin{aligned}
\frac{dT}{c_1P - c_2(T - T_0)} &= 1dt \\
\frac{-1}{c_2}\int \frac{-c_2}{c_1P - c_2(T - T_0)}\,dT &= \int 1\,dt \\
\frac{-1}{c_2}\ln\left(|c_1P - c_2(T - T_0)|\right) &= t + C \\
c_1P - c_2(T - T_0) &= e^{-c_2(t+C)} \\
T = \frac{-1}{c_2}\left(e^{-c_2(t+C)} - c_1P - c_2T_0\right) & \\
= \underbrace{\frac{-\tilde{c}}{c_2}e^{-c_2t}}_{\text{dynamic part}} + \underbrace{\frac{c_1}{c_2}P + T_0}_{\text{static part}} & \qquad (3.3)
\end{aligned}$$

So the process of adjusting to a new temperature determined by the input power is exponential and can be compared to charging and discharging a capacitor. Obviously only c_2 has influence on the speed of the adjustment, while the resulting temperature depends on P, c_1, c_2, and T_0 and is linear in P.

The first constant to be measured is c_2. For this, the temperature increase of the processor from a halted state to continuously running one of the test programs was recorded using `mbmon` (figure 3.1), and then, with `xmgrace`'s ablility of "non-linear curve fitting", an exponential function was fitted to the data. The coefficient in the exponent was taken to be the value of c_2.

For c_1 and T_0, the various static temperatures produced by running different test programs and their corresponding power consumption were measured (figure 3.2). Fitting a linear function to these values makes it possible to compute the values of c_1 and T_0 by comparing coefficients with the static part of equation 3.3. This linear function has to be fitted above all these values to avoid an underestimation of CPU-temperature. For adjustment to the current ambient temperature the model parameter T_0 can be used.

This model basically works, but it is inaccurate either at the idle or maximum power consumption, depending on the linear fitting function.

3.3.2 Dual Model

The next model tried has two different heat sinks: the heat sink on the CPU and the air in the computer case forming a secondary heat sink. It is based on the observation that the ambient temperature does change slightly (and slowly) between idle and maximum power consumption, according to the health monitoring chipset.

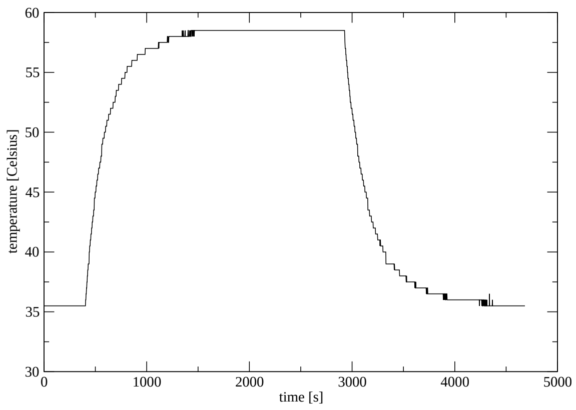

Figure 3.1: CPU temperature on a constant power input (idle and 50 Watt)

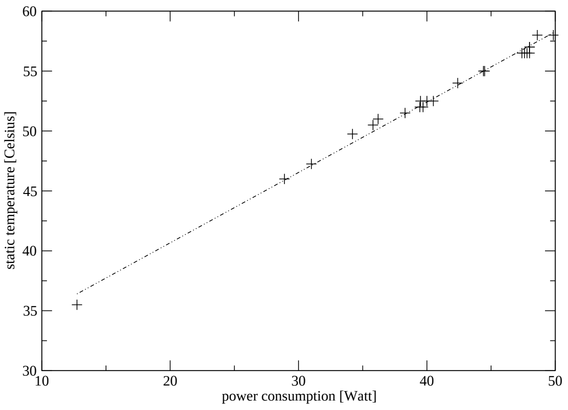

Figure 3.2: power consumption and static temperatures of the test programs

$$\begin{aligned} dT &= [c_{11}P - c_{12}(T - T_1)]\,dt & & c_{11}, c_{12} \text{ const.} \\ dT_1 &= [c_{21}(T - T_1) - c_{22}(T_1 - T_0)]\,dt & & c_{21}, c_{22}, T_0 \text{ const.} \end{aligned}$$

This model does not work as intended. It was designed to produce a better estimation for the idle temperature, but it only works if the value for c_{22} is larger than c_{12}. The larger c_{22} is, the more accurate the idle temperature estimation.

A large c_{22} means the secondary heat sink has a low heat capacity thus reacting faster than the primary heat sink. This is in contradiction to the observation leading to this model.

With an infinitely large c_{22}, the model is equivalent to a better fitting of the values in figure 3.2, which is the reason behind the next model.

3.3.3 A More Dynamical Model

The idea of this model is to use a better function to be fitted to the energy/temperature values in figure 3.2. For this purpose, the `dqed` program was modified to fit polynomial functions to values with the restriction on the function to be an upper limit for the values.

A quadratic function, $T_s(P)$, was found to fit these values better than a linear function with the same restrictions (figure 3.3). The fitting to the idle temperature is very accurate, whereas for the other values the accuracy of the interpolation is changed only slightly.

The values c_1 and T_0 are computed like before, only this time from a tangent to the quadratic

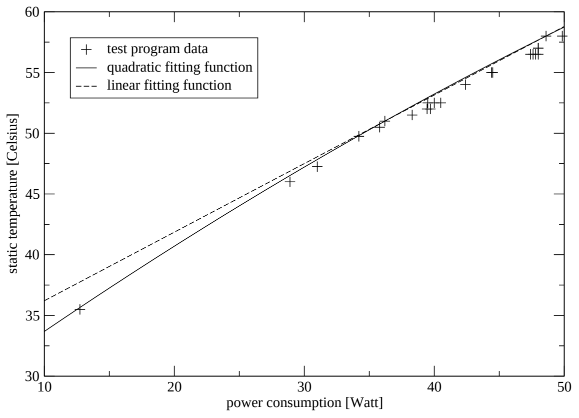

Figure 3.3: the quadratic function $T_s(P)$ fitted to the test program characteristics

function in the point $(P, T_s(P))$. The complete calculation is shown below:

$$\begin{aligned}
T_s(P) &= a_2P^2 + a_1P + a_0 \qquad a_2,\, a_1,\, a_0 \text{ const.}\\
dT &= (c_1(P)P - c_2(T - T_0(P)))dt \qquad c_2 \text{ const.}\\
P \text{ const. } : T(t) &= \frac{-\tilde{c}}{c_2}e^{-c_2t} + \frac{c_1}{c_2}P + T_0 \qquad \text{(equation 3.3)}\\
\frac{c_1(P)}{c_2} &= T_s'(P) = 2a_2P + a_1 \rightarrow c_1(P) = c_2T_s'(P)\\
T_0(P) &= T_s(P) - T_s'(P)P\\
dT &= (c_1(P)P - c_2(T - T_0(P)))dt\\
&= \left[c_2(2a_2P + a_1)P - c_2(T - (a_2P^2 + a_1P + a_0 - (2a_2P + a_1)P))\right] dt\\
&= c_2\left[2a_2P^2 + a_1P - T + a_2P^2 + a_1P + a_0 - 2a_2P^2 - a_1P\right] dt\\
&= \left[a_2P^2 + a_1P + a_0 - T\right] c_2dt\\
&= \left[(a_2P + a_1)P + a_0 - T\right] c_2dt
\end{aligned}$$

While the original idea to this model seems a bit more complex, the value of dT can be computed using only 4 multiplications and 3 additions, which is only slightly more than in the initial model with 3 multiplications and 2 additions.

In this model, the parameter for adaption to ambient temperature is a_0, the constant of the quadratic function $T_s(P)$. A change of this parameter eventually causes the same amount of change on the estimated temperature.

This model works almost perfectly, it has only one problem: it is a little too slow on a temperature increase, and a little too fast on a temperature decrease, so that the predicted temperature is always a little bit lower than the actual temperature.

3.3.4 Final Model

The final model used is only a minor extension from the previous model. On a sudden power increase, the model should react more quickly, and more slowly on a power decrease. The only parameter controlling the speed of temperature adjustment is c_2, so two new parameters are used instead: $c_{2,up}$ and $c_{2,down}$ (with $c_{2,up} > c_{2,down}$ and $c_{2,up} \approx c_2 \approx c_{2,down}$).

The constant $c_{2,up}$ is used if the last computed value of dT was ≥ 0, indicating a raise in temperature, $c_{2,down}$ in the other case. The algorithm for this model is shown in algorithm 1.

Algorithm 1 Temperature estimation

```
$T \leftarrow 35.5$ {in °C}
$up \leftarrow true$
loop
  {e.g. the main loop of mbmon}
  if $up$ then
    $c_2 \leftarrow c_{2,up}$
  else
    $c_2 \leftarrow c_{2,down}$
  end if
  compute $P$, $dt$ {done in energy estimator}
  $dT \leftarrow [(a_2 P + a_1) P + a_0 - T] c_2 dt$
  $T \leftarrow T + dT$
  if $dT \geq 0$ then
    $up \leftarrow true$
  else
    $up \leftarrow false$
  end if
end loop
```

3.4 Evaluation

The accuracy of the estimated static temperatures of the test programs using the final thermal model without energy estimation can be seen in figure 3.3. The thermal model is designed not to underestimate the temperature of the test programs, which is obviously achieved. The highest occuring overestimation is 1.2°C with one of the program at 48 Watt, resulting in a relative error of 2.12%.

In the mode using the combined power and thermal model, however, errors from the power model propagate through the thermal model, resulting in a larger temperature overestimation. This can be seen in figure 3.4 where the estimated and measured static CPU-temperature on a run of some test programs are displayed. The program `mbmon` with the combined energy and temperature

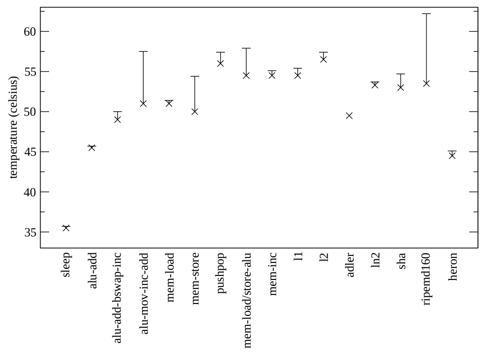

Figure 3.4: temperature estimation errors of the test programs

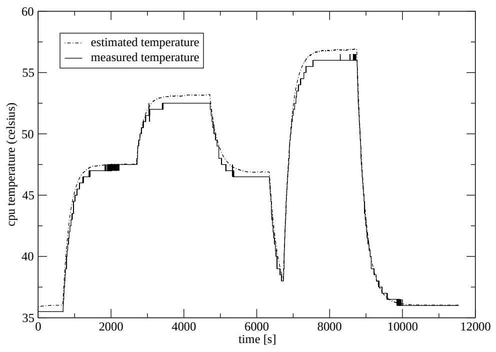

Figure 3.5: dynamic view on the model's accuracy

estimator was used to produce these measurements. These programs are the same as in figure 2.3 to allow comparisons.

Here the highest temperature difference is about 8.7°C on running the test program `ripemd-160`, which is the test program with an energy consumption overestimation of 29%. Therefore the temperature overestimation is not surprising. The overestimation on the programs alu-mov-inc-add, mem-store and mem-load/store-alu can be explained partly by their energy consumption overestimation, too.

The dynamic properties of the thermal model can be seen in figure 3.5. It shows a sequential run of the test programs alu-add, mem-load, heron and mem-inc, selected because of their particular low temperature overestimation. Each program is run at most 2000 seconds to allow the CPU to reach a static temperature. The run of the first program (alu-add) clearly demonstrates the necessity of this length. This experiment supports the decicion to separate the constant c_2 into $c_{2,up}$ and $c_{2,down}$ to keep the estimated above the actual temperature.

Of the real-world applications tested in the last chapter only perl-bench and kernel compilation are suited for an unattended 2000 second run. The accuracy of the combined energy/temperature estimation on these programs is shown below:

program	measured temperature	relative error of temperature estimation
kernel compilation	52°C	3.8%
perl-bench	54°C	8.5%

Chapter 4

Temperature Control

With a working energy and temperature estimation it is not difficult to implement temperature control of the machine. This could be done, as in [13], by additional code in the scheduler.

This work, however, uses resource containers which provide more flexibility in accounting and limitation of processes and the machine. The temperature dependent adaption of a machine-level energy limit is sufficient to enforce a given temperature limit on the processor.

4.1 Resource Containers

Resource containers were introduced by Banga [3] to provide a data structure for accounting and controlling various resources like CPU-time, energy consumption, network bandwidth, etc. In this work an implementation of resource containers in the Linux kernel 2.5.58 by Martin Waitz [19] was used. It consists of a hierarchy of containers and is currently able to account and limit the resources CPU-time and energy using the time stamp counter and the energy estimator of this work.

Each process is associated with a resource container. Consumed resources are accounted to the current process's resource container and all of its parent resource containers. Thus, the root resource container hold the accounted resource consumption of the entire machine.

The implementation ensures that limits placed on a container and its parents are never exceeded. A process is allowed to run if all resources of the associated container and its parents are available. If no process is runnable, the CPU is put into the halt state with a low power consumption, resulting in a reduction of CPU temperature.

In the case of energy, the implementation features simultaneous limits on two different time-slices (128 ms and 1024 ms per default) on every container. This is useful, because short CPU bursts do not have an immediate effect in temperature. Therefore, the sum of the limits in the short time-slice can safely exceed the longer time-slice's limit, as long as every limit holds.

Access to resource containers, like reading the contents or setting a new limit, is available via an additional system call. Another system call can be used to modify the container hierarchy. These two system calls provide a flexible way of manipulating every aspect of resource containers from userspace.

4.2 Computing Energy Limits and Implementation

The approach to temperature control taken in this work is to use the thermal model 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:

$$dT = [(a_2 P + a_1)P + a_0 - T]c_2 dt \leq T_{limit} - T$$

This forms the quadratic inequation

$$a_2 P^2 + a_1 P + a_0 - T \leq \frac{T_{limit} - T}{c_2 dt}$$

Because $a_2 < 0$, the solution to this equation is:

$$\begin{aligned}
P^2 + \frac{a_1}{a_2}P + \frac{a_0}{a_2} - \frac{T}{a_2} &\geq \frac{T_{limit} - T}{a_2 c_2 dt} \\
P^2 + \frac{a_1}{a_2}P + \left(\frac{a_1}{2a_2}\right)^2 &\geq \frac{1}{a_2}\left(\frac{T_{limit} - T}{c_2 dt} + T - a_0 + \frac{a_1^2}{4a_2}\right) \\
\left(P + \frac{a_1}{2a_2}\right)^2 &\geq \frac{1}{a_2}\left(\frac{T_{limit} - T}{c_2 dt} + T - a_0 + \frac{a_1^2}{4a_2}\right)
\end{aligned}$$

$$P \geq \frac{-a_1}{2a_2} + \sqrt{\frac{1}{a_2}\left(\frac{T_{limit} - T}{c_2 dt} + T - a_0 + \frac{a_1^2}{4a_2}\right)} \quad \vee \tag{4.1}$$

$$P \leq \frac{-a_1}{2a_2} - \sqrt{\frac{1}{a_2}\left(\frac{T_{limit} - T}{c_2 dt} + T - a_0 + \frac{a_1^2}{4a_2}\right)} \tag{4.2}$$

The solution 4.1 can be ignored because it would enforce a minimum power consumption limit in order to reduce CPU temperature, in contradiction to fundamental laws of physics.

The code for limiting the root container was added to `mbmon`. To avoid problems with the process `mbmon` getting throttled too much to be able to set a new limit, the `mbmon`-process is accounted, but never throttled.

4.3 A Note on Flexibility

The claimed task-level specific flexibility of this temperature control method is derived from the flexibility provided by the resource container concept and implementation, and as such does not

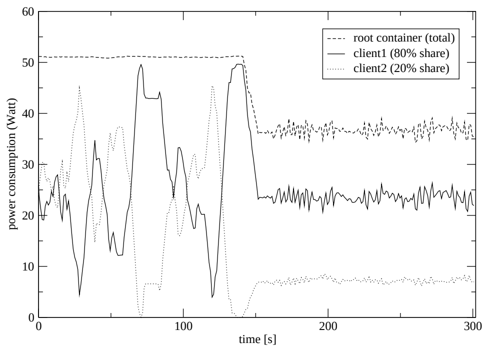

Figure 4.1: Example of Flexible Limitation

lie within the scope of this work. However, an experiment is included here to demonstrate this flexibility.

It consists of an apache web server running under a Linux kernel with resource container support. Different clients send CGI-requests which result in the execution of a test-program used in this work. Depending on the client IP, a forked apache process is attached to one of two special resource containers on the machine. The two containers are limited to 30% and 70% of the total power consumption limit. Note that no modification to the apache web server was necessary (see details in [19]).

The estimated power consumption of the processes attached to the two resource containers and the root container is shown in figure 4.1. Initially, the total power consumption limit is infinite, so the processes attached to each of the two containers are able to consume the maximum power available. At the beginning of the limitation, the relative limits of the two containers are enforced. It can be seen that the relation of the two containers' power consumption is near 30 : 70.

The power consumption of the root container is higher than the sum of the two other containers, because energy consumption of the halted CPU is accounted only to the container of the idle process and its parent, the root container.

4.4 Evaluation

4.4.1 Accuracy

The accuracy of the implemented temperature control method can be seen in figure 4.2. It shows the estimated temperature during a run of one of the test programs with the temperature limit set to 40°C, between the first and the last temperature values exceeding the limit.

The highest transgression of the limit is near the beginning and lies only 0.036°C above the limit. Statistical analysis yields an average temperature of $(40 - 0.00102)$°C and a standard devi-

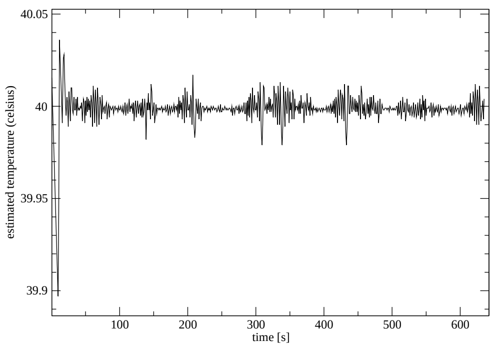

Figure 4.2: Closeup view on the estimated temperature being limited to 40°C

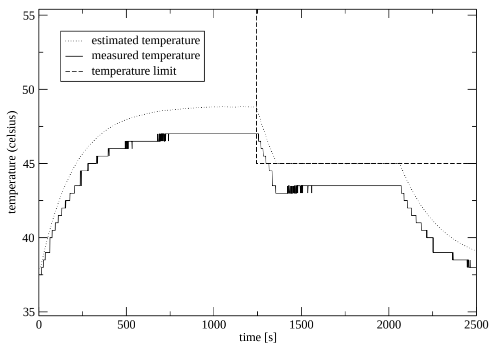

Figure 4.3: Different temperature limits

ation of 0.0073°C. The slight exceeding of the limit can be explained by the reactive nature of the resource container limitation code which is only invoked on a limit transgression.

As the limitation of the estimated temperature is quite precise, the accuracy of the real temperature limitation essentially depends upon the accuracy of the thermal model. This can be seen in figure 4.3. After reaching a static situation, the difference between the estimated and the real temperature remains constant under different limits within the scope of the health monitoring chip's limited exactness.

4.4.2 Overhead

Overhead evaluation was done measuring the execution time of a long running process in different environments. The test was run using the modified Linux 2.5.68 kernel with resource container support provided by Martin Waitz [19].

resource container support	temperature measurement	temperature control	average time [s]	overhead (normalized)
✗	✗	✗	1528.0880	0.00%
✗	✓	✗	1603.5450	4.94%
✓	✗	✗	1537.1560	0.59%
✓	✗	✓	1537.2185	0.60%
✓	✓	✗	1613.2570	5.57%
✓	✓	✓	1613.2915	5.58%

The environment to which these overhead values are normalized consists of the modified Linux 2.5.68 kernel with the resource container support switched off, which is equivalent to a standard Linux 2.5.68 kernel, without any temperature measurement or control. The overhead values seem to be additive: the resource container support adds $\sim 0.6\%$ of overhead, periodical measurement $\sim 5\%$. This can be explained by the fact that each temperature measurement takes approximately 8.5ms in contrast to 40ns for the reading of a performance counter.

The overhead of the temperature control method alone cannot be measured exactly with a program running only ~ 1500s, because it is lower than the exactness of measurement. So it is estimated to be $\leq 0.04\%$ which is two times the maximum observed fluctuation relative to the average run length. Therefore the total overhead of this temperature control method on the system is $\sim 0.6\%$ in contrast to the $\sim 5\%$ overhead of periodical measurement.

Chapter 5

Future Work

5.1 Energy Estimation

Analysis Methods Although the analysis with `dqed` provided satisfying results, the program itself is rather old (written in 1985). Analysis using better algorithms could provide more accurate energy models, but a leap in accuracy is not expected.

Energy Models The power model presented in this work uses only direct event-energy correlation, which results in this linear approach. However, non-linear combinations of events, such as the ratio of computed μops and clock cycles, could be useful, too. Non-linear energy models could provide more accurate estimations or could be applicable to a wider range of applications.

Energy Counters The worst problem of this work are the restrictions incorporated into the performance counter architecture, severely limiting the set of simultaneously countable event sets. Less restrictions in the performance counter architecture or the introduction of energy related events would greatly enhance the exactness of task specific energy consumption estimation. A small delay in the propagation of event occurence to the counter register is acceptable, because the accounting is done continuously and the heat sink also has a delaying effect on the temperature.

5.2 Temperature Estimation

Thermal Models The thermal models presented in this work, especially the final model, is rather ad hoc than physically correct. A more elaborate and physically correct model could provide better temperature estimations, although the computational costs are expected to be significantly higher. For example, the addition of thermal radiation to the model could provide a better temperature estimation, but the differential equation (like equation 3.1) would be much more difficult to solve.

Ambient Temperature Throughout this work, the ambient temperature was considered constant. While this assumption is acceptable in case of air-conditioned computer rooms, it does not hold in case of a normal room or failure of the air-conditioning unit. This has to be incorporated into the thermal models, possibly by measuring the ambient temperature and adjusting the corresponding parameter.

5.3 Temperature Control

Implementation Issues The disadvantage of temperature control from user-space, as presented in this work, is that the controlling program has to be scheduled, and therefore is not run in a deterministic fashion. Thus a kernel implementation would be desireable, and since temperature estimation only needs an update every second, the time-consuming floating-point arithmetic or integer emulation of the same should be acceptable.

Cluster Support The most useful application of this temperature control method is expected to be in computing centers. As shown in section 1.1, quite an amount of research has been spent on this topic, concentrating on request management. Energy estimation of incoming requests can be rather difficult in general, unless the events have a very simple form.

Applying this work to computer clusters means that individual temperature limits could be set for each computer based on its location inside the cluster and the location of the air-conditioning units. In this scenario, energy estimation of incoming requests prior to execution is not necessary, because the estimation happens at execution time. This also avoids a potential bottle-neck, since the estimation is done parallel on the cluster. Cluster temperature control could relax the requirements on the air-conditioning unit, therefore making it possible to save energy.

Chapter 6

Conclusions

This work presents a different approach to temperature control without the need for continous measurement. This is achieved by using performance counters together with a power and thermal model of the processor to predict its temperature. The reverted thermal model, combined with resource containers, can then be used to enforce a temperature limit.

Different analysis methods for the power model and various thermal models are detailed and discussed in the course of this work. Concerning the power model, it is shown that a simple linear combination of performance events can be a good energy consumption indicator for a group of programs. From a physically correct thermal model taking only conversion into account, a more accurate model is constructed, using only a slight variation of the constants.

The reversal of the thermal model can be used to compute power consumption limits for the CPU based on the estimated temperature and a temperature limit. In combination with the concept of resource containers or more specifically, energy containers, which can enforce such a limit, this results in a flexible temperature control method providing task level throttling while maintaining the global temperature limit.

The flexibility of this method exists due to the flexibility of the resource container concept only. However, this work provides a power model for a subclass of resource containers named energy containers, which are dependent on this kind of task specific energy consumption information.

It is also shown that the power model and therefore all dependent work, the thermal models and the temperature control method, are only applicable to a specific group of programs. This is mainly due to limitations in the performance counter architecture, and, to a lesser extent, the simple linear approach to the power model.

As a proof of concept, the models are applied to a standard Pentium 4 computer in a midi-tower and tried with a variety of programs, including synthetic micro- and normal benchmarks as well as real-world applications. These experiments show that the approach of this work is applicable to a group of programs including real-world applications. It could readily be used in environments such as computing centers to save energy by controlling the temperature of the individual computers.

Bibliography

[1] Various authors. netlib - a collection of mathematical software, papers and databases. `http://www.netlib.org`.

[2] David Avis. lrslib. `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`.

[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'1999*, Feb 1999.

[4] 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)*, Jan 2001.

[5] 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'2001*, October 2001.

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

[7] Torbjorn Granlund and Kevin Ryde. Gnump - library for arithmetic on arbitrary precision numbers. `http://www.swox.com/gmp/`.

[8] Hanson and Krogh. Solving nonlinear least squares and nonlinear equations. `http://www.netlib.org/opt/dqed.f`.

[9] Intel. *IA-32 Intel Architecture Software Developer's Manual*, 2002.

[10] Alexander Larsson and Frodo Looijaard. Hardware monitoring tools. `http://secure.netroedge.com/~lm78/`.

[11] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. *Compilers and Operating Systems for Low Power*, chapter Dynamic Cluster Reconfiguration for Power and Performance. Kluwer Academic Publishers, 2002.

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

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

[14] 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*, Feb 1997.

[15] 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, Feb 2003.

[16] Yoshifumi R. Shimizu. Mother board monitor program for the x window system. `http://www.nt.phys.kyushu-u.ac.jp/shimizu/index.html`.

[17] 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)*, Jan 2002.

[18] Brinkley Sprunt. Brink and abyss. `http://www.eg.bucknell.edu/~bsprunt/emon/brink_abyss/brink_abyss.shtm`.

[19] Martin Waitz. Accounting and control of power consumption in energy-aware operating systems. 2003.

Ereignisgesteuerte Temperaturregelung in Betriebssystemen

Betrachtet man die Prozessorentwicklung in den letzten Jahren, so wird deutlich, dass Kriterien wie Leistungsaufnahme und damit verbunden die Prozessorkühlung immer wichtiger werden. Besonders gut sieht man diese Entwicklung am Aufwand für die Kühlung: Hier reicht das Spektrum von einfachen Chipgehäusen ohne zusätzliche Kühlung bis hin zu aufwändig verarbeiteten Prozessorgehäusen und massiven, belüfteten Kühlkörpern.

Da es auch und besonders in Server-Farmen mit der dort vorhandenen hohen Prozessordichte schwierig wird, mit normalen Klimaanlagen auszukommen, wird an diesem Problem schon einige Zeit gearbeitet. Bisherige Lösungen, wie z.B. das „Thermal Monitoring“ im Pentium 4 oder auf Messung der CPU-Aktivität basierende Softwaremethoden, haben jedoch teilweise gravierende Nachteile: Entweder wird der Prozessor gedrosselt, was sich sowohl auf alle momentan laufenden Prozesse als auch auf die Interruptlatenz auswirkt, oder aber die Methode kann auf aktuellen Prozessoren nicht angewendet werden.

In dieser Arbeit wird deshalb ein anderer Weg benutzt: Da die CPU-Aktivität nicht mehr im direkten Zusammenhang mit der tatsächlich aufgenommenen Leistung steht, müssen mehr Informationen über interne Prozessorabläufe hinzugezogen werden. Hier bietet sich die „Performance Counter“-Architektur in modernen Prozessoren an, die es ermöglicht, prozessorinterne Ereignisse zu zählen.

Aus diesen Häufigkeiten wird durch Linearkombination eine Energieabschätzung gewonnen. Die Linearkombination begründet sich durch die linearen Eigenschaften von Prozessorereignissen, die im direkten Zusammenhang zu einer aufgenommenen Energiemenge stehen. Die Auswahl der konkreten Ereignisse erfolgt manuell, da es zu viele Kombinationsmöglichkeiten gibt. Für die Berechnung möglichst präziser Koeffizienten der Linearkombination werden zwei unterschiedliche Programme vorgestellt. Hier wird darauf geachtet, den Energieverbrauch nicht zu unterschätzen, da sonst Limitierungsmassnahmen nicht den gewünschten Effekt hätten.

Die Auswertung des Energiemodells zeigt, dass es für eine relativ grosse Gruppe von Programmen, die auch reale Anwendungen beinhaltet, zufriedenstellende Ergebnisse liefert. Anhand eines Programms wird auch demonstriert, dass das Modell nicht geeignet ist, den Energieverbrauch aller möglichen Programme abzuschätzen. Hinderlich sind hier in erster Linie die Einschränkungen der Performance Counter des Pentium 4.

Um aus dem Energieverbrauch die aktuelle Prozessortemperatur abzuschätzen, wird der Kühlkörper des Prozessors modelliert. Berücksichtigt werden dabei die Aufnahme elektrischer Energie und die Abgabe von Wärmeenergie durch Konvektion. Bei diesem Ansatz müssen nur noch die Konstanten bestimmt werden. Dabei stellt sich heraus, dass durch eine leichte Variation der Konstanten eine genauere Abschätzung möglich wird. Die Bestimmung der Konstanten erfolgt ebenfalls unter der Bedingung, die Temperatur nicht zu unterschätzen.

Die Auswertung des kombinierten Energie/Temperaturmodells ergibt ähnliche Fehler, was darauf schliessen lässt, dass die Hauptfehlerquelle im Energiemodell zu suchen ist.

Der Vorteil eines solchen Energie/Temperaturmodells liegt darin, dass das Temperaturmodell leicht umkehrbar ist, um damit aus der aktuellen Temperatur und einer Maximaltemperatur ein Energielimit für die nächste Zeitscheibe zu berechnen. Dieses Temperaturregelungsverfahren wird mit Hilfe von „Resource Containern“ implementiert, einer hierarchischen Datenstruktur im Linux Kern, die es ermöglicht, verschiedene Resourcen feingranular abzurechnen und den Resourcenverbrauch zu limitieren. Um den Energieverbrauch zu berechnen, benutzt die Resource-Container Implementierung von Martin Waitz [19] bereits das Energiemodell dieser Arbeit. Ein Benutzerprogramm berechnet damit die aktuelle Prozessortemperatur und setzt neue Energielimits für den Rechner.

Die Auswertung dieses Verfahrens zur Temperaturregelung zeigt, dass die geschätzte Temperatur sehr präzise unter dem Temperaturlimit gehalten wird. Die zusätzlichen Kosten für die Energie- und Temperaturabschätzung sind marginal, insbesondere im Vergleich zur periodischen Messung der Prozessortemperatur.

In Zukunft könnte vor allem das Energiemodell noch verbessert werden, entweder durch einen neuen, nicht linearen Ansatz oder mit der Einführung von energierelevanten Performance Counter Ereignissen in zukünftigen Prozessoren.