Process Cruise Control

Event-Driven Clock Scaling for Dynamic Power Management

Andreas Weissel
University of Erlangen
weissel@cs.fau.de

Frank Bellosa
University of Erlangen
bellosa@cs.fau.de

ABSTRACT

Scalability of the core frequency is a common feature of low-power processor architectures. Many heuristics for frequency scaling were proposed in the past to find the best trade-off between energy efficiency and computational performance. With complex applications exhibiting unpredictable behavior these heuristics cannot reliably adjust the operation point of the hardware because they do not know where the energy is spent and why the performance is lost.

Embedded hardware monitors in the form of event counters have proven to offer valuable information in the field of performance analysis. We will demonstrate that counter values can also reveal the power-specific characteristics of a thread.

In this paper we propose an energy-aware scheduling policy for non-real-time operating systems that benefits from event counters. By exploiting the information from these counters, the scheduler determines the appropriate clock frequency for each individual thread running in a time-sharing environment. A recurrent analysis of the thread-specific energy and performance profile allows an adjustment of the frequency to the behavioral changes of the application. While the clock frequency may vary in a wide range, the application performance should only suffer slightly (e.g. with 10% performance loss compared to the execution at the highest clock speed). Because of the similarity to a car cruise control, we called our scheduling policy *Process Cruise Control*. This adaptive clock scaling is accomplished by the operating system without any application support.

Process Cruise Control has been implemented on the Intel XScale architecture, that offers a variety of frequencies and a set of configurable event counters. Energy measurements of the target architecture under variable load show the advantage of the proposed approach.

Categories and Subject Descriptors

D.4.1 [**Software**]: Operating Systems—*scheduling / process management*

CASES 2002, October 8–11, 2002, Grenoble, France.

General Terms

Algorithms, Measurement, Performance

Keywords

Power Management, Scheduling, Clock Scaling, Event counters

1. INTRODUCTION

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

This paper contributes to this awareness and initiates a new approach in system software: the on-line evaluation of counters that register performance- and energy-critical events. We will show that there is not only a correlation between events and performance but also between events and power consumption. By exploiting these counters the operating system has the complete knowledge

- where the energy has been consumed
- where the time has been spent, and
- who has been responsible for the use of energy.

According to the individual demands of each application, power management can find a trade-off between energy consumption and quality of service demands. To fulfill this task an operating system has a variety of options for the activation and configuration of hardware components. Not only the time of activity, but also the degree of activity can be controlled. Our approach to an integrated energy monitoring system respects these power states and provides the essential information for advanced power management policies.

In this paper we focus on two energy-critical hardware components, the CPU and the memory, because the use of both components can already be monitored by the performance monitoring counters found in many contemporary processor architectures. To demonstrate the energy savings possible with Process Cruise Control we chose the Intel XScale architecture as our target platform. The Intel XScale supports dynamic frequency scaling which can be used to

save energy. Other architectures like Intel Speedstep-M and AMD Mobile Athlon are ready for Process Cruise Control. With the acceptance of event counters as a prerequisite for reliable power-management decisions we expect more low-power architectures to offer event-counters, some of the counters even specially dedicated to energy profiling.

This paper is organized as follows: In the next section, we investigate the energy characteristics of advanced processor architectures. This motivates our scheduling approach Process Cruise Control that is presented in section 3. In Section 4 we describe the implementation in an embedded Linux operating system and we exhibit energy measurements that validate the benefits of our approach. Finally in section 5, we propose further architectural innovations advantageous for Process Cruise Control.

2. ENERGY CHARACTERISTICS OF PROCESSOR AND MEMORY

2.1 Characteristics of Interest

In semiconductor technology, energy is used whenever current is flowing due to leakage or due to loading/de-loading of capacitors triggered by transistor switch operations. The leakage current depends on static parameters like time, voltage and properties of the semiconductors. In addition to these static parameters the dynamic energy consumption depends on the switching frequency of the gates.

If we want to identify those parts of the CPU/memory complex which contribute significantly to the total energy consumption, we have to look at those parts containing most of the capacitors and those with the highest switching frequencies:

- The processor core executing algorithmic, logical, or control flow operations consumes energy depending on the switching activity within the essential functional units. We expect some relation between energy consumption and clock frequency. However, a major part of the activity and thus energy consumption depends on the type of instructions and their operands. Therefore, we have to keep an eye on the activity of each functional unit.

- If a memory management unit (MMU) is used in a computer architecture for reasons of mapping and protection, the MMU will contribute significantly to the energy consumption as it is built-up from full associative memory that is accessed whenever memory is referenced. Therefore, the energy consumption might depend on the memory-reference patterns of the executed software.

- Caches contribute to static energy consumption depending on their size but also to dynamic energy consumption depending on the frequency of cache references and the associativity of the cache indexing algorithm. The higher the associativity, the more cache-tags have to be compared at each cache reference.

- Dynamic random access memory (DRAM) is build up of capacitors to store information. As these capacitors have to be recharged to keep their information, we expect a significant contribution of memory to the static energy consumption depending on the size of memory. Due to several decode and multiplex stages and the complex transfer and switching activity, we should see a high dynamic energy consumption of DRAM.

  To satisfy memory requests, data has to be transferred between the data-rows which make up a memory bank and the sense amplifiers. Several decode and multiplex stages have to be passed to move data to the output drivers and from the receiver registers. Because of this comprehensive transfer and switching activity, we should see a high dynamic energy consumption of DRAM.
  Advanced memory modules (e.g., RDRAM) also offer several low-power states which differ in the latency to re-activate the memory module again. In a low-power state some parts of the address-logic and the bus connectors of the module are shut down. This saving in passive energy consumption has to be paid by a higher access latency.

- The interconnection network (e.g., the bus system) contributes mainly to the dynamic energy consumption as the capacitance of the bus-lines has to be loaded (unloaded) at each bus cycle. Therefore, we expect an energy consumption which is related to the activity level of the interconnect.

As we want to influence the energy efficiency of a system by a task specific clock frequency we have to identify

- which components benefit from clock scaling and those which do not.

- which components are used by a specific task.

- which consequences on the speed of executions a variation in clock speed will entail.

To identify the energy-specific characteristics of a system with configurable clock speed that also offers the opportunity for performance analysis and event-driven energy-accounting, we explored the energy consumption of an Intel IQ80310 system [14].

The Intel XScale 80200 processor used in this system can operate at clock speeds from 333 MHz to 733 MHz [12], [11], [13]. The high-speed core clock is produced by a programmable clock multiplier. Changing the clock frequency is done by writing the multiplication factor into a configuration register. Although the processor can operate at different core voltages depending on the selected core frequency, we could not scale the voltage when modifying the clock frequency (dynamic voltage scaling DVS), because the voltage could not be adjusted with the IQ80310 evaluation board. To get an idea of how much additional energy can be saved by voltage scaling, we calculated the minimum possible energy savings which can be achieved by reducing the core voltage to the values given in the Intel 80200 datasheet [13].

The processor instruction set is compliant to the ARM V5TE instruction set. It implements a 32-KByte, 32-way set associative data- and instruction cache with a line size of 32 bytes. The policy of the data cache is configured to write-back. The instruction and data MMU implements a 32 entry full associative TLB. The processor offers performance monitoring counters to register events like executed instructions, cache references & misses, and memory requests.

Currently several microprocessor architectures support dynamic frequency scaling and performance monitoring. Some offer a wide variety of selectable speeds as for example the Athlon 4 PowerNow! and the Alchemy Au1100 from AMD. Others like Intel's Pentium III and 4 with Speedstep Technology at the time just distinguish slow and fast modes. Transmeta's Crusoe Processor is not a candidate for Process Cruise Control because frequency and voltage scaling (LongRun Power Management) is performed internally as part of the code morphing engine [5].

2.2 Measurement Set-Up

The 80200 processor is available with the Intel IQ 80310 evaluation platform. The IQ80310 system is built up as a PCI board plugged into a PCI slot of a host PC or PCI backplane. The power is supplied by the 3.3 V supply voltage pins of the PCI slot. This modular form, in contrast to closed systems like laptops or notebooks, makes power-measurement possible.

To measure the energy consumption, a sense resistor of low resistance (0.02 Ω) is placed in series with the power supply for current measurement. A data acquisition system with a sampling frequency of up to 3000 KHz guarantees a high temporal resolution of the measurements. To separate the static (idle) power of the processor, chip-set and memory from the dynamic (active) power, we measured the power consumption when the processor is in the idle-state and the dynamic part when the CPU is busy. Just the dynamic power consumption is essential for further investigations in this paper. The idle power is constant for all clock frequencies.

Apart from the dynamic energy consumption of the CPU and memory, our measurements also comprise the dynamic energy consumption of the voltage regulators and the chip set.

2.3 Basic System Energy Characteristics

Goal of the basic measurements is to determine the variation in the power consumption of the processor and the memory. We executed several simple micro-benchmarks that involve different sets of functional units. During all the runs the processor was constantly busy. For the presentation in this paper we selected some micro-benchmarks which exhibit interesting characteristics of today's computer architectures (see figure and table 1).

First, we ran an arithmetic test (`add reg`) doing arithmetic operations with all operands in registers, followed by a synthetic application (`goto label`) to investigate the influence of branches on the energy consumption. One of the tests (`call function`) passes parameters to subroutines. As the stack is used to pass parameters we see a mix of normal operations on registers, branches and cache references. Finally several tests were executed that trigger read and write cache- and memory-operations (`read`, `read/write L1 cache` and `memory`).

The dynamic power consumption of the low-power XScale system (see figure 1) is quite stable for CPU intensive applications without main memory requests. Here the power drain ranges between 570 mW and 800 mW at a clock frequency of 733 MHz. As soon as the main memory serves requests, the memory controller and the SDRAM module are involved. This leads to a boost in energy consumption up to 1220 mW.

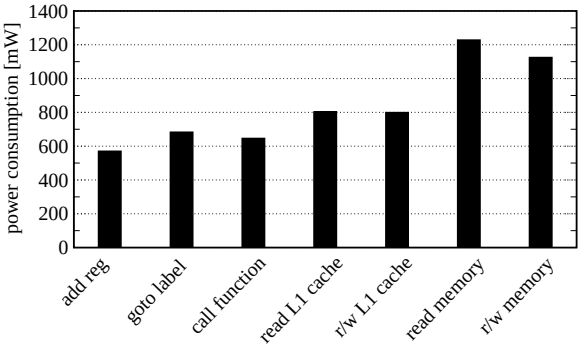

Figure 1: IQ80310 board power breakdown (dynamic CPU & memory power consumption)

benchmark	instr. per μs	branches per μs	L1 ref. per μs	mem.req. per μs
`add reg`	680	12	0	0
`goto label`	313	104	0	0
`call function`	379	52	143	0
`read L1 cache`	548	46	182	0
`r/w L1 cache`	578	38	307	0
`read memory`	85	7	28	3.7
`r/w memory`	43	3	23	4.3

Table 1: rates of characteristic events

Four factors which can be monitored with the performance monitoring counters seem to influence the energy consumption (see table 1): the rate of executed instructions, of executed branches, of data cache references and the rate of memory requests. While the number of executed instructions seems to have no impact on energy consumption, the frequency of branches (when executing `goto label`), the activity of the MMU and the caches (`call function`, `read`, `read/write L1 cache`) increases the energy needs of the processor core.

The values of the typical power dissipation given in the datasheet of the Intel 80200 processor [13] coincide with our measurements of the benchmarks `call function` and `goto label`.

2.4 Energy/Performance Characteristics

The effect of frequency scaling on the performance and system energy consumption is demonstrated in figures 2 and 3.

Figure 2 shows a linear increase in performance for all CPU- and cache intensive applications. Doubling the clock speed results in twice the performance. In the same way the dynamic energy consumption rises linear with the clock frequency for those applications (see Figure 3).

While memory intensive applications show the same linearity in energy consumption they suffer in performance because the processor stalls in a busy mode while waiting for memory requests to be served. Up to four 32 byte read requests can be outstanding in a fill buffer before the XScale 80200 needs to stall. Furthermore 8 write buffers of 16 bytes help to buffer data while the bus is not available. The read memory application can tolerate the memory latency to a certain degree because of the fill buffer, whereas the

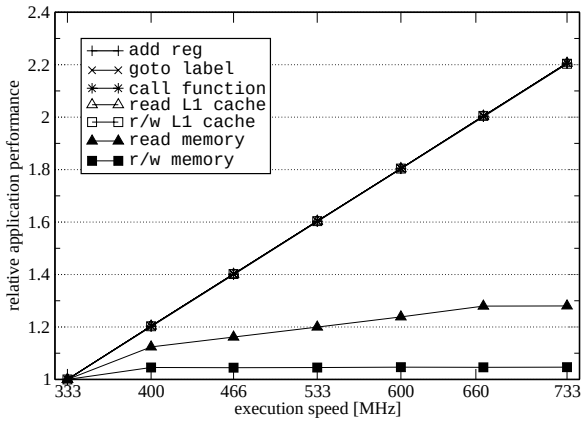

Figure 2: relative application performance at various clock speeds

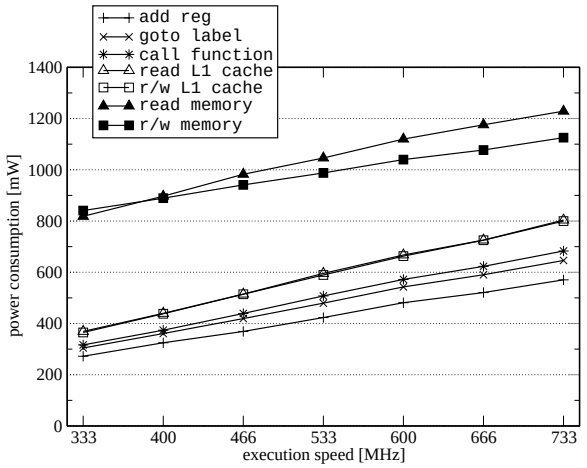

Figure 3: power consumption of the IQ80310 board (XScale 333-733 MHz) at various clock speeds

read/write memory application stalls for each new allocated cache line because a dirty cache-line has to be written back to memory before a new one can be stored.

To show the relation between application performance and energy consumption, the energy consumption is divided by the application performance and normalized relative to the value at 733 MHz (see figure 4). We call this value the *energy performance ratio*. An energy performance of *ep* at a certain clock speed *cs* means that running at *cs* MHz needs *ep*% of the energy to fulfill the task at 733 MHz.

CPU intensive applications operate energy efficiently at all clock speeds. They are even a little bit less efficient (*ep* = 105%) at lower clock speed. The reason is the constant overhead of periodic kernel activities (e.g., timer interrupt processing). The percentage of cycles for these operating system activities rises with slower clock frequency and fewer cycles remain for the execution of the applications. Consequently it requires slightly more energy to execute the application at low speed.

From the point of energy efficiency it does not pay off to run the CPU and cache intensive applications at lower clock speed. Therefore we can run those applications at the highest speed.

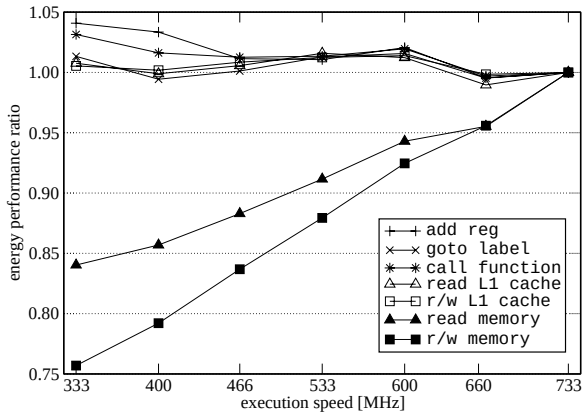

Figure 4: relation between application performance and energy consumption

The more memory requests are issued by an application the more it pays off to drive the application at low speed. Energy savings of 22% (*ep* = 78% for read/write memory at 333 MHz) to complete the same task are possible without a substantial reduction in application performance (4% performance loss for read/write memory at 333 MHz as shown in figure 2).

Depending on the memory reference characteristics of an application we can save a significant amount of energy without severe losses in performance by running the application at the suitable clock speed.

3. PROCESS CRUISE CONTROL

Performance and energy consumption at variable speeds are two characteristics which are correlated, but the degree of correlation depends on the use of performance- and energy-critical hardware components. Only if the operating system knows the component-specific usage patterns of each of the managed execution entities (threads or processes), it can find the best energy/performance trade-off and select the right speed of execution.

3.1 The Policy Model

Our approach to find the patterns is the on-line evaluation of event-counters. For a specific architecture we have to find a set of countable events that characterize the behavior of a thread concerning performance and energy consumption when the thread is executed at various clock frequencies.

The rates at which these events can happen at a certain clock frequency span a multidimensional space which describes all the potential patterns a thread could exhibit. For each point in the space we can find the proper clock frequency that minimizes the energy consumption for a given performance requirement (the *optimal (clock) speed*).

We are facing the challenge to partition this space into domains with equal clock frequency and to describe these partitions (frequency domains) in a way that the scheduler of the operating system can determine the clock speed of a specific thread by a fast mapping from event rates to clock frequencies. The optimal speeds for the various event rates and, consequently, the resulting partitions depend on the restriction in performance loss. The current set of partitions defines the *model policy*.

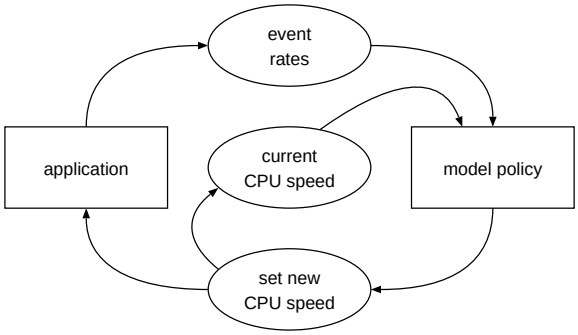

Figure 5: Process Cruise Control loop

A scheduler implementing Process Cruise Control adapts the clock speed when switching from one thread to another. The new frequency is determined by a periodic evaluation of the event rates in the latest history of the thread. Therefore the scheduler has to find the frequency domain that matches all the event rates of the thread.

We saw that while the energy specific characteristics of a single thread change only slowly over time, the characteristics of concurrently running threads alter frequently and show a wide variation. For a single thread it is sufficient to analyze the behavior of this thread at each context switch. Our approach respects the variation in behavior by adapting the clock speed at each thread switch according to the characteristics of this thread. This guarantees an optimal speed adaptation.

The data flow in figure 5 shows the relation of the Process Cruise Control model to a car cruise control or any other controlled system. Sampled event rates along with corresponding execution speeds run into the model policy. An optimal speed prognosis is made and applied to the application. Again, the new CPU speed is fed into the model along with newly measured event rates closing the loop of control.

To summarize, our approach can be outlined as follows:

1. Determine the correlation between the rates of different events and both performance and energy consumption.

2. Identify the lowest possible clock frequency (the *optimal speed*) for a certain combination of event rates under an user-specified upper bound for performance degradation.

3. On each task-switch, scale the clock frequency according to the pre-computed optimal speed for the thread-specific event rates.

3.2 XScale Frequency Domains

The Intel XScale 80200 processor implements one clock and two event counters. Under these restrictions, and considering the energy/performance characteristics of section 2.4 and table 1, the selection of the following events is recommended:

- The *memory requests per clock cycle* clearly indicate the degree of memory use. The higher the rate of memory requests the more the energy performance will benefit from a reduction in clock speed.

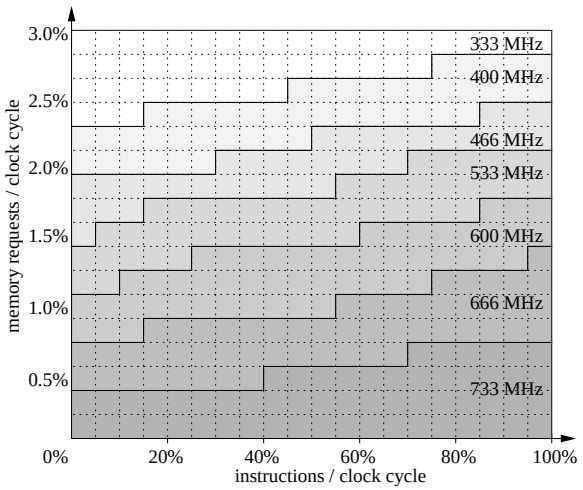

Figure 6: XScale frequency domains

- The *instructions per clock cycle* indicate the sensitivity for a performance loss due to speed reduction. The lower the rate of executed instructions the less the performance of a thread will suffer from a reduction in clock speed.

Cache misses as an indicator for energy consumption can not be used because several event counters for all types of energy-relevant cache events are not available on the target architecture. Furthermore several counters for different cache events would have to be monitored in parallel. Therefore we used the counter for memory requests because they showed the best correlation to energy consumption.

To span the space of both event rates we constructed micro benchmarks producing various event rates. For each clock speed, we determined the event rates for each of these benchmarks.

The next step is to find the minimal clock speed which can be tolerated for given performance requirements. For our tests we chose 10% as an acceptable performance loss.

The last step is to partition the two-dimensional space into frequency domains.

We chose a simple approach with matrices that define the frequency domains (for an example see figure 6). The dimensions of the matrix are the event rates, the percentage of instructions per cycle, ranging from 0% to 100%, and the percentage of memory requests per cycle, ranging from 0% to 3% (this is the maximum value achievable by artificial micro benchmarks). A simple matrix look-up operation yields the optimal clock speed.

4. PROCESS CRUISE CONTROL IN LINUX

4.1 Implementation

The enhancement of Linux 2.4.18 to support event-driven frequency scaling comprises several modifications, summing up to almost 1000 lines of code:

- Context switch routines and kernel data structures are modified to collect and hold, for each thread, the values of the two available performance-monitoring event counters and the clock speed. Thread-specific event

modules	overhead in cycles
Linux scheduler	> 26000
Process Cruise Control	3000 – 4000
· read perf. counter	200
· switch frequency	1000 – 2300 (= 3.1 μs)

Table 2: overhead of Process Cruise Control

counters are updated within the timer interrupt processing.

- The computation of the event ratios and the determination of the optimal clock frequency of the current thread as well as the speed adjustment for the next thread to run is performed in the main task switch routine (`schedule()`).
- Each thread can be configured to run at a thread-specific fixed speed or to run with dynamic event-driven clock-scaling. We extended the context switch routines for thread-specific speed tuning and offer the necessary user-interfaces (`/proc/scale`).
- The determination of the optimal clock frequency with respect to the thread-specific event rates is done by look-up tables (matrices) in the Linux scheduler. The content of these tables can be set during runtime by the module `policy` which provides an interface in the `/proc`-filesystem. A default model policy is provided.

With all these modifications to the Linux scheduler there arises the overhead question. Two modifications contribute to the overhead (see table 2):

- the determination of the optimal clock frequency (at every thread switch)
  Division operations are necessary to obtain event rates from the performance counter values and offsets into the policy-matrix from event rates.
- the frequency switch

Our initial implementation increases the overhead of the scheduler by approx. 15% with a high variation due to a variable number of compulsory cache misses.

4.2 Measurements

To show the benefits of event-driven clock scaling we measured the energy consumption and performance of four simple applications to find the optimal clock speed according to external energy measurements. Figures 7 and 8 show the energy/performance characteristics of these applications which exhibit different behavior. `find|grep` searches for a string in the RAM file system, `gzip` compresses a shared library, `djpeg` decompresses a JPEG file to an image file and finally, `factor` factors a number. We ran the four applications at all possible clock frequencies to determine the energy consumption, performance and the optimal speed according to the allowed penalty of 10% performance loss.

Furthermore Process Cruise Control determined the rate of memory requests and executed instructions. Table 3 summarizes the results. For three tests (`find|grep`, `gzip` and `factor`) Process Cruise Control comes to the same clock speed as the external measurements. One application `djpeg`

application	optimal speed	Process Cruise Control clock scaling	Process Cruise Control energy savings
`find|grep`	400 MHz	400 MHz	15%
`gzip`	466 MHz	466 MHz	10%
`djpeg`	600 MHz	533 MHz	8%
`factor`	600 MHz	600 MHz	4%

Table 3: Process Cruise Control clock speed and energy savings

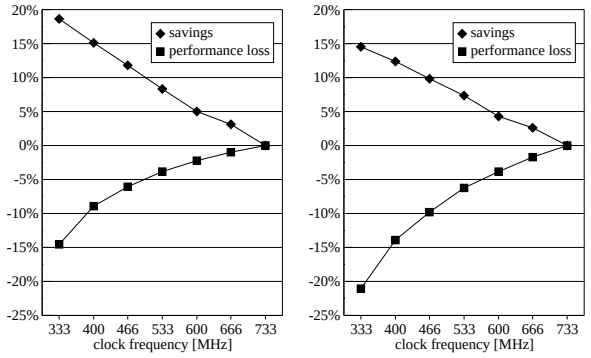

Figure 7: `find|grep` (left) and `gzip` (right) energy/ performance profile

is scheduled with a speed that is one step to low. The application would suffer a performance loss of 12% which is below the tolerated limit of 10%. In almost all cases the optimal clock speed is reached after one or two iterations.

Additionally we designed an inhomogeneous test application which involves several subtasks (see table 4). The split up savings of the individual tasks are shown in figure 9.

As expected, very memory intensive tasks like `free db` and `memcpy` gain energy-efficiency from Process Cruise Control without losing measurable performance. Average memory access rates from `fill string` and `file rw` result in energy savings with considerable but limited impact on performance.

Tests with low memory access rates as `read db`, `sort db` and `search db` become slowed down considerably. However, constraints for maximum performance loss are satisfied. De-

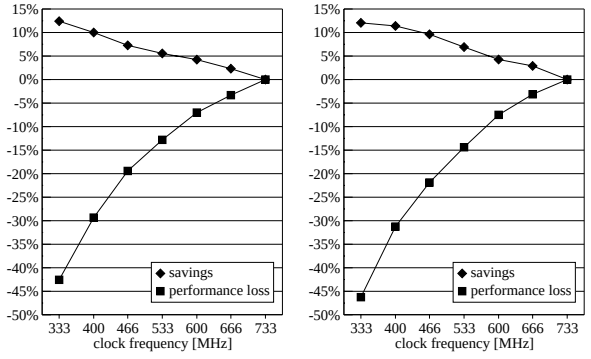

Figure 8: `djpeg` (left) and `factor` (right) energy/ performance profile

sub-task	description
`loop`	simple `for`-loop
`mult`	multiplication loop
`primes`	Eratosthenes' sieve
`file rw`	write data to a file and verify
`read db`	read word database into memory
`sort db`	quick sort
`search db`	full text search in database
`fill string`	dump word database using `strcat`
`free db`	release memory
`memcpy`	swap blocks of memory

Table 4: sub-tasks of the inhomogeneous application

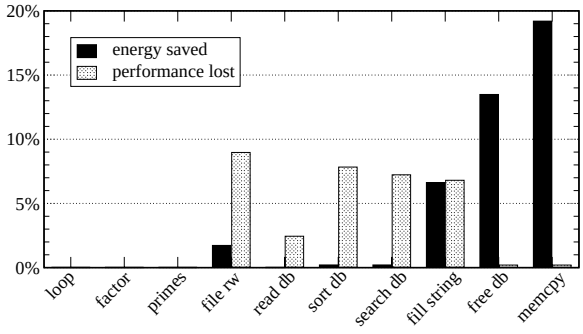

Figure 9: energy savings of the inhomogeneous application

pending strongly on CPU speed, the performance of these sub-tasks suffers most, compared to other tests.

Sub-tasks with very low memory access rates like `mult`, `loop` and `primes` are not slowed down making performance loss disappear.

In another test `ghostscript` was used to convert postscript files to several different device formats. During program run Process Cruise Control switches between clock frequencies very often; the computed optimal speeds range from 333 to 733 MHz. Overall energy savings of 4.5% are achieved while not exceeding 10% performance loss. Figure 10 shows the clock frequency switches during a run of ghostscript.

The runtime behavior of the postscript interpreter is influenced by the content of the postscript file to convert. Therefore the frequency switching pattern of the process run depends on the content and structure of the input file. Figure 11 shows the frequency switches of two runs of ghostscript.

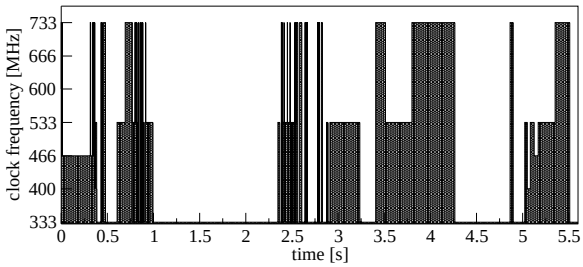

Figure 10: clock frequency switches for ghostscript

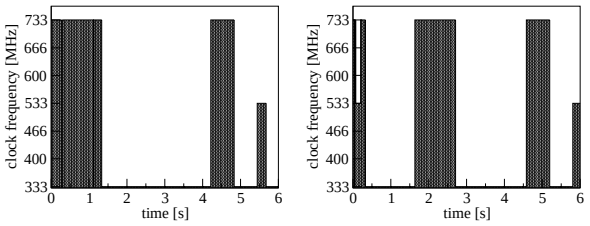

Figure 11: ghostscript processing two similar files

clock frequ. [MHz]	333	400	466	533	600	666	733
min. core voltage [V]	1.0	1.1	1.1	1.1	1.3	1.3	1.5

Table 5: minimum core supply voltages for the Intel 80200 processor

The processed postscript files both contain one image and text; the only difference is the position of the image in the text. The patterns of the frequency switches reflect the different positions of the image in the source files.

We could prove that it is possible to come very close to the optimal clock frequency by an on-line evaluation of event counters. For the Intel IQ80310 system energy savings of 15% are possible without severe performance impact. If we tolerated a higher performance degradation, the energy efficiency could be improved further.

To get an impression of how much energy can be saved by frequency *and voltage* scaling, we calculated the minimum additional energy savings which can be achieved by reducing the core voltage to the values given in the Intel 80200 datasheet [13] (see table 5). In previous measurements the core voltage was fixed at 1.5 V.

As can be seen in figure 3, the energy values of the `add reg` benchmark represent the lower bound in power consumption of the active Intel 80210 processor. We used these values to determine the minimum additional energy savings possible with voltage scaling. Figure 12 shows the results for the applications `gzip` and `factor`. The steps in the figure between 333 and 400 MHz, 533 and 600 MHz and 666 and 733 MHz reflect the different values for the minimum core voltages (see table 5).

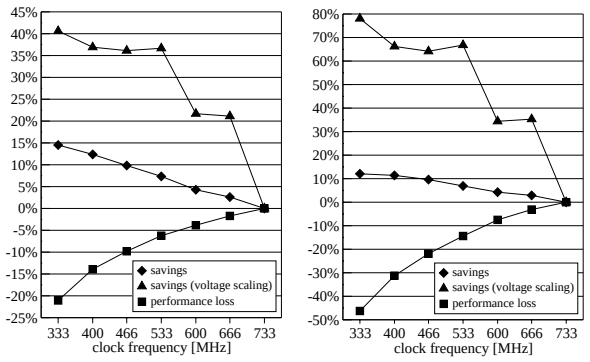

Figure 12: `gzip` (left) and `factor` (right) energy/ performance profile with voltage scaling

5. ENERGY MONITORING COUNTERS

The performance monitoring counters implemented in contemporary architectures hold a number of drawbacks:

- The selection of countable events was done to support performance profiling and not energy profiling.
  Several events which differ substantially in their energy consumption cannot be differentiated. An example are move and arithmetic/logical instructions. Move instructions need less energy, but is takes the same amount of time.
  Another example are read and write memory requests which differ in their energy consumption [17]. Event counters should register the type of memory request and the number of row activations in the memory modules.

- Because we use the performance monitoring counters for energy/performance characterizations of threads, they cannot be used for application profiling. Therefore an energy-aware system should implement two sets of counters: a set of performance monitoring counters, and a set of energy monitoring counters.

- The value of the event counters is sampled at special points in the operating system code. Sampling implies some overhead, so we cannot afford to sample at the beginning and at the end of interrupt service routines and other short running kernel activities. Consequently, kernel activity is accounted to the currently running thread which is not beneficial for a precise characterization.
  Therefore two sets of energy monitoring counters, one for user- and another for kernel activity, is recommended to reduce the number of sampling points and to increase the accuracy of the characterization.

6. RELATED WORK

This research contributes to the work on system power-analysis and dynamic clock-scaling for non-real-time systems. The innovative approach is the use of event-counters to enable reasonable energy/performance tradeoffs in a time-sharing scheduler.

Event-counters were explored extensively in the context of static performance analysis (e.g., see [1]). The first on-line evaluation of event counters for the purpose of scheduling was reported in [4]. The focus of this work was on cache affinity scheduling in NUMA architectures. Furthermore, the operating system can manage the memory-bandwidth in multiprocessors at run-time if counters provide information about memory requests and memory stall cycles.

Recent work employs event counters for run-time power estimations and energy accounting and throttling [2], [16], [3].

There is a rising number of papers which describe the power measurement of systems which support dynamic clock scaling, partly in combination with dynamic voltage scaling [6], [7], [15], [20].

Several papers cover the topic of scheduling on systems with variable speed. Most of them focus on real-time scheduling [18], [10], [19]. Here the scaling factor is reduced as long as the real-time requirements are fulfilled.

When clock speed becomes a novel system parameter, there is a need for system interfaces to control this parameter. Unlike our simple architecture-specific `/proc/scale` interface, the BUFScale API [8] offers an architecture independent interface for power system tools and applications.

Gurumurthi developed a complete system power simulator, called SoftWatt, that is built on top of the SimOS infrastructure [9]. As demonstrated in the previous section, the performance monitoring counters of existing architectures impose certain limits on event-driven power management mechanisms like Process Cruise Control. With SoftWatt it is possible to fully exploit the potential of event-driven power management for a complete system.

7. CONCLUSION

In the past, scheduling had a single dimension. The scheduler had to decide which thread of control should run on the CPU. With the emergence of power-sensitive devices we enlarge the freedom of scheduling to a second dimension, the speed of execution to control power-consumption effects. Within the space of decision, the scheduler has to control the execution in a way that the scheduling goals are achieved.

In this paper we have tackled the problem of finding the optimal process-specific execution speed in a time-sharing environment. The more the operating system knows what is going on inside the hardware the better it can react on changing usage patterns. We have identified event counters as the valuable source of information for the operating system scheduler. The reading of event counters and the processing of counter values does not imply a high overhead so we have found a cheap and easy methodology to detect thread-specific usage patterns for power characterization.

Once we have characterized a system according to certain performance requirements, Process Cruise Control determines the optimal clock frequency of a thread according to its patterns.

Our approach does not impose any hints on the application. Process Cruise Control minimizes the energy consumption while it schedules unmodified and uninstrumented code with just minor performance penalties.

A prototype implementation on a low-power Intel XScale evaluation system running Linux shows energy savings of 22% for memory intensive applications. We calculated the minimum energy savings which would be possible with voltage scaling; in this case the energy consumption of memory intensive applications could be reduced by at least 37% with a performance loss of less than 10%. The current implementation can only use a small number of counters that were intended originally for performance profiling. If the operating system technology is ready to deal with a variety of counters it is just a small step to embed new counters which are exclusively devoted to energy profiling.

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

8. REFERENCES

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

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

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

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

[5] M. Fleischmann. Longrun power management. White Paper of Transmeta Corporation, January 2001.

[6] J. Flinn, G. Back, J. Anderson, K. Farkas, and D. Grunwald. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems SIGMETRICS'2000*, June 2000.

[7] J. Flinn, K. Farkas, and J. Anderson. Power and energy characterization of the itsy pocket computer. Technical Report TN-56, COMPAQ Western Research Lab, February 2000.

[8] D. Grunwald. Boulder unified frequency/voltage scaling interface (bufscale). http://systems.cs.colorado.edu/EnergyEfficientComputing/cdcvsi.html, 2001.

[9] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture HPCA'02*, February 2002.

[10] I. Hong, M. Potkonjak, and M. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proceedings of the International Conference on Computer-Aided Design ICCAD'98*, November 1998.

[11] Intel. *Intel 80200 Processor based on Intel XScale Microarchitecture Developer's Manual*, November 2000.

[12] Intel. *Intel XScale Microarchitecture Technical Summary*, July 2000.

[13] Intel. *Intel 80200 Processor based on Intel XScale Microarchitecture Datasheet*, September 2001.

[14] Intel. *Intel IQ80310 Evaluation Platform*, July 2001.

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

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

[17] Micron Technology. Calculating memory system power for DDR. Technical Report TN-46-03, 2001.

[18] T. Pering and R. Broderson. Energy efficient voltage scheduling for real-time operating systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session*, June 1998.

[19] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th Symposium on Operating Systems Principles SOSP'2001*, October 2001.

[20] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the International Symposium on Mobile Multimedia Systems & Applications MMSA'2000*, November 2000.