OS-Directed Throttling of Processor Activity for Dynamic Power Management

Frank Bellosa

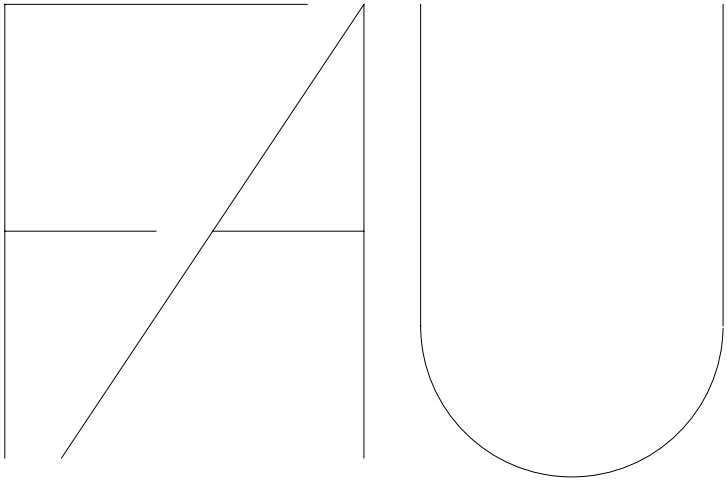
June 1999 TR-I4-3-99

Technical Report

Computer
Science Department

Operating Systems – IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany



EndurIX

OS-Directed Throttling of Processor Activity for Dynamic Power Management

FRANK BELLOSA

Computer Science Department (Operating Systems), University of Erlangen, Germany

Today, embedding fast processors in portable devices is infeasible because such battery operated systems cannot be supplied with enough power and cannot be kept cool for a reasonable period of time.
Many processors offer the feature of reducible clock speed to save power. Reducing clock speed improves the performance-to-energy ratio for three reasons: First, the number of stall cycles that the CPU has to wait for main memory is reduced, because the clock-cycle time of the processor core gets closer to the memory latency. Second, by reducing the clock speed one can draw more energy out of batteries due to the impact of the lower discharge rate on the capacity of a battery. Finally, a reduced clock speed offers the possibility for further energy savings, because the supply voltage can be reduced as well.
Our approach to OS-directed power management adds the clock speed to the runtime context of a thread. In addition to the questions when a thread has to be executed and which CPU should be used, we enlarge the freedom of scheduling to a third dimension: the speed of execution. By tuning the clock speed, the operating system can adjust the quality of service to the power constraints of a device.

1 Introduction

Battery operated systems with sporadic yet extremely high demands in computing power (e.g., hand-writing or speech recognition) call for fast processors. To lower the energy consumption, today's operating systems stop the processor in the idle thread. But frequent context switches to the idle thread imply additional computing overhead and energy consumption. Therefore halting and restarting the processor does not pay for short periods of idle time. (You don't repeatedly open and close the faucet to regulate the throughput of water in the shower.) Furthermore, there are background activities performing uncritical optimization tasks (e.g., page recoloring in the operating system, motion optimization in the area of robotics) preventing the system from idling.

Some of today's embedded processors (AMD ELAN SC400 [Amd97], PowerPC860 [Mot98], StrongARM1100 [Int99b], Hitachi SH4 [Hit98]) offer the feature of reducible clock speed to save power. Reducing clock speed improves the energy performance for three reasons: Reduced stall cycles, improved battery capacity and power savings due to voltage reduction.

1.1 Saving memory stall cycles

The die of high performance processors is dedicated mostly to the caches, e.g., the 1.5 MB first level cache of the PA-RISC 8500 processor makes up the bigger part of its 140 million transistors. To improve the die yield and to cut down the costs per unit the number of gates in an embedded processor should be minimal. Without sacrificing functionality, costs can be cut by leaving out large caches. Consequently most embedded controllers have only small first level caches of 1 KB – 16 KB. The cache miss rate of those systems is in the range of 2% – 10% [HP96]. Because of those high miss rates, the memory latency has a significant impact on the system performance.

Assume the cache miss penalty is 30 clock cycles, all instructions take 2 clock cycles (ignoring memory stalls). Assuming the miss rate is 5%, and there is an average of 1.33 memory references per instructions. With this scenario the average clocks per instruction are

$$CPI_{total} = CPI_{execution} + MemRefRate \times MissRate \times MissPenalty$$

$$CPI_{total} = 2.0 + 1.33 \times 0.05 \times 30 = 3.995$$

Reducing the clock frequency lowers the number of cycles the CPU has to wait for main memory. In our example a reduction of the clock speed to a third marks down the miss penalty to 10 cycles.

$$CPI_{total} = 2.0 + 1.33 \times 0.05 \times 10 = 2.665$$

Given a fixed supply voltage, the energy a processors needs to solve a task is nearly proportional to the number of clock cycles. In our example, the number of clock cycles and with it the energy spent to solve a task could be cut down to 67% by reducing the clock speed to 33%.

This effect becomes very important for highly clocked controllers with small caches. Reducing the clock speed of those processors improves the performance relative to the clock speed and therefore improves the energy performance while preserving the computing power for those situations when power consumption does not matter.

1.2 Exploiting battery capacity

The current I drawn by a processor has two components, the static current S that is independent of the CPU frequency F and the dynamic component that is proportional to the frequency. Reducing the clock speed down to 20%-90% reduces the current $I = S + D \times F$ within a range, that is still dominated by the dynamic component. The static component has to be considered only if the device should be suspended for a long period of time.

The battery capacity is the total amount of power a battery can deliver when discharged at a constant current, called a 1C discharge rate, over a a defined period (e.g., 5 hours) [MaSi96]. The discharge rate has a non-linear impact on the total amount of power a battery can deliver. A typical lithium-ion battery has the following characteristics [Int98]:

Discharge Rate	Usable Battery Capacity (Normalized to 1C Discharge Rate)
C/5	107%
C/2	104%
1C	100%
2C	94%
4C	86%

Using a processor in a high-speed mode producing a high discharge rate (e.g., >4C) can lower the usable battery capacity to 70%-80%. In battery powered systems with sporadic yet extremely high demands in computing power (e.g., hand-writing or speech recognition) dynamic clock-speed setting helps to draw the best usable power out of batteries while providing high computing power for short bursts.

1.3 Reducing power consumption

Since dynamic power consumption is proportional to the square of the supply voltage, lowering the voltage provides a disproportionate boost to battery life [WWD94]. First processors with clock speed that is adjustable via software are rated for multiple voltages and clock rates, e.g., the voltage of the AMD ELAN SC400 can be lowered from 3.3 Volts to 2.7 Volts if the clock rate is lowered from 100 MHz to 33 MHz. The power drops from 1818 mW (100 MHz/3.3V) down to 469 mW (33MHz/2.7V) [Amd97]. This reduction of speed and voltage cuts down the energy per clock cycle to 77% of the amount that is necessary in the high speed mode. Expanding the range of supply voltage to the technological limits would create the opportunity for quadratic savings.
However it takes several microseconds to adjust the supply voltage. Therefore boosting the voltage before increasing the clock frequency has to be considered in the medium-term scheduling. A short-term reduction of the clock frequency is possible, whereas a short-term increase is prohibited if the current supply voltage is not sufficient.

2 Approach of This Paper

This paper discusses the fine-grain **and** thread-specific control of CPU clock speed to save power while considering real-time requirements. Because the expected power savings base on very subtle effects (saving memory cycles, battery discharge analysis, speed-voltage correlation) a trace-driven simulation is unfeasible. Therefore we describe our approach in a way, that a true evaluation with real hardware can follow.

Section 3 motivates the thread-specific speed control and sketches mechanisms to tune the processor's clock speed. Policies for OS-directed throttling of clock speed are presented in Section 4. Finally we outline future work in Section 5, including additional benefits from thread-specific speed control in multiprocessors. Finally, Section 6 provides our conclusions.

3 Adding the Clock-Speed to the Context

The operating system has to tune the clock speed according to the power constraints of the computer and the quality of service demands of the applications.

The real-time properties of a task related to computing concerns (we intend not to cover the field of I/O throughout this paper) depend on two factors:

- the latency to start-up a task after the upcoming of an event
- the time it takes to complete the task

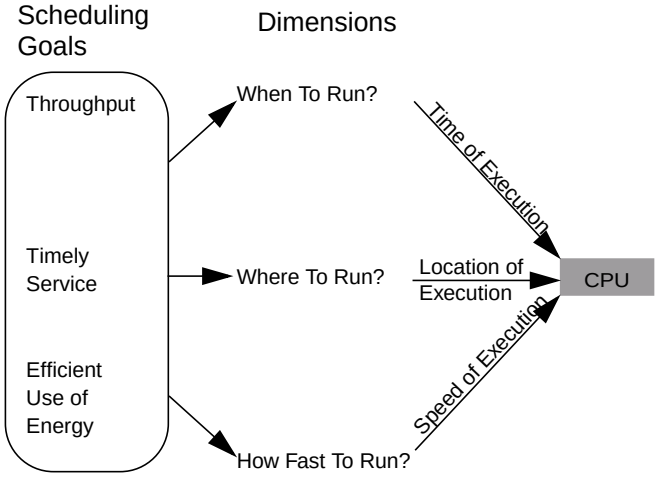
Assuming a preemptive system, the first factor depends on the priority assigned to a task and the time span to perform a context switch. The second factor depends on the number of clock cycles dedicated to a task within a time span.

Tuning the clock speed presumes an entity to which a specific clock speed must be assigned and a policy to determine the appropriate speed settings. Previous work in the field of dynamic speed settings [WWD94] [GCW95] adjust the clock speed in intervals. The speed is determined by looking a fixed window into the past. The cycles spent in the idle loop determine the speed settings for all threads running in the next time window. Global speed adjustment for all threads in the system implies some drawbacks:

- Real-time requirements demand for deterministic execution times of critical tasks, e.g., threads blocking valuable resources should proceed as fast as possible. The execution time for a specific thread cannot be predicted with a variable clock speed defined on a global level.
- Background activities that perform optimization tasks prevent the system from idling. Frequently, optimizing is a pro-active task, that runs steadily with low priority in the background. Because the optimal result will never be reached on a given target-hardware within a span (e.g., self-learning hand-writing recognition, motion optimization in the area of robotics), the suboptimal result will be used at a point in time, when a real-time or interactive task needs a result. With thread-global settings derived from idle cycles, those background tasks prevent any effort for power savings.

To overcome the limitations of a global clock speed, we propose to enlarge the context of a thread. Beside the question when a thread has to be executed and which CPU should be used (in the case of a multiprocessor), we enlarge the freedom of scheduling to a third dimension, the speed of execution. We

allow a *three-dimensional division of the computing resources* among competing threads: in time (*when?*), in space (*where?*) and in velocity (*how fast?*).



Within the space of decision, the scheduler has to control the execution in a way that scheduling goals like throughput, timely service and efficient use of energy are achieved (see Section 4).

Additional to the traditional context of a thread (address space, registers, I/O descriptors) the speed value is saved/restored in the switching routine. Without any significant overhead, speed setting is done in many processors by saving a speed value in one of the control registers to program the internal PLL circuit that generates the clock [Amd97][Mot98].

To emulate a divided processor clock frequency on processors without a programmable PLL, the Advanced Configuration and Power Interface Specification (ACPI) [Int99a] recommends a programmable clock-logic that periodically stops the processor's clock by stop grand cycles. With a given clock-on-time and a clock-off-time, the clock logic can throttle the clock cycles so that the effective frequency is reduced. The latency to tune the clock speed by ACPI should be the same as the latency to write into a register of an external memory-mapped device. Although this value is higher than the time for reprogramming the CPU-internal PLL, it should be possible to tune the clock speed within the time of a context switch.

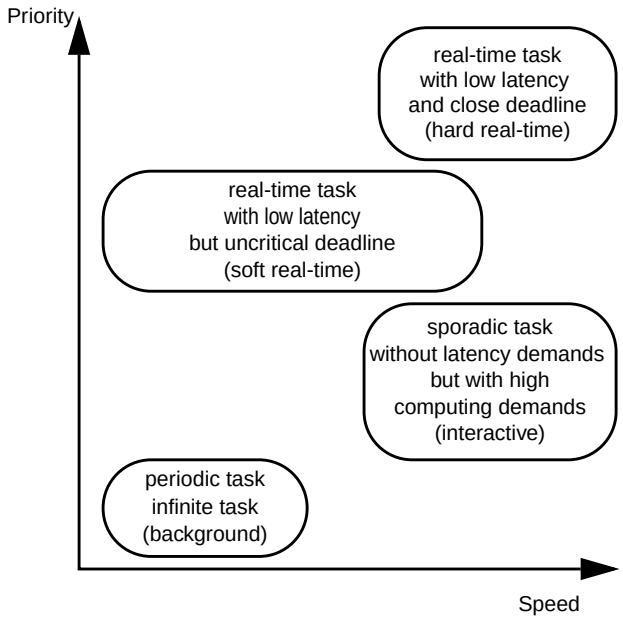
To guarantee a proper operation of the processor the voltage has to exceed a minimum level imposed by the clock frequency and the settling time for the gates in the chip. Reducing the clock speed is possible immediately, voltage reduction can follow with a delay. When raising the clock speed the voltage has to be boosted first. We propose to trigger the speed boost by a high precision timer. While the voltage can be configured at the context switch, the speed adjustment has to be delayed until the voltage has reached a speed-compliant voltage level. With hard real-time threads imposing narrow time-constraints, voltage reduction is prohibited.

4 OS-Directed Throttling

Our approach to OS-directed throttling of processor activity separates the aspect of execution speed from the aspects of execution order and execution location. From the separation of execution speed and execution order evolve four scheduling classes. Theses classes build partitions within the space of decision. To simplify the aspect of execution order, we assume that the order in time can dynamically be transformed to an execution priority. Firstly we focus on monoprocessor architectures. The field of multiprocessing is marked in Section 5.

4.1 Scheduling Classes

From the orthogonality of speed and priority we derive four scheduling classes:



- A task that has predefined service-rate objectives with a low start-up latency and deterministic execution time should be assigned to the hard real-time class. Because the scheduler operates on the highest priority-range, the latency can be kept low. Those time-critical tasks guarantee a proper operation of the system. Therefore the dynamic throttling of their speed is no question. If a task with close limits cannot tolerate the delay of speed-adjustment implied by voltage-reduction, the voltage must be frozen on the highest level.
- If missing a dead-line does not cause severe consequences, the speed of real-time threads can be tuned, so that they meet their service-rate objectives with a high probability. The start-up latency of soft real-time tasks is low, because their priority is directly below the range that is dedicated to hard real-time tasks. If power-consumption is an important concern, the quality of service of those tasks can be reduced without compromising system stability.
- The rapid finishing of a task might have a high importance for an interactive user. While latency is associated with priority, importance is more related to speed. To obtain a re-

sult quite fast, the processor can run with high speed as long as it is justifiable from the view of power management. Many important tasks have a low frequency (e.g.,an update of cross-references in documents, compilation of sources). Therefore the speed boost is mostly short-term so that the higher energy consumption can be accepted in many cases.

- Background activities should operate at a speed level that is low enough to save a significant amount of energy.
  Frequently, background threads fulfill optimization tasks. Running those tasks with full speed prevents the system from idling and drains the battery. Stopping those tasks abandons any optimization.
  An other example of background activity are periodic updates (automatic saving of an editor, scanning of external servers for incoming mail/news, roaming in a wireless network, recoloring of physical pages in the operating system).
  We recommend to assign energy budgets (e.g., cycles/second) to those background tasks. The background-scheduler cares for the assignment of the budgets at a minimal clock speed. This prevents the system from idling while keeping the average clock speed as low as possible.

4.2 Energy Accounting

Significant data related to the subject of decision should be a prerequisite of any judgement. Because of the lack of energy accounting, serious power management is not possible with current operating systems, in which adjusting power-management parameters to achieve a long system-lifetime is at best a black art.
To determine if a thread is a candidate for speed throttling, we have to account its energy use. Monitoring hardware is already embedded in many processors to count events like clock cycles, stall cycles, and cache misses. A first approach to energy accounting is the use of a cycle counter. The number of clock cycles that a thread has used is multiplied by an energy factor which depends on the clock frequency. The characteristic curve of the energy factor is derived from power measurements carried out on the target system[BBC98].

4.3 Speed-setting Policies

The objective of scheduling in general is to provide timely services. This includes the option to complete the service in some point in the future.Therefore the system has to control the energy consumption of a battery powered device so that the system is still operational at this point in time.

It is important to know how long the system has to be kept operational under the current workload. According to the user's preferences the characteristics of the battery, and the power characteristics of other components in the system (e.g., backlight, disk, chip set, etc.), the average power consumption which can be granted to the processor is determined.

Keeping the processor busy at low speed is the key to dynamic speed throttling. The threads of both the real-time classes and the background tasks form the base-load of the system. The speed values assigned to those threads should be specified in such a way that context switches to the idle thread become a rare event.

Periodically the speed-adjustment software caries out the following steps with respect to the power characteristics of individual threads:

(1) The speed of the hard real-time threads should be fixed at a conservative level (e.g. twice the recommended speed) to avoid any deadline misses.

(2) The speed of the soft real-time threads should be tuned in a way that they meet their deadlines with a high reliability.
  If the power consumption of hard and soft real-time threads exceeds the limit imposed by the user preferences, the user should be asked to authorize a cutback in the quality of service of the power-intensive soft real-time tasks.

(3) The remaining power is distributed among the background tasks.

(4) Because of the conservative rating in step (1) and (2), we should have enough energy for sporadic interactive bursts. If the energy budgets are not sufficient in the long-term, we further throttle the background tasks down to a minimum level (e.g., 10% of the maximum speed). If this is not sufficient, even the interactive threads are throttled according to their energy consumption.

The speed adjustment of individual threads is a concern of medium-term scheduling.

5 Future Work

The locality of execution has not been covered in this paper, because multiprocessors play no part in the field of low power devices. From the point of view of power consumption, a single chip multiprocessor [HNO97] is quite attractive compared to a high-end single processor requiring the same number of gates, because some of the processors can be powered down in periods of low load to save energy. It is easier and more efficient to power up a processor unit than to preempt a running thread if a request has to be processed. Depending on the power characteristics of a processor unit, the real-time demands of the work load, and the demands of the interactive user, we have to find the optimal operation point of each task in the system concerning execution time, execution location, and execution speed.

The proposed heuristic speed-setting policies can certainly be refined in the future. However, a realistic evaluation of novel policies is only possible with an adapted operating system running on real hardware, because the expected power savings

base on very subtle effects (saving memory cycles, battery discharge analysis, speed-voltage correlation). Therefore we plan an implementation of speed-setting policies in RT-Linux [Yod99] running on the AMD ELAN processor and in the JAVA based RTOS JBED [Jbed99] running on the PowerPC860 processor.

6 Conclusions

OS-directed throttling of processor activity improves the endurance of low-power devices by throttling clock rate and core voltage, while providing quality of service (QoS) in a real-time environment. We expect thread-specific speed settings to become an unrenounceable element of future operating systems for power-sensitive devices.

7 References

[Amd97] AMD, *ElanSC400 and ElanSC410 Microcontroller User's Manual*, 1997

[BBC98] L. Benini, A. Bogliolo, S. Cavallucci, Brunoa Ricco, "Monitoring System Activity for OS-directed Dynamic Power Managament", *Proc. of int. Symposium on Low Power Electronics and Desing ISPLED'9*8, 1998

[Hit98] Hitachi, *SuperH (SH) 32-Bit RISC MCU/MPU Series SH7750 Hardware Manual*, July1998

[HP96] J. Hennessy and D. Patterson, *Computer Architecture, a Quantitative Approach*, San Francisco, Morgan Kaufmann, 1996.

[GCW95] K. Govil, E. Chan, H. Wassermann, *Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU*, Technical Report TR-95-017, ICSI Berceley, April 1995

[HNO97] L. Hammond, B. Nayfeh and K. Olukotun, "A Single-Chip Multiprocessor", *IEEE Computer Special Issue on "Billion-Transistor Processors"*, September 1997

[Int98] Intel, *Mobile Power Guidelines 2000 Rev 1.0*, Dec. 1998

[Int99a] Intel, Microsoft, Toshiba, *Advanced Configuration and Power Interface Specification 1.0b*, Feb. 1999

[Int99b] Intel, *Intel StrongARM SA-1100 Microprocessor Developer's Manual*, April 1999

[Jbed99] Esmertec, *JBed RTOS Manual*, Zürich, May 1999

[MaSi96] T. Martin, D. Siewiorek, "A Power Metric for Mobile Systems", *Proc. of int. Symposium on Low Power Electronics and Design ISPLED'96*, 1996

[Mot98] Motorola, *MPC860 PowerQUICC User's Manual Rev. 1*, September 1998

[WWD94] M. Weiser, B. Welch, A. Demers, S. Shenker, "Scheduling for Reduced CPU Energy", *Proc. of the First Symposium on Operating System Design and Implementation OSDI'94*, November 1994

[Yod99] V. Yodaiken, *The RTLinux Manifesto*, Dept. of Computer Science, New Mexico Institute of Technology, April 1999