Energy Estimation for CPU-Events Dependent on Frequency Scaling and Clock Gating

Philip P. Moltmann

System Architecture Group (IBDS Bellosa)
Department of Computer Science
University of Karlsruhe (TH)

STUDY THESIS

Advisors:	Dipl. Inform. Andreas Merkel
	Prof. Dr. Frank Bellosa
Begin:	May 2, 2006
Submission:	September 1, 2006

Abstract

The power consumption and heat dissipation of multipurpose CPUs have become a real problem. E.g. a Pentium 4 3.7 Ghz EE consumes $154W/cm^2$. The heat dissipation has to be handled by the cooling system: heat sink, fan, and room air condition. This means that for a big server farm or cluster the cooling system can get very big and expensive.

The power consumption of a CPU depends heavily on the current load and even on the specific instruction executed. This means, that the maximum heat dissipation will very seldomly be reached. The idea of an energy aware system is to assure, that an specified maximum power is never reached. This can be used to throttle the system in a way, that only few performance is lost, but that the requirements of the cooling system will be much lower.

The system estimates the current energy consumption of the CPU from built in event counters. Each counter is multiplied by a precomputed energy weight to get the total power usage. This work introduces a new approach to get more accurate energy weights and shows how these weights vary, while the CPU is running at different frequencies or while the CPU's clock is modulated.

Up to now an energy aware system keeps energy limits by switching to the idle thread. This work presents an algorithm that enables the system to use frequency scaling and clock gating to keep the energy limits. The system was implemented as a modification of the Linux 2.6.17.3 kernel. The evaluation of this systems shows a performance benefit of up to 100% and more compared to a system that only uses idling.

Acknowledgments

I would like to thank Simon Kellner and Prof. Wolfgang Weil for their support with this study thesis.

Contents

1 Introduction

During the last 10 years the heat dissipation of a multipurpose-CPU rose steadily by 650%. The i80386, introduced in 1985, needed $1W/cm^2$. In 1995, the Pentium Pro needed $23W/cm^2$ and todays maximum heat dissipation is reached by the Pentium 4 3.7 Ghz EE with $154W/cm^2$. Almost all of this power is transformed into heat. Therefore the heat dissipation also grew exponentially.

In some scenarios, there is no problem with handling such large heat dissipation, but in mobile computers or large server farms, heat can be a serious problem. The power usage and consequently the heat dissipation can be split in two categories.

Most of the static power usage is leakage power and accounts for at least 40% of the total power consumption in modern microprocessors. Static power is consumed, when the transistors of the CPU are supplied. This means, that it can be reduced on hardware level e.g. by powering off parts of the CPU or reducing the frequency or voltage. From software level it can only be reduced, if the hardware offers special power saving features.

If the CPU is running a program, it needs additional power. This power is called dynamic power. It can comprise up to 60% of the current power.

There are three important values describing the power usage characteristics of a CPU. The idle power describes how much power is used when running the idle thread. Most operating system are executing the HLT-instruction in the idle thread. On the Pentium 4 and most of the other modern microprocessors, this instruction sets the CPU in a sleep mode, resulting in powering off major parts of the CPU and sometimes reducing frequency and voltage.

On the other side, the maximum power usage is described by the TDP – thermal design power. This is a theoretical value, describing the peak power usage in the worst case, which seldomly occurs in the real world. In between this values is the current power usage of the executed thread. This value is highly specific to the executed command and therefore thread. For example, the idle power of a Pentium D 830 is about 43 W, the TDP is 130 W ([Int06]) and the current power usage varies between 103 W and 89 W, if the CPU is running at 3 Ghz and 74 W and 67 W if the CPU is running at 2.8 Ghz.

In a server farm the air conditioning system has to be designed for the worst case. Since it is unlikely to happen that all CPUs dissipate the theoretical maximum of heat, the cooling solutions is highly overpowered.

A so called energy aware system assures an (adjustable) maximum power usage to the user. Additionally it is able to estimate the power usage of a CPU, so that the overall power usage of a server farm is calculable. With this system it is possible to design the cooling solution to the average case and throttle the system, if it exceeds this limit. This results in a small performance loss, but will enable the designer of the system to build a much smaller and therefore cheeper cooling system [BKW03].

Reducing the power consumption not only enables the designer of a server farm to use a smaller air conditioning system, it also enables him to use more appropriate power supply units (PSUs). As the PSU would have been designed for the worst case in a traditional system and for the average case in an energy aware system, they can be much smaller. As most PSUs work most efficiently at maximum load, this results in an additional power usage reduction.

Up to now, energy aware systems (as described above) execute the idle thread

if the energy limit is reached [BKW03]. This approach is justified, if the CPU doesn't support any hardware-level throttling features.

Some of the more modern Pentium 4/D CPUs do support the so called Thermal Monitor 2 (also called Enhanced Speedstep Technology). This function enables system programmers to manually adjust the frequency and voltage of the CPU. Additionally the clock of all Pentium 4 CPUs can be gated, resulting in a fake frequency, but in no voltage adjustment.

This study thesis examines, if and in which way these two techniques – frequency/voltage scaling and clock gating – can be used to build an energy aware system. The reason for using such features for an energy aware system is, that they are designed to reduce power consumption and therefore can be helpful for keeping energy budgets. Unfortunately they are not specifically designed to be easily usable in a system that uses performance counters for energy estimation. The reason of this counters was to help programmers to optimize the performance of their algorithms. This work researches the influence of such features on the energy estimation and on the throttling process, specifically on the energy weights used to calculate the energy form the performance counters as described in Section 2.1.1.

In Section 2, the power saving technologies used are described. Also described is how to estimate used energy from recorded performance counters. If you're interested in some related work, please refer to Section 3. Section 4 gives a generic overview about how to design an energy aware system that uses frequency scaling. In Section 5, the energy weights, needed for estimating the energy, are gained. I implemented a sample system and evaluated it, to see how good it performs (Section 6). Finally, Section 7 brings up some ideas about future work, which might be done, and concludes.

2 Prerequisites

2.1 Performance counters

An energy aware system uses the performance counters of the Pentium 4/D to estimate the current power usage. These counters were initially introduced into the CPUs as help to optimize the performance of a program [Int03] [Int04] [Int05]. The performance counters are model specific registers (MSRs) and not part of the IA-32 architecture. This means that some IA-32-compatible CPUs may have these counters, some may not.

The Pentium 4 and its descendant, the Pentium D, feature 18 performance counters, which can be configured to count a variety of events, for example to count each clock tick or each mispredicted branch. The idea is that some events correspond to the energy consumption of the CPU [BKW03]. E.g. the time stamp counter (a special performance counter) can be used to simulate the static power, because it is incremented at each clock tick, even if the CPU is executing HLT.

2.1.1 From performance counters to power consumption

As there are only 18 counters and a Pentium 4/D is a quite complex CPU, it is not possible to configure the counters in a way that each energy consuming event is accounted by exactly one counter. Kellner states that the counters shown in Table 1 give the best estimation of the used energy on a Pentium 4 2.0 Ghz [Kel03].

Each counter is multiplied by the corresponding energy weight to calculate the total power usage.

The power consumption of a specified period of time is:

$$power = \frac{\sum events\ counted\ in\ this\ period \cdot \overbrace{energy\ per\ event}^{energy\ weights}}{period\ length} \tag{1}$$

To predict future power consumption, it is assumed that the period length is much shorter than the average time between load changes. Then the power consumption of the next period is similar to the of the last period.

2.2 Clock gating

Clock gating was introduced to the Pentium 4 as an emergency-function, if the CPU is about to overheat. Clock gating means that the CPU is stopped for some cycles and then is working for some cycles. The Pentium 4/D can be throttled

Event	Energy per event
Time stamp counter	6.17
Unhalted cycles	7.12
μop queue writes	4.75
Retired branches	0.56
Mispredicted branches	340.46
Mem retired	1.73
Mob load replay	29.96
Ld miss 1L retired	13.55

Table 1: Energy per event in nJ on a Pentium 4 2.0 Ghz [Kel03]

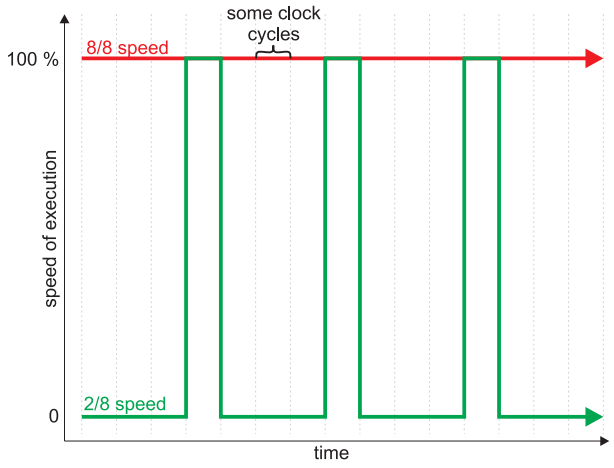

Figure 1: CPU state while clock gating

to 8/8, 7/8, 6/8, 5/8, 4/8, 3/8, 2/8, 1/8 speed. Throttling to 2/8 speed for example means, that for $3 \cdot x$ cycles the CPU is stopped and during the next x cycles the CPU is working (See Figure 1).

As a cycle when clock gating has the same length as when running at full speed, it is not possible to reduce the voltage, because the speed of the state change of a transistor depends on the supplied voltage.

Dynamic power is only consumed, if the CPU is working. As clock gating stops the CPU for some cycles, the CPU will consume less energy per period. The total power usage of a program will be the same or even more, because even if the CPU is stopped, it consumes static power.

2.3 Frequency scaling

Some of the Pentium 4/Ds support frequency scaling. Frequency scaling means that the cycle length of the CPU is adjusted. In Figure 2, the cycle length is set to be 4 times the cycle length of the maximum speed of the CPU. This results in a performance of 1/4 of full speed.

In contrast to clock gating, frequency scaling enables the CPU to reduce the voltage of the CPU. The dynamic power usage of a CPU can be roughly described as:

$$power = active\ transistors \cdot frequency \cdot voltage^2$$

This means that voltage scaling has a much deeper impact on power consumption than frequency scaling. But since voltage is depended on the needed transition-speed of the transistors, voltage scaling is dependend on frequency scaling.

The Pentium 4/Ds support only two frequencies: full speed and 2.8 Ghz.

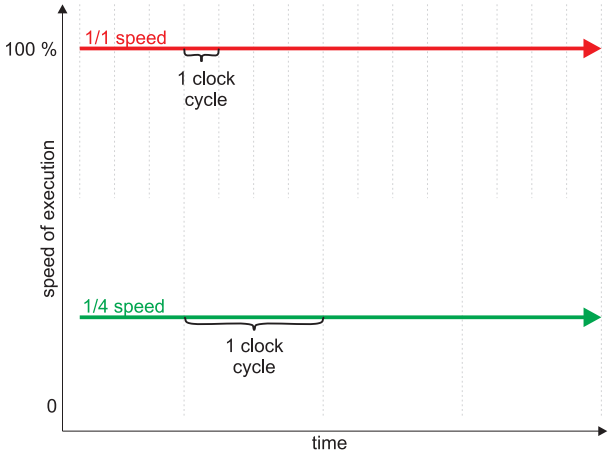

Figure 2: CPU state while frequency scaling

3 Related work

A review about power saving and thermal management technologies is [BM01]. It also gives some hints, when to use which technology, but it is written from a hardware designer's point of view.

Kellner tries to find the best performance counters for energy estimation purposes. He also develops a model to predict CPU temperature from the previously estimated power [Kel03]. Section 5 of this work is an extension to his work.

In [BKW03] the results of [Kel03] are used to implement an energy aware system, as it is described in Section 4.

Furthermore Merkel showed in [Mer05] how to use performance counters and energy weights to balance runqueues in a multiprocessor system, in order to achieve a homogeneous energy consumption.

Another interesting usage of performance counters is described in [Bel01]. The author is using one of the performance counter of the Intel Xscale to predict the influence of frequency scaling on performance. His results show, that the performance is not only dependent on the frequency, it is also dependent on the frequency of memory / cache references. If a thread contains many memory references, its performance is dominated by the memory speed, not by the CPU speed.

4 Design

This thesis describes a system that uses frequency/voltage scaling to stay under a given power consumption level. The goal of this system is to achieve maximum performance while using the least possible power.

All accounting can be done either per thread or for the current set of threads. Accounting for each thread is more accurate, since it regards the different characteristics of the threads. Since the focus of this work is to prove that power saving features can be used, I ran only one benchmark at a time. This means that a global acounting is accurate too.

4.1 Keeping energy budgets by idling

As described in Section 1, the idle thread uses far less energy than a real working thread, because it is executing HLT. This can be used for adjusting the power consumption.

An energy budget is a certain amount of energy used in a specified period. To throttle the system by idling, I divided the time into periods of equal length. At the beginning of each period the performance counters are set to 0. At each scheduling interval, the performance counters are read and

$$usedEnergy = \sum events\ counted\ in\ this\ period \cdot energy\ per\ event \tag{2}$$

and

$$usedEnergy + idlePower \cdot timeLeft > maxPower \cdot periodLength \tag{3}$$

are calculated. If Equation 3 is true, the idle thread is scheduled. Since the system switches to the idle thread not until too much energy is consumed, more power is used than specified. To solve this problem, the energy budget of the next period will be lowered to compensate the energy overuse.

This approach simulates a lower heat dissipation by switching to idle and back. Because the system is running at full speed for a short period of time, the system dissipates more than the maxPower. This is no problem, because heat sink attached to the CPU usually works as a heat capacitor to smooth the heat dissipation to a homogeneous level.

Another benefit of keeping energy budgets is that the power consumption can be controlled. But due to the inhomogeneous power consumption it also has to be smoothed to assure that it stays below the limit. This can be done by adding some big capacitors to the power supply unit of each computer.

4.2 Keeping energy budgets by frequency scaling

Additionally to switching to the idle thread (Section 4.1), frequency scaling can be used to keep the energy budget. If the clock cycles are longer, the voltage can be reduced. The voltage is set to the lowest possible value for each frequency. As there is only one MSR responsible for frequency and voltage settings, both setting are applied concurrently.

At the beginning of each period it is computed, at which frequency the next period would perform best. As it is assumed that the load of the next period would be similar to the load of the last period, the best frequency can be computed as follows:

First the time the working threads would have run at each available frequency, is to be computed. For example, if the last period was run at 3 Ghz, the working time at 2.8 Ghz would have been.

$$workTime_{2.8} = \frac{workTime_3 \cdot speedRatio_3}{speedRatio_{2.8}} \tag{4}$$

$speedRatio_{freq}$ describes how much work is done in one second compared to other frequencies. This can also be computed from the performance counters, but fixed values (in this case 30 and 28) are giving good results too.

$workTime_{freq}$ is the time the CPU would have been working (not idling) in the last period, if the frequency would have been $freq$. After computing the $workTime_{freq}$ for all frequencies, the performance at each frequency can be computed.

The power, which would have been used in the last period at some frequency, is computed as:

$$usedPower_{freq} = \frac{\sum events\ counted\ while\ working \cdot energy\ per\ event_{freq}}{workTime_{freq}} \tag{5}$$

For each frequency it has then to be computed, how long the working time in the next period would be ($nextWorkTime_{freq}$). A description how to get this equation can be found in Appendix B. $maxPower$ is the specified maximum power usage of the system and $idlePower$ is a precomputed value, which contains the power usage of idle thread.

$$nextWorkTime_{freq} = \frac{periodLength \cdot (maxPower - idlePower)}{usedPower_{freq} - idlePower} \tag{6}$$

From this you can calculate, how much work is done in the next period: hence the performance.

$$performance_{freq} = nextWorkTime_{freq} \cdot speedRatio_{freq} \tag{7}$$

The frequency with the highest $performace_{freq}$ is chosen for the next period. The values of *energy per event* are dependent on the frequency as described in Section 5 and for computing Equation 2 and 5 the events while working and idling have to be recorded separately.

4.3 Keeping energy budgets by clock gating

Clock gating can be used for keeping the energy budgets similar to the way it is done with frequency scaling, because from the OS's point of view, both techniques have the same effect, even though these technologies are very different.

Using techniques like frequency scaling or clock gating is useful, if there is a performance benefit, while using the same amount of energy or if they enable to use less energy, while having the same performance. But as the measurement

data below show, there is no performance or energy benefit. In contrast: there is a slight performance loss. The reason is, that clock gating doesn't allow the CPU to run with a reduced voltage and the power saving of the idle thread is as efficient as the power saving with turning off the CPU for some clock ticks as described in Section 2.2.

If you compare keeping the energy budget with switching to the idle thread and with clock gating, clock gating will perform worse. For example $bench1_{60}$ needed 1966 J and 27.8 seconds at full speed and 2101 J and 31.65 seconds at 7/8 clock gating speed on a Pentium 4 3.0 Ghz. If you want to achieve the same performance by switching to the idle thread, you have to run the idle thread for 3.75 seconds. Because the idle thread uses $\sim$ 29.4 W on this CPU, the same performance will be achieved while using 2076.25 J, this is 25 J less. Clock gating might be helpful, e.g. to keep the power consumption more homogeneous. It influences on the energy weights is also researched in Section 5.2 and 5.3.

4.4 Design summary

Switching to the idle thread is used for throttling the CPU at short load changes, as they appear as phases of an algorithm, the appearance of new threads in a dispatcher/worker scenario, or sudden heavy memory usage. It is also used, if other power saving features can't reduce the power consumption as desired.

Additionally frequency/voltage scaling is used to reduce power consumption. As a frequency/voltage adjustment takes some time, it can only be used for long term load changes, as they appear, when a new task is started or during slow I/O, like writing big files to the hard disk.

There are some combinations of current thread and maximum power, when running at a higher frequency and using idling to reduce the power performs better than using a lower frequency and no (or less) idling. In those cases the decision depends on the goal of the energy aware system: higher performance, or higher $\frac{performance}{power}$. The goal of the system described in this thesis is to maximize performance.

Clock gating can help to make the power consumption more homogeneous. It is useful in (soft) real time systems, but can't replace switching to the idle thread, because there are only a few speeds available, which is in most cases too coarse grained. In a system using batch scheduing it would only be useful, if there would be a performance or power benefit. The system described in this work doesn't use clock gating.

5 Influence of frequency scaling / clock gating on energy weights

In Equation 1, 2, and 5 the performance counters are multiplied with variables called *energy per event* also called energy weights. To obtain these weights I ran some benchmarks counting the events suggested by Kellner [Kel03]. Then I used the simplex-algorithm as described at the end of Section 5.2 to acquire the best values for the energy weights.

5.1 Measuring environment

Because every CPU model has its own energy characteristics, the energy weights have to be computed for every model. The values in this paper will be for the Pentium D 830 running only one core. The benchmarks (see Appendix A) were C / C++ programs running on a Linux 2.6 system. The power measurement was done by a Labview system. The benchmarks were run at 3 Ghz, 2.8 Ghz and all possible clock gating speeds.

5.2 Optimizing

The benchmarks and measuring returned the performance counters $\overrightarrow{c}$ and the used energy e.

$$\overrightarrow{c}^{\top} \cdot \overrightarrow{w} = e \tag{8}$$

Equation 8 is the same as Equation 2 and describes how to compute e from $\overrightarrow{c}$ and $\overrightarrow{w}$ which is a vector containing all energy weights.

As there are much more benchmarks than energy weights is it likely that there are is no perfect $\overrightarrow{w}$ that fits to all benchmarks. Therefore, you have to find the optimal solution.

To do this, I introduce a variable called y_{bench_x}, which contains the error. For a given frequency or a clock gating speed the complete Equation to be optimized is:

$$\begin{pmatrix} \overrightarrow{c_{bench_1}}^{\top} & 1 & 0 & \dots & 0 \\ \overrightarrow{c_{bench_2}}^{\top} & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \overrightarrow{c_{bench_n}}^{\top} & 0 & \dots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \overrightarrow{w} \\ y_{bench_1} \\ y_{bench_2} \\ \vdots \\ y_{bench_n} \end{pmatrix} = \begin{pmatrix} e_{bench_1} \\ e_{bench_2} \\ \vdots \\ e_{bench_n} \end{pmatrix} \tag{9}$$

We have to optimize the matrix in order to achieve, that

$$\sum_{x=1}^{n} y_{bench_x} \tag{10}$$

will be minimal, because this means that the sum of the energy misprediction for all benchmarks will be minimal.

According to Kellner, we have to pay attention that all entries in $\overrightarrow{w}$ and y_{bench_x} are positive [Kel03]. This was done, because his assumption was that

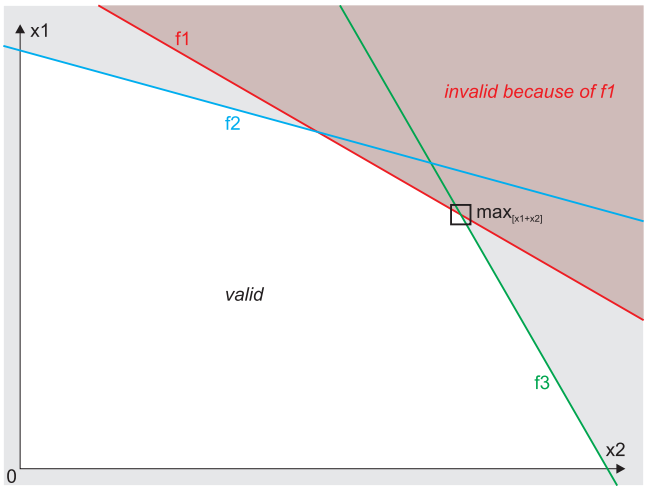

Figure 3: Visualisation of the simplex algorithm for two dimensions, three hyperplanes (f1, f2, f3), goal $\max_{x1+x2}$ and constraints $x1 > 0$ and $x2 > 0$. White is the valid polyhedron.

each recorded event corresponds to a real event and real events can't consume less than no energy at all.

Kellner used a software package called `dqed` to optimize Equation 10 for Equation 9. Because `dqed` returns different results after each run and none of these result is optimal, I used the simplex algorithm. An other problem of `dqed` is, that non-stable results would prevent to detect any tendencies of the increase or decrease of energy weights as done in Section 5.3.

The simplex algorithm finds the best point for a goal function in a given n-dimensional space under several constrains. Each benchmark is transformed into a hyperplane, dividing the space in a valid and an invalid part. The constraints $\overrightarrow{w} \geq 0$ and $\forall x : y_{bench_x} \geq 0$ can also be transformed into hyperplanes. In the resulting polyhedron, the minimum value of the goal function (Equation 10) is sought after. A visualization of this algorithm can be found in Figure 3.

The optimal energy weights were computed for each frequency, and all clock gating speeds.

5.3 Evaluation

The optimization showed, that on a Pentium D 830 three of the eight performance counters suggested by Kellner [Kel03] are not necessary. The simplex algorithm returned 0 for their energy weights. The other results can be seen in Table 2 and 3.

The energy weights are not to be confused with real energy usage. Kellner states that each event accounted means that there was a situation using some energy. I suggest, that the events should be seen as indicator for energy usage. This means, that energy weights could be negative. For example if there are two situations A and B, situation A is using 1 J, and performance counters α and β are increased in this situation. Situation B is using 2 J, and only performance counter α is increased. In this case, events counted by β, would have a weight of -1 J. But to stay comparable to previous works, only positive energy weights are used in this thesis.

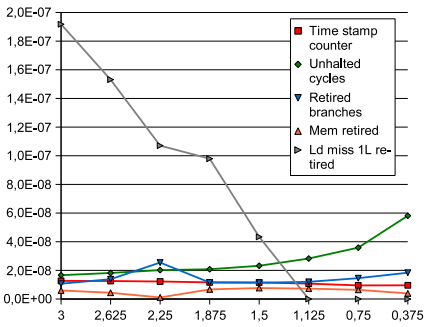

Figure 4: Weights of events for all clock gating speeds in J

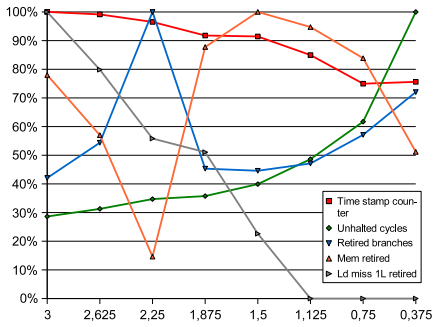

Figure 5: Weights of events for all clock gating speeds (relative)

Clock gating	8/8	7/8	6/8	5/8	4/8	3/8	2/8	1/8
Fake frequency (Ghz)	3	2.625	2.25	1.875	1.5	1.175	0.75	0.375
Event	Energy per event							
Timestamp counter	12.7	12.6	12.2	11.6	11.6	10.8	9.5	9.58
Unhalted cycles	16.6	18.2	20.2	20.8	23.2	28.3	35.9	58.1
Retired branches	10.7	13.9	25.5	11.5	11.4	12	14.6	18.4
Mem retired	5.98	4.37	1.12	6.72	7.66	7.26	6.43	3.92
Ld miss 1L retired	192	153	107	97.9	43.4	0	0	0

Table 2: Energy weights in nJ for a Pentium D 830, using only one core

In Figures 4 and 5 the energy weights (also in Table 2) are shown as curves. These curves demonstrate the changing of the weights for lower clock gating speed an frequency/voltage. Despite I can't prove it – because of too few frequencies/clock gating speed – it appears to be obvious, that the energy weights of the time-stamp counter slowly decreases and the one of the unhalted cycles increases proportional to the speed.

The curves of the energy weights of the retired branches and retired memory seem to depend on each other. The weight of the Ld miss 1L retired decreases rapidly and is 0 for the slowest two speeds. This can be understood, if it is considered that this counter describes energy usage, which is used, while waiting for the main memory, which only appears, if the CPU is much faster than the main memory.

Figures 6 and 7 show the energy weights for full speed and frequency scaling to 2.8 Ghz. The core voltage was automatically reduced to the minimum possible value. The Ld miss 1L retired shows the same tendency as while clock gating, but slightly stronger. While the energy weights for unhalted cycles and retired branches increase in lower clock gating speeds, the energy weights decrease while frequency scaling. As there are only two frequencies available on the Pentium D, no further prognoses seem to make sense.

The accuracy of the energy weights for frequency scaling can be seen in Figure 8. The "Ideal" line shows the maximum allowed power usage. As the benchmarks have a different power consumption and therefore only some are throttled at a specified maximum power (X-Axis) and the curves represend average values for all benchmarks, the curves to slowly approach the maximum power.

E.g. if we consider only two benchmarks b_1, and b_2 the average power usage curve would have two breaks, one from each benchmark.

$$powerUsage(b_1) := \left\{ \begin{array}{lll} 90 & for & maxPower \geq 90 \\ maxPower & for & maxPower < 90 \end{array} \right\} \quad (11)$$

$$powerUsage(b_2) := \left\{ \begin{array}{lll} 80 & for & maxPower \geq 80 \\ maxPower & for & maxPower < 80 \end{array} \right\} \quad (12)$$

From 115 to the first break it would be level and after the last break it will follow the ideal curve. In between the breaks it would have the half gradient as the ideal curve, because in this area only one benchmark is throttled and the other is still consuming its maximum power.

$$\frac{pU(b_1) + pU(b_2)}{2} = \left\{ \begin{array}{lll} 85 & for & maxPower \geq 90 \\ 40 + \frac{maxPower}{2} & for & 90 > maxPower \geq 80 \\ maxPower & for & maxPower < 80 \end{array} \right\} \quad (13)$$

For 3 Ghz the deviation from the ideal power consumption is about 3 W, when all benchmarks are throttled. For 2.8 Ghz, the deviation is about 3.5 W. This is less than 5%.

To summarize: Using hardware level technologies like lock gating and frequency/voltage scaling has a severe impact on the energy weights. This means, that for all possible combinations of settings for these technologies, the energy weights have to be precomputed, if no mathematical description of the energy weights' changing can be found.

As with the five performance counters and two frequencies is wasn't possible to find a mathematical description of the energy weights at different frequencies, I used the precomputed weights (Table 3).

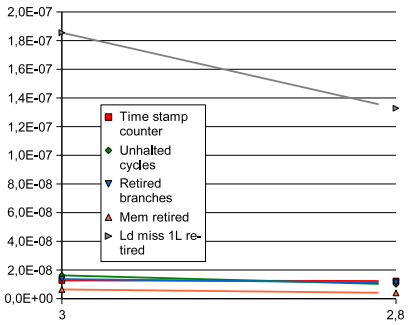

Figure 6: Weights of events for all frequencies in J

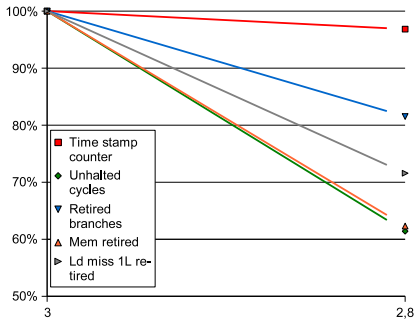

Figure 7: Weights of events for all frequencies (relative)

Frequency (Ghz)	3	2.8
Event	Energy per Event	
Timestamp counter	12.7	12.3
Unhalted cycles	16.2	9.97
Retired branches	13.6	11.1
Mem retired	6.41	4
Ld miss 1L retired	186	133

Table 3: Energy weights in nJ for a Pentium D 830, using only one core

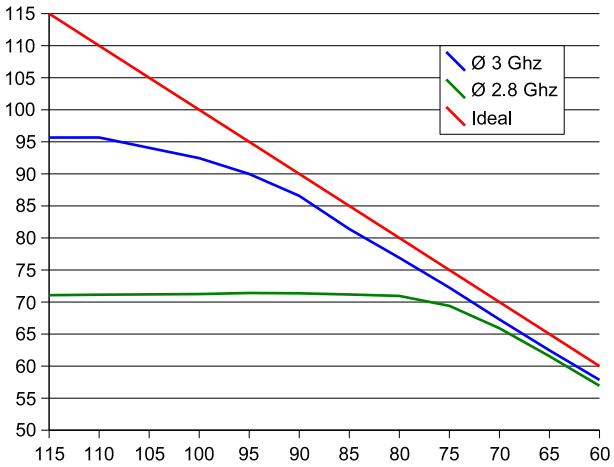

Figure 8: Average power usage for both frequency scaling settings. X-Axis is the maximum allowed power usage (W). Y-Axis is the actually used power. (W)

6 Implementation

To prove frequency/voltage scaling to be a reasonable technology for an energy aware system, I modified a Linux 2.6.17.3 kernel to support keeping energy budgets by idling and frequency/voltage scaling. The main scheduling function of Linux, `void ← schedule()`, selects the next thread to run at each timer interrupt. I modified the function in two ways.

The runtime is separated in periods of equal length. In each period there are several timer interrupts when `schedule()` is run.

At the end of each period, the scheduler predicts, at which frequency the next period would perform best. Therefore it computes Equations 4 to 7, and chooses the next frequency, as described in Section 4.2. The frequency is only adjusted every period, because it is time-expensive because the voltage is also adjusted. It is assumed, that the load will change slower than the periods.

At each timer interrupt the scheduler computes Equations 2 and 3, and if Equation 3 is true, the idle thread is scheduled, even if there is another runnable thread. This results in a sudden decrease of the used power, which make the power usage look like pinnacles of a castle, if you take a close look. If you only look at the complete period the pinnacles are not visible. There are some important variables, which can be adjusted:

maximum power: Defines how much power should be used. This is the most important value, because it makes the tradeoff between performance and power consumption.

overflow threshold: If there is not enough load to use the complete energy budget, the unused energy is shifted to the next period. In order to prevent the unused energy from growing too high, a threshold can be defined.

frequency: The system can either run at a fixed frequency or can dynamically adjust the frequency to use the least power, while not reducing the performance compared to full speed.

period length: This value is important, because a short period will result in inaccurate throttling and a too long period will result in an inhomogeneous power usage.

If the period is too short, the length of the idle time can't be set accurate. It can only be set to multiples of the timer interrupt period and the minimum timer interrupt period on a Pentium D is 1 ms. This can be too coarse grained, if the period is very short. E.g. if the period length is 10 and the idle time has to be $\frac{1}{3} \cdot periodLength$, there will be a inaccuracy of $\frac{10}{3} - 3 = \frac{1}{3}ms$, which is 10% of the idle time. Additionally the frequency/voltage transitions take some time. This time can be respected, but this is only needed if the period is very short. These inaccuracies cause a power misprediction and therefore an inaccurate throttling.

If the period is long, the pinnacles described above will be wide; Consequently, the idle thread is scheduled for quite a long time. This causes the system to stop for this time and may cause problems with some (soft) real time applications, like e.g. a movie player.

One problem during implementing was that it was only possible to readjust the throttling at some points, which may not be the perfect ones. This results in a misprediction of energy usage, because the length of the period is not perfectly predictable. To solve this problem, the period was chosen to be quite long (0.5 to 1 second). This is feasible on a server, running non-interactive tasks. For a faster responding system – e.g. a personal computer or a web server – the timer interrupt has to come more often to get the needed higher precision.

6.1 Evaluation

To evaluate the implementation, the same system as described in Section 5.1 was used. It is not part of this work to prove that energy estimation from performance counters is possible, as this was done elsewhere [BKW03].

The goal of this system was to achieve maximum performance by using less than a given maximum power. Another possible goal would be to use less energy, while preserving a certain performance or some mixture of these goals.

In Section 4, an algorithm is described, how to determine the optimal frequency for a load. In Figure 9, the average time for all benchmarks is shown. The dynamic-curve using this algorithm is always very close to the best frequency. Between 106 W and 96 W it is below both frequencies, because for some benchmarks, full speed is better and for some others reduced speed. If the CPU would support more frequencies, this effect would be much stronger, and would appear at most maximum power settings.

Figure 10 shows the same as Figure 9, but for $bench_1$ only. You can see a misprediction between for 90 W and 88 W maximum power. This is caused by the fixed *speedRatio* values, by the small error in predicting energy usage, and the problems with the inaccurate measurements due to the implementation. The design also doesn't care about the transition time spent when changing frequency and voltage. The other benchmarks show a very similar behaviour.

In general, the system showed good results and proved frequency scaling to be a feasible technology for an energy aware system. The problems described above can be solved by modifing Linux deeper or by using another OS.

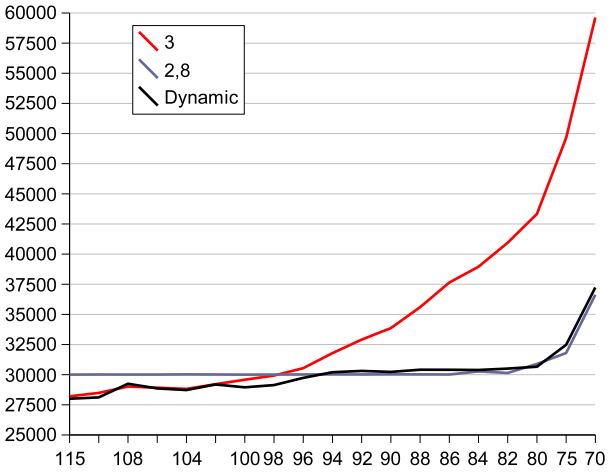

Figure 9: Average time in ms of the benchmarks for different values of maximum power. The first two and the last 3 X-Axis Sections have a distance of five. The rest have a distance of two.

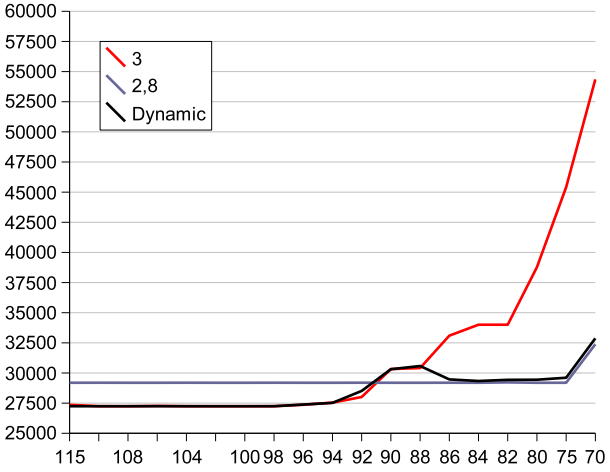

Figure 10: Used time in ms of $bench_1$ the benchmarks for different values of maximum power. The first two and the last three X-Axis Sections have a distance of five. The rest have a distance of two.

7 Conclusion and future work

7.1 Conclusion

This work presents an approach to use power saving technologies like frequency/voltage scaling and clock gating for constructing an energy aware system. This system was implemented and evaluated and proved to be much better than the energy aware systems that were constructed up to now. At lower maximum power the performance advantage was up to 100% compared to a system only using idling for throttling.

Clock gating was found not to be useful in terms of power saving, as used in this work. Clock gating saves power, but has a big impact on performance. The benefits of clock gating are less than the disadvantages. Therefore clock gating was not used in the implementation.

Frequency scaling proved to be a good technology. Since frequency scaling enables voltage scaling, the benefit of the combination of this technologies is impressive. For a small maximum power setting, the system performed two and more times as fast as without frequency/voltage scaling and clock gating. Even despite there were only two frequencies available, the algorithm developed to predict the best frequency would be usable for more frequencies.

As this energy aware system relies on performance counters to estimate the current power usage, this work also investigates the impact of frequency/voltage scaling on their energy weights. As there were only two frequencies available on the Pentium 4/D it was not possible to draw real conclusions from the measurements. The measurements of different clock gating speeds show some tendencies, described in Section 5.3

For evaluation purposes an energy aware system was implemented and it shows the possibility of the approach. The algorithm detecting the best frequency for a given load returns reasonable results. Measurements show, that dynamic frequency choosing performs as good or better than any predefined, fixed frequency, if more than one benchmark is run.

7.2 Future work

Although the performance counters of Section 5 give a reasonable energy estimation, it is not proved to be the best possible estimation. As I used a better optimizing and more reliable algorithm than Kellner had used, the choice of the used performance counters should be redone. Since the energy weights are still a product of an optimization algorithm and are dependent on the benchmarks, there will most likely be no perfect energy estimation.

What can be done is to enhance the throttling process, either by more exact measurements and timers, or further power saving features.

If an energy aware system should be used in an productive environment, there is a need for an automatic energy weights computation tool or support from the chip manufactures. The next step would be to include the heat dissipation of the harddisk, RAM, and the graphics adapter. In the end the system can be extended to control the energy usage of complete server farms.

A Benchmarks

Number	Name	Description
1	PrimeCount	Counts prime numbers in a given interval
3	Sort	Selection sort
4	Not	Executes logical NOT
5	Memory write	Continuous writes to every byte of a 128 MByte sized array
6	Add	Executes ADD
7	Tree search	Searches numbers in a tree
8	Div	Executes DIV for integers
9	Mult	Executes MULT
10	Cache	Continuous writes to every byte of a 32 KByte sized array
11	Div float	Executes DIV for floats
12	Sqrt	Computes some square roots
13	Memory read	Similar To “Memory read 2”, but not continuous
14	Xor	Executes logical XOR
16	Memory read 2	Same a Memory write, but reading
19	Guess number	Binary search for a random number
20	Spin	while(1) for some time

B How to compute the *workTime* for a specified frequency

The power usage of each period can be described as:

$$\begin{aligned} & workTime_{freq} \cdot powerWhileWorking_{freq} \\ + \quad & idleTime \cdot idlePower \\ = \quad & maxPower \cdot periodLength \end{aligned} \tag{14}$$

As it is assumed, that the power characteristics are similar to the one of the last period $powerWhileWorking = usedPower_{freq}$, while $usedPower_{freq}$ is the power used, when not running the idle thread. Additionally you have to replace $idleTime$ by $periodLength - workTime_{freq}$.

$$\begin{aligned} & workTime_{freq} \cdot usedPower_{freq} \\ + \quad & (periodLength - workTime_{freq}) \cdot idlePower \\ = \quad & maxPower \cdot periodLength \end{aligned} \tag{15}$$

If you solve Equation 15 to $worktime_{freq}$ you get

$$workTime_{freq} = \frac{periodLength \cdot (maxPower - idlePower)}{usedPower_{freq} - idlePower} \tag{16}$$

which is Equation 6.

References

[Bel01] Frank Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. Technical Report TR-I4-01-11, University of Erlangen, Germany, December 14 2001.

[BKW03] Frank Bellosa, Simon Kellner, and Andreas Weissel. Event-driven energy accounting for dynamic thermal management. Technical Report TR-I4-03-02, University of Erlangen, Germany, July 30 2003.

[BM01] David Brooks and Margaret Martonosi. Dynamic thermal management for high-performance microprocessors. *hpca*, 00:0171, 2001.

[Int03] Intel Corp. *Events on Intel Pentium 4 Processors, Including Hyper-threading Technology-Enabled Processors*, July 2003.

[Int04] Intel Corp. *IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, 2004.

[Int05] Intel Corp. *IA-32 Intel Architecture Optimization Reference Manual*, June 2005.

[Int06] Intel Corp. *Intel Pentium D Processor 800 Sequence Datasheet*, February 2006.

[Kel03] Simon Kellner. Event-driven temperature control in operating systems. Study thesis, University of Erlangen, Germany, December 1 2003.

[Mer05] Andreas Merkel. Balancing power consumption in multiprocessor systems. Diploma thesis, System Architecture Group, University of Karlsruhe, Germany, September 30 2005.