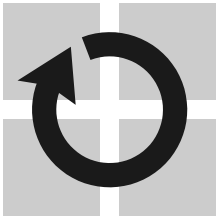
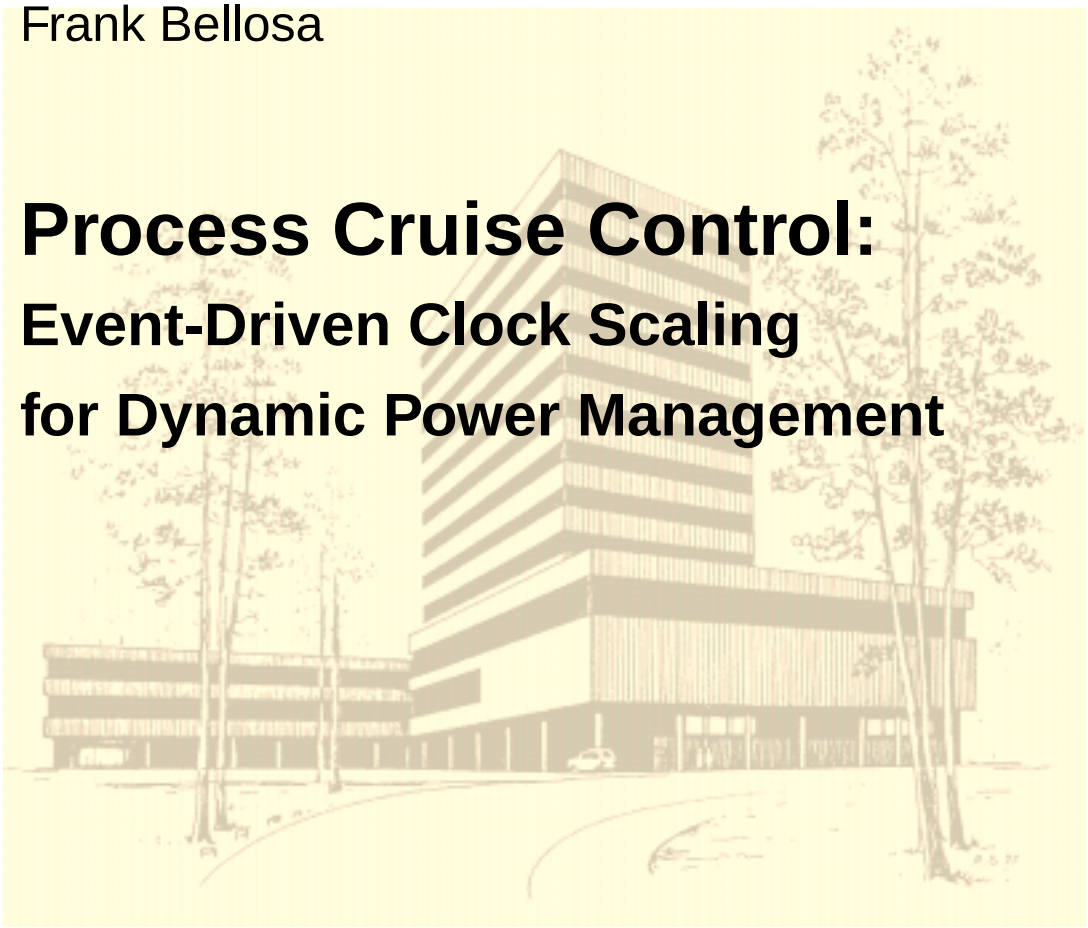


Department of Computer Science 4
Distributed Systems and Operating Systems

Frank Bellosa

Process Cruise Control:
Event-Driven Clock Scaling for Dynamic Power Management



Technical Report TR-I4-01-11

Friedrich-Alexander-Universität
Erlangen-Nürnberg
TECHNISCHE FAKULTÄT

Process Cruise Control:
Event-Driven Clock Scaling for Dynamic Power Management

Frank Bellosa

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

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 characterize the power-specific characteristics of a thread.

In this paper we propose an energy-aware scheduling policy 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.

Keywords: Power Management, Scheduling, Clock Scaling, Event Counter

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 affecting data centers and server-farms in many parts of the United States we are suddenly facing a rising awareness for the topic of energy management.

This paper contributes to this awareness and initiates a new approach in system software: the on-line evaluation of counters that register performance- and energy-critical events. 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 HW-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 HW-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 choose the Intel XScale architecture as our target platform. Other architecture 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 benefit of our approach. Finally in section 5, we propose further architectural innovations that Process Cruise Control could avail oneself of and then conclude in section 6.

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. As the basic activity of the CPU core proceeds in clock cycles, we expect some relation between energy consumption and clock frequency. However, a major part of the activity depends on the type of instructions and their operands. Therefore, we have to keep an eye on the activity of each functional unit.
- If a memory management unit (MMU) is used in a computer architecture for reasons of mapping and protection, the MMU will contribute significantly to the energy consumption as it 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.
  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.

Although architectural simulators [6] can simulate selected processor targets with a sufficient resolution in time and energy, they do not yet cover the whole system including off-chip communication and I/O. However a complete system is necessary to boot a full-featured operating system and to run real-world applications so we can demonstrate the benefit of OS directed power management.

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 [13, 12] processor used in this system can operate at clock speeds from 333 MHz to 733 Mhz. 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.

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.Thr 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.

The Intel 80312 I/O companion chip includes a PCI, SDRAM and flash memory controller. Serial and ethernet interfaces are provided to communicate with the device.

The system is running a modified version of BlueCat Linux from LynuxWorks.

2.2 Measurement Set-Up

The IQ803010 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. To measure the energy consumption, a sense resistor of low resistance (0.05 Ω) is placed in series with the power supply. A data acquisition system with a sampling frequency of up to 300 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 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.

First, we ran an arithmetic test (add_reg) doing arithmetic operations with all operands in register, then 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/memory).

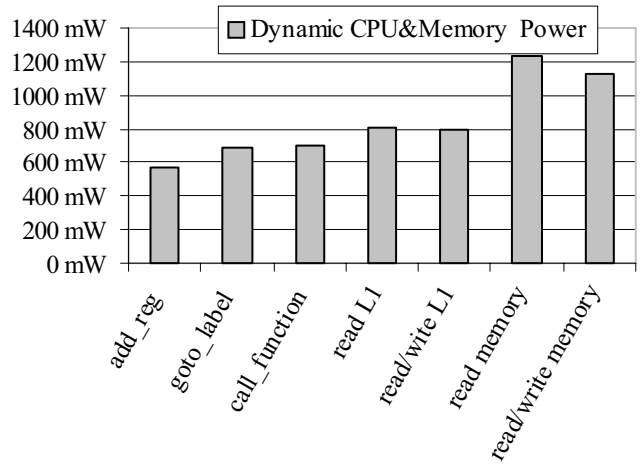


FIG. 1 : IQ80310 Board (XScale@733MHz, 32 MB SDRAM) Power Breakd

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.

Benchmark	Instructions per µs	Branches per µs	L1 References per µs	Memory Requests per µs
add_reg	680	12	0	0
goto_label	313	104	0	0
call_function	379	52	143	0
read L1	548	46	182	0
read/write L1	578	38	307	0
read memory	85	7	28	3.7
read/write 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 not to influence the 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.

2.4 Power/Performance Characteristics

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

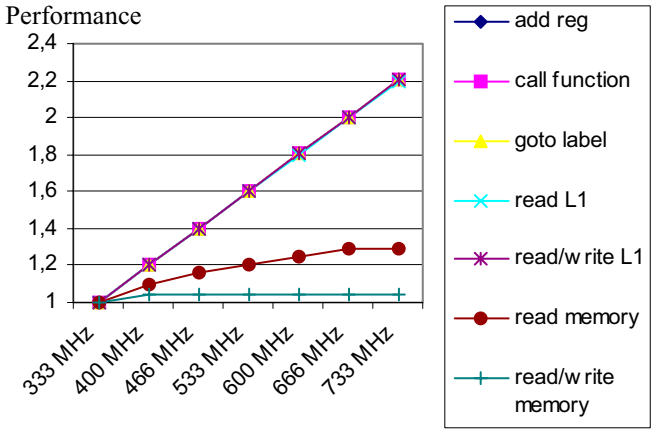


FIG. 2 : Relative Application performance at various clock speeds

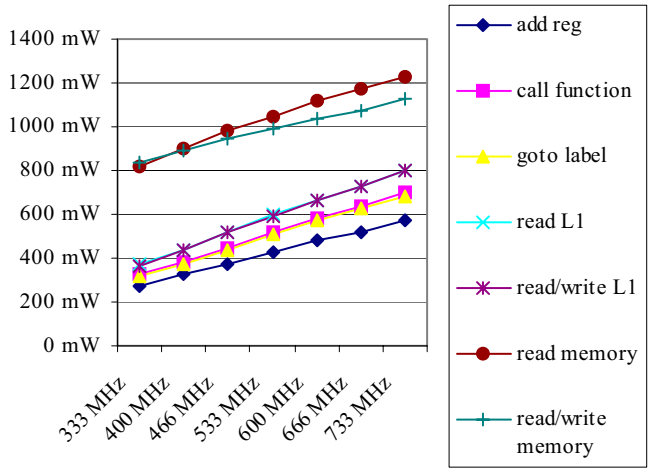


FIG. 3 : Energy consumption of the IQ80310 board (XScale 333-733MHz) at various clock speeds

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. Further-

more 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 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

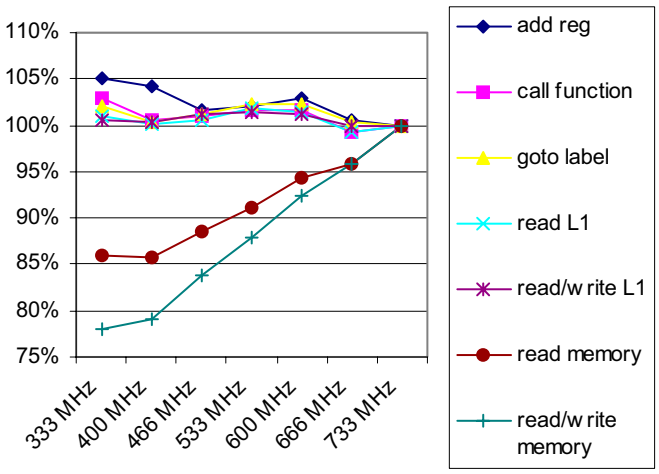


FIG. 4 : Relative Energy Performance of the IQ80310

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 to run the CPU and cache intensive applications at lower clock speed. Therefore we can run those applications at the highest speed.

The more memory requests are issued by an applications the more it pays 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 speed are two characteristics which are correlated, but the degree of correlation depends on the use of performance- and energy-critical HW-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 power/performance trade-off and select the right speed of execution.

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.

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.

A scheduler implementing P*rocess 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.

3.1 XScale Frequency Domains

The Intel XScale 80200 processor implements two event counters and one clock counter. 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 second 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.
- The instructions per second indicate the sensitivity for a performance loss due to speed reduction. The lower the rate of executed instructions the less a thread's performance will suffer from a reduction in clock speed.

To span the space of both event rates we constructed microbenchmarks producing various event rates. For each clock speed, we determined the event rates for each of these benchmarks. The dots represent the micro-benchmarks with their characteristic event rate at a fixed clock speed (666 MHz in figure 5).

The next step is to find the minimal clock speed which can be tolerated for given performance requirements. In figure 5 we choose 10% as an acceptable performance loss. The shading of the dots represents the minimum speed at which a certain thread will run with less than 10% performance loss.

The last step is to partition the two-dimensional space in fre-

quency domains. We choose a simple approach with lines that define the frequency domains. The lines separate micro-benchmark with different optimal clock speed. If the coordinates of a thread fall below a line, the thread can run without significant performance loss at the speed that corresponds to this line.

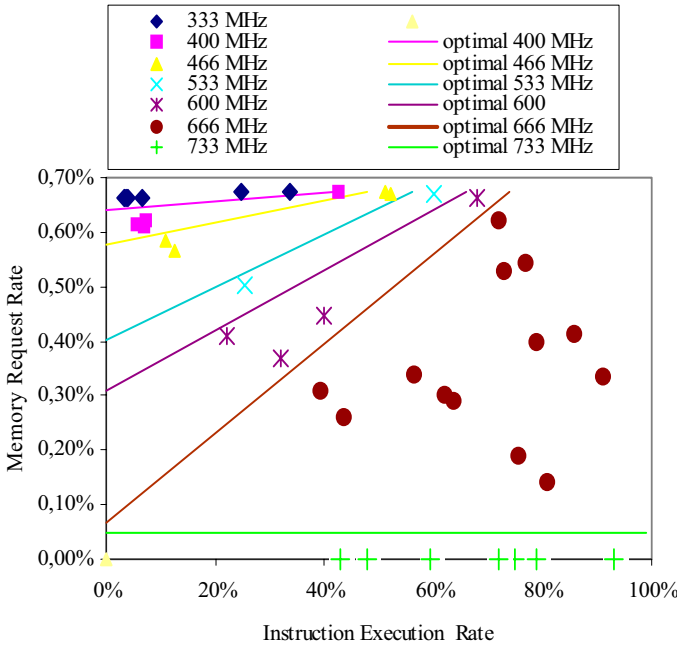


FIG. 5 : Frequency domains of a space spanned by instruction-execution and memory-request rate at 666 MHz.

4 Process Cruise Control in Linux

4.1 Implementation

The enhancement of LynuxWorks BlueCat Linux 2.2.12 to support event-driven frequency scaling comprises several modules:

- The module “perfctr” provides basic support for the configuration and reading of the event counters. The character device /dev/perfctr offers an interface for the configuration and monitoring of the counters from the user level.
- Context switch routines and kernel data structures are modified in module “virtual_perfctr” to hold the values of the two available performance-monitoring event counters. Within the timer interrupt processing thread-specific virtual event counters are updated and the event ratio for each event is determined. To ease the implementation we base on the Performance API (PAPI) from the University of Tennessee [18, 7].
- Basic speed adjustment is performed in module “scale” that offers a /proc/scale interface for the configuration of default speed settings.
- Each thread can be configured to run at a thread-specific fixed speed or to run with dynamic event-driven clock-scaling. Module “virtual_scale” extends the context switch routines for thread-specific speed tuning and offers the necessary user-interfaces (/proc/vscale).
- The determination of the clock frequency with respect to the values of the virtual event counters and to the thread-specific event rates is done in the module “policy”. To provide a higher flexibility, different policies are possible. A thread can be configured to use a certain policy for its speed decisions. Our initial policy module provides a /proc/event_scale interface to configure and reconfigure the frequency partitions of a running system.
- Finally “event_scale” is the wrapper around the other modules. Periodically the thread-specific data is gathered from “virtual_perfctr”, the clock speed is determined by “policy” and the clock frequency is adjusted by “virtual_scale” if this should be recommended.

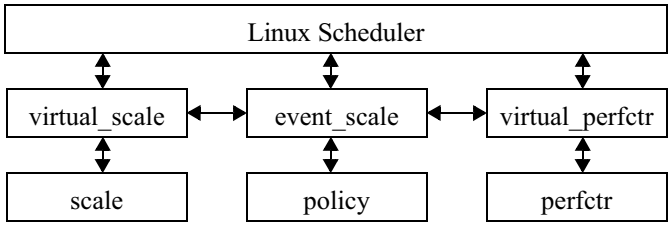


FIG. 6 : Module structure of Process Cruise Control in Linux.

With all these modules to be hooked into the Linux scheduler there arises the overhead question. Our initial implementation nearly doubles the overhead of the scheduler with a high variation due to a variable number of compulsory cache misses. While the pure scheduler is optimized and avoids function calls, we designed process cruise control for maximum modularity and easy re-configuration, even of the running system. A bunch of indirect function calls within the scheduler is not beneficial for the scheduler performance. Therefore there are still many options for tuning of the code, for re-arranging of run-time data and for inlining of functions.

Modules	Overhead in Cycles
Linux scheduler	> 26000
perfctr	184
event_scale &	6000-18000
scale	1000-2300 (= 3.1 µs)

4.2 Measurements

To show the benefit 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. Figure 7, 8, 9 and 10 show the power/performance characteristics of four 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.

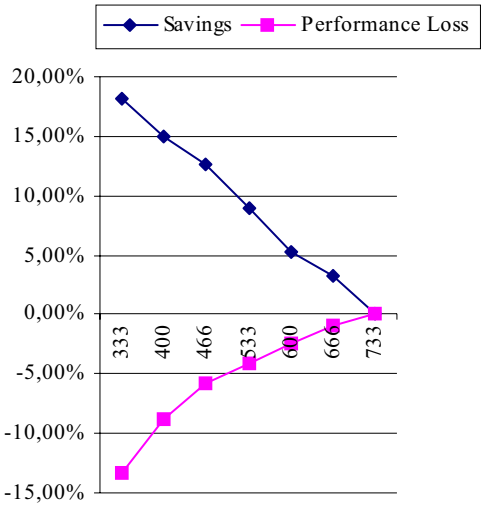


FIG. 7 : find & grep Power/performance Profile

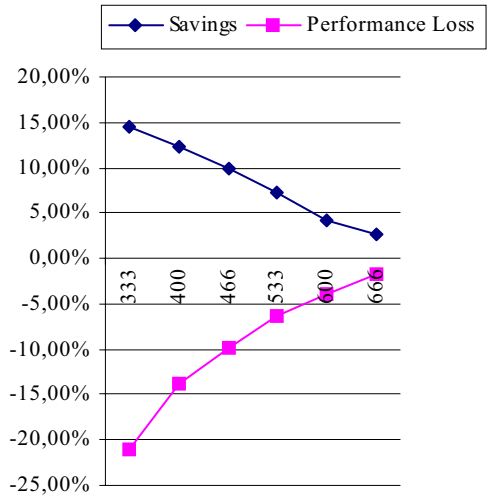


FIG. 8 : gzip power/performance profile

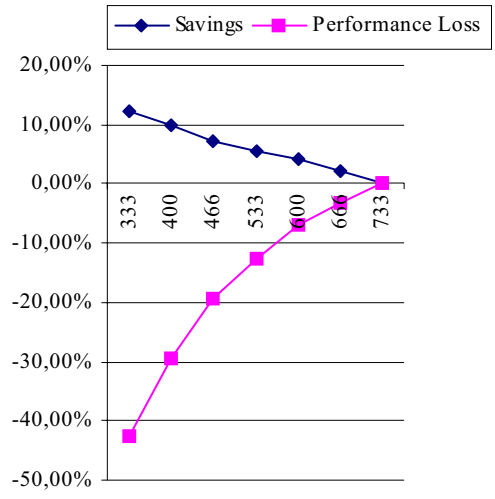


FIG. 9 : djpeg Power/performance Profile

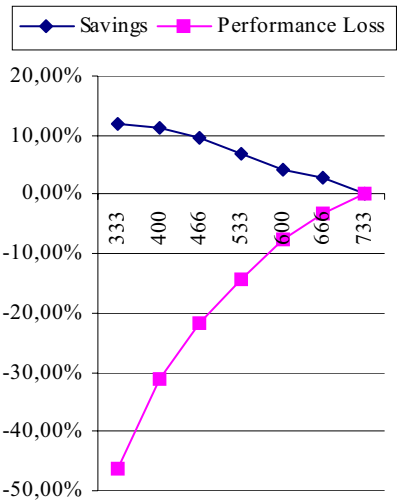


FIG. 10 :factor Power/performance Profile

Furthermore Process Cruise Control determined the rate of memory requests and executed instructions. The characterization according to the rates gathered at 666 MHz is plotted in figure 11. Here the speed according to Process Cruise Control is determined by the line above the marker of each application.

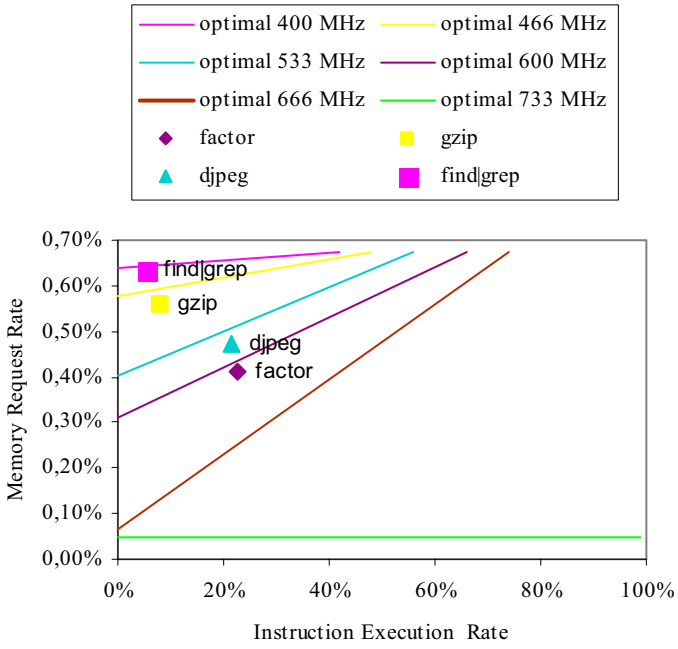


FIG. 11 :Frequency domains of a space spanned by instruction-execution and memory-request rate at 666 MHz.

Table 2 summarizes the results. For three tests ("find|grep", "gzip" and "factor") Process Cruise Control comes to the same clock speed as the external measurements.

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 2:Process Cruise Control clock speed and energy savings

One application "djpeg" 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%.

However 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.

5 Energy Monitoring Counters

The performance monitoring counter that are 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. Both types 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 power/performance characterizations of threads, they cannot be used for application profiling. Therefore an energy-aware system should implements 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. The innovative approach is the use of event-counters to enable serious power/performance trade-offs 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 in a scheduler for the purpose of scheduling was reported in [5]. 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[2].
Recent work employs event counters for run-time power estimations and energy accounting and throttling [3, 16, 4].

There is a rising number of papers which describe the power measurement of systems which support dynamic clock scaling sometimes in combination with dynamic voltage scaling [8, 9, 15, 21].

Several papers cover the topic of scheduling on systems with variable speed. Most of them focus on real-time scheduling [19, 11, 20]. 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 and /proc/vscale interface, the BUFScale API [10] offers an architecture independent interface for power system tools and applications.

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 his patterns.
Our approach does not impose any hints on application. *Process Cruise Control* minimizes the energy consumption while 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. 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.

References

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

[2] BELLOSA, F. Process cruise control: Throttling memory access in a soft real-time environement. Tech. Rep. TR-I4-97-2, University of Erlangen, Department of Computer Science, Jul 1997.

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

[4] BELLOSA, F. The case for event-driven energy accounting. Tech. Rep. TR-I4-01-07, University of Erlangen, Department of Computer Science, June 2001.

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

[6] BROOKS, D., TIWARI, V., AND MARTONOSI, M. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings Of The 27th Annual International Symposium on Computer Architecture ISCA-27* (June 2000).

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

[8] FLINN, J., BACK, G., ANDERSON, J., FARKAS, K., AND GRUNWALD, D. 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).

[9] FLINN, J., FARKAS, K., ANDERSON, J., AND FARKAS, K. Power and energy characterization of the itsy pocket computer. Tech. Rep. TN-56, COMPAQ Western Research Lab, Feb 2000.

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

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

[12] INTEL. *Intel 80200 Processor based on Intel XScale Microarchitecture Developer's Manual*, Nov 2000.

[13] INTEL. *Intel XScale Microarchitecture Technical Summary*, Jul 2000.

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

[15] JOSEPH, R., BROOKS, D., AND MARTONOSI, M. 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] JOSEPH, R., AND MARTONOSI, M. 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. Tech. Rep. TN-46-03, 2001.

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

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

[20] PILLAI, P., AND SHIN, K. 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).

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