Dynamic Voltage Scaling on a Low-Power Microprocessor

Johan Pouwelse * Koen Langendoen Henk Sips
Faculty of Information Technology and Systems
Delft University of Technology, The Netherlands
{pouwelse,koen,sips}@ubicom.tudelft.nl

ABSTRACT
Power consumption is the limiting factor for the functionality of future wearable devices. Since interactive applications like wireless information access generate bursts of activities, it is important to match the performance of the wearable device accordingly. This paper describes a system with a microprocessor whose speed can be varied (frequency scaling) as well as its supply voltage. Voltage scaling is important for reducing power consumption to very low values when operating at low speeds. Measurements show that the energy per instruction at minimal speed is 1/5 of the energy required at full speed. The frequency and voltage can be scaled dynamically from user space in only 140 μs. This allows power-aware applications to quickly adjust the performance level of the processor whenever the workload changes. Experiments with an H.263 video benchmark show that the power-aware decoder outperforms a static fixed-frequency policy as well as a dynamic interval-based scheduler.

1. INTRODUCTION
Today's bulky portable computers will be replaced by small wearable devices in the near future. Wearable devices introduce several challenges that go beyond those of the traditional computing model. The goal for the next-generation wearable devices is to extend the services beyond mere voice, address book, e-mail, and limited computation [1]. As an example, the Ubiquitous Communications (UbiCom) project at Delft University of Technology aims at developing a wearable device equipped with a long-range 10 Mbps wireless link and an augmented-reality display [10].

Power consumption is becoming the *limiting factor* for the functionality of wearable devices, because advances in battery technology are progressing slowly whereas computation and communication demands are increasing rapidly. It is important to utilize the available energy as efficient as possible. Energy preservation, or energy management, is further translated into a low power consumption of all parts of a wearable device. The initial response to the low-power demand was to lower the supply voltage. For example, reducing the supply voltage from standard 5.0 V to 3.3 V reduces power by 56%.

Lowering the supply voltage requires all components to operate at low voltage. Additional reductions can be obtained by selectively lowering the supply voltage of specific parts. An obvious candidate is the processor since it is responsible for 10 to 30% of the power consumption [12].

The dynamic approach to low-power is using power down features to minimize the power consumption of unused hardware. For portable computers this means turning off the hard disk, processor, screen, modem, sound, etc. Re-activation of hardware can take some time, which affects performance (e.g., response time). Using simple power-down-when-idle techniques the processor's power consumption can be significantly reduced. Depending on the usage pattern, the power savings can amount to a 66% reduction [13]. A refinement is to make continuous trade-offs between performance and cost. The user demand (performance) must be supplied at the lowest cost (power consumption). Performance can be expressed as the response time for interactive applications, and as spatial/temporal resolution, color depth, and distortion level for video. *Voltage scaling* is a method to trade-off processor speed against power consumption. The power consumption of a processor running at high speed and high voltage is much larger than running at low speed and low voltage.

This paper studies the trade-off between power consumption and performance of processors supporting dynamic voltage scaling. Unlike previous studies in voltage scaling that rely on simulations we have realized an actual implementation being part of a wearable computer. This allows us to present measurements of the performance/power trade-off in voltage scaling. From these numbers we derive the potential gain in a system where applications assist the OS in adjusting the voltage by specifying their (bursty) requirements.

2. VOLTAGE SCALING
This section introduces the basic principles of power consumption and the effects of voltage scaling. For digital CMOS circuits, used in the majority of microprocessors, the power consumption can be modeled accurately with simple equations [3, 9]. CMOS circuits have both dynamic and static power consumption. Static power consumption is caused by bias and leakage currents but is insignificant in most designs that consume more than 1 mW.

*Supported by the Dutch Organization for Applied Scientific Research (TNO), Physics and Electronics Laboratory.

The dominant power consumption for CMOS microprocessors is the dynamic component. Every transition of a digital circuit consumes power, because every charge and subsequent discharge of the digital circuit's capacitance drains power. The dynamic power consumption is equal to

$$P_{dynamic} = \sum_{k=1}^{M} C_k \cdot f_k \cdot V_{DD}^2 \quad (1)$$

where M is the number of gates in the circuit, C_k the load capacitance of gate g_k, f_k the switching frequency of g_k, and V_{DD} the supply voltage. It is clear from Equation (1) that reduction of V_{DD} is the most effective mean to lower the power consumption. Lowering V_{DD}, however, creates the problem of increased circuit delay. An estimation of circuit delay is given by

$$\tau \propto \frac{V_{DD}}{(V_G - V_T)^2} \quad (2)$$

where τ is the propagation delay of the CMOS transistor, V_T the threshold voltage, and V_G the input gate voltage [9]. The propagation delay restricts the clock frequency in a microprocessor. From Equations (1) and (2) it follows that there is a fundamental trade-off between switching speed and supply voltage. Processors can operate at a lower supply voltage, but only if the clock frequency is reduced to tolerate the increased propagation delay. When we assume the dynamic power is the most dominant one, and the gates g_k of the microprocessor form a collective switching capacitance C with a common switching frequency f, we obtain

$$P = C \cdot f \cdot V_{DD}^2 \quad (3)$$

Equation (3) shows that clock frequency reduction linearly decreases power and voltage reduction results in a quadratic power reduction. The critical path of a processor is the longest path a signal can travel. The implicit constraint is that the propagation delay of the critical path τ must be smaller than $\frac{1}{f}$. In fact, the processor ceases to function when V_{DD} is lowered and the propagation delay becomes too large to satisfy internal timings at frequency f.

To put the above formulas in perspective Table 1 gives the relation between frequency, voltage and power consumption for the Transmeta TM5400 or 'Crusoe' processor [22]. Transmeta refers to their voltage scaling technology as LongRun, AMD calls it PowerNow, and Intel uses the term SpeedStep. The Crusoe specifications give insights in the relation between frequency and voltage. At the lowest frequency (200 MHz) the Crusoe processor operates at 29% of the maximum speed for less than 13% of the maximum power. Without voltage scaling the power would be reduced to only 29%. Therefore, voltage scaling effectively more than halves $\frac{1.1^2}{1.65^2}$ the energy per processor instruction.

Unfortunately energy savings with voltage scaling can be limited due to the influence on system performance. Consequently, the benefits of voltage scaling are to be weighted against the power consumed by the remainder of the system. This issue will be addressed in Section 4.2. Second, application performance is a combination of processor speed and memory access latency. The performance of the memory subsystem is not linearly related to clock frequency, so application performance does not scale linearly as will be shown in Section 4. Third, certain applications like video players have real-time deadlines to meet. This limits the possibilities to apply voltage scaling by reducing the clock frequency, because the time needed to complete a task increases proportionally. The applicability of voltage scaling will be discussed in Section 5.

Frequency f (MHz)	Voltage V_{DD} (V)	Relative power (%)
700	1.65	100
600	1.60	80.59
500	1.50	59.03
400	1.40	41.14
300	1.25	24.60
200	1.10	12.70

Table 1: Clock frequency versus supply voltage for the Transmeta Crusoe processor.

3. IMPLEMENTATION

Voltage scaling has been primarily studied through simulation, see the related work discussed in Section 7. The simulation results are promising, so time has come to validate the technique. Within UbiCom we assembled a wearable computer that supports voltage scaling to experiment with application-directed voltage-scaling policies (Section 5). Therefore, we needed a processor capable of voltage scaling, but very few actually exist; the ARM8 implementation by Berkeley [16] is not readily available, and the Crusoe processor [22] has only been introduced recently. This is remarkable since Weiser et al. already in 1994 showed the potentials of voltage scaling for general purpose processors [23]. We selected the embedded StrongARM 1100 processor [8] that does support frequency scaling, but only operates at 1.5 V according to the data sheet. Operating outside the specifications is a risk, but experiments show that a range from 0.8 V to 2.0 V is feasible.

3.1 Experimental platform

The embedded StrongARM processor board displayed in Figure 1 forms the heart of the wearable augmented-reality terminal that we are developing within UbiCom [19]. The board, dubbed LART, has a size of 10x7.5 cm, a weight of 50 gr, 32 MB of volatile memory, 4 MB of non-volatile memory, a SA-1100 190 MHz processor, and various I/O capabilities. The LART has a programmable voltage regulator to control the voltage of the processor core.

The LART runs under control of the Linux operating system (Version 2.4.0), which has been enhanced to support frequency and voltage scaling. We added a kernel module that changes the clock frequency and subsequently recalibrates the kernel's internal delay routines, in particular those that busy-wait by counting instruction cycles. In addition, the kernel module adjusts the memory parameters that control the read/write cycles on the external bus. The code has been structured such that it may be interrupted and does not depend on external memory, which is temporarily unavailable during a clock frequency change. All LART design schematics and kernel modules are openly available [2].

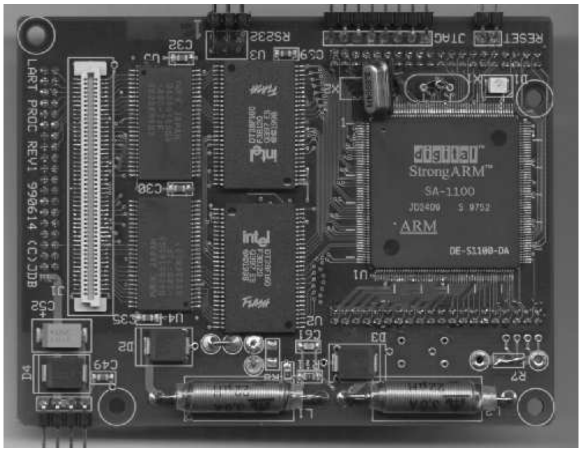

Figure 1: Low-power StrongARM embedded Linux platform (LART).

3.2 Measurement setup

To measure the power consumption of the LART, we used the configuration shown in Figure 2. The unregulated power of a battery is converted into a fixed 3.3 V for all the components on the board, except the processor. The processor voltage is supplied by a variable regulator. The accuracy of the measurements is within 2%.

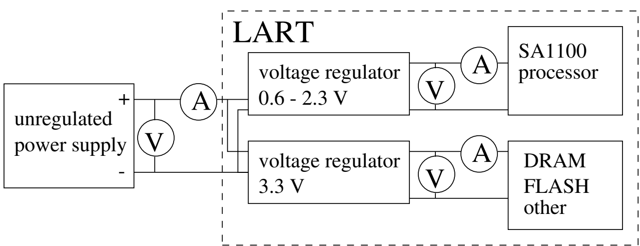

Figure 2: Measurement setup.

4. RESULTS

This section describes the effects of frequency and voltage scaling on the power consumption, processor performance, and memory performance of the LART. We ran several micro benchmarks (dhrystone, lmbench) as well as a full-blown H.263 video decoder. During each run the clock frequency and voltage were kept constant; dynamic voltage scaling is discussed in Section 5.

4.1 Required voltage

The first experiments determine for each clock frequency the minimum required voltage at which the SA-1100 processor still functions properly. We used the H.263 decoder to check if the processor functions at a given frequency and voltage combination. The resulting processor envelope is shown in Figure 3. Although the SA-1100 is specified for 190 MHz and a V_{DD} of 1.5 V, it can be over-clocked up to a frequency of 251 MHz with a V_{DD} of 1.65 V. The minimum clock frequency at which the processor functions correctly is 59 MHz with a V_{DD} of 0.79 V. Voltage scaling really pays off: at the lowest clock frequency the processor consumes

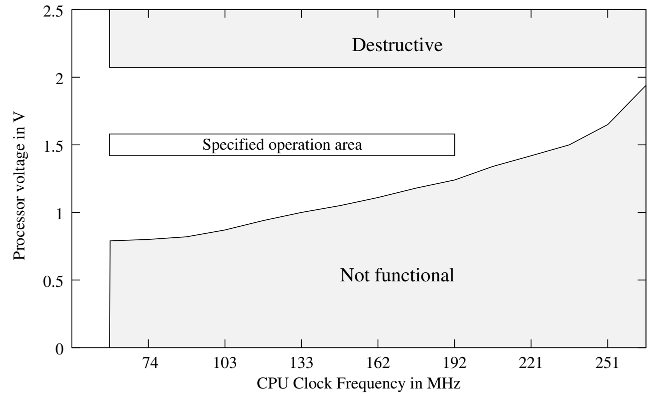

Figure 3: Processor envelope.

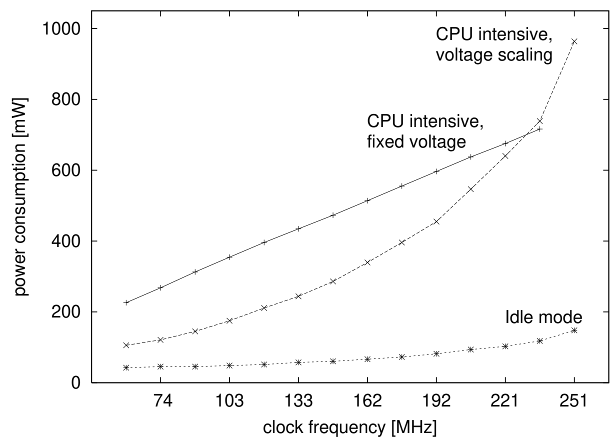

Figure 4: Total power consumption for idle and cpu-intensive workloads.

1/5 of the energy per instruction that is required at peak performance. Note that the voltage range and, hence, the efficiency gain of the SA-1100 is much larger than that of the Crusoe processor (see Table 1).

4.2 System performance

The next experiment determines the impact of voltage scaling on the power consumption of the complete LART (processor, memory, etc). Figure 4 shows the total power consumption of the LART under two different workloads: idle and cpu-intensive.

The idle workload was used to measure the background power consumption of the LART, which is always spent regardless of the processor load. The Linux scheduler puts the processor into idle mode when no processes are active. Idle mode stalls the CPU clock, but other services of the embedded processor such as the memory controller and OS-timer are still functional [8]. All these services are driven by the processor clock, which explains why the power consumption in idle mode increases with the frequency. The SA-1100 also supports a more efficient sleep mode, but this mode interrupts DMA transfers, stops the LCD controller, blocks memory access, etc. and the wake-up sequence takes much longer than in idle mode.

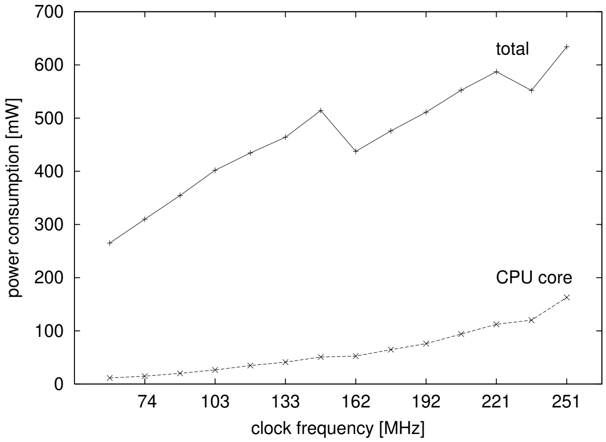

Figure 5: Power consumption for read.

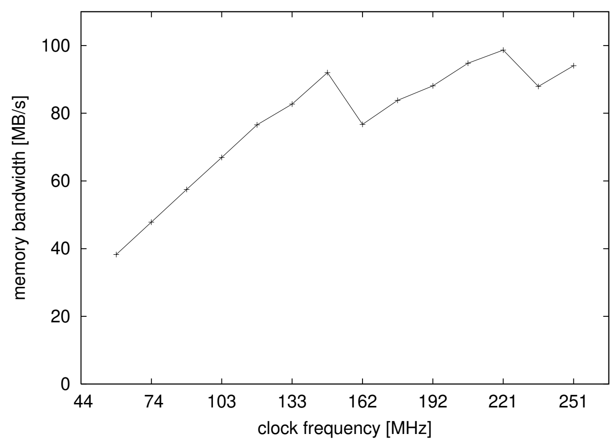

Figure 6: Memory bandwidth for read.

The cpu-intensive workload consists of the dhrystone benchmark exercising the CPU and cache that operate on the variable core voltage. We first measured the effect of scaling the clock frequency while keeping the voltage constant at 1.5 V. In this case the power consumption increases roughly linear with the frequency, as is expected. Next, we measured the power consumption when the core voltage was set to the minimal value reported in Figure 3. The resulting curve shows the expected quadratic increase of power consumption when the frequency is varied from 59 to 251 MHz.

From the power consumption at 59 MHz (105.8 mW) and at 251 MHz (963.7 mW) it follows that an instruction at peak performance consumes a factor 2.1 more energy than at lowest performance. When we only consider the processor power consumption, instead of the total LART, the difference is a factor 4.94 (33.1 versus 696.7 mW). This observation only holds for cpu-intensive applications; memory references introduce other effects as will be discussed next.

4.3 Memory performance

The impact of memory references on power consumption is difficult to predict since memory always operates at 3.3 V, while the processor core operates at a variable voltage. The LART has 32 MB of EDO-DRAM with an access time of 60 ns. We used the "lmbench" toolkit [15] to measure the power consumption while reading randomly from memory, circumventing the cache. The power consumption numbers in Figure 5 include voltage scaling, as is the case in all following figures. Note the general trend of a linear increase in power consumption, but with break downs occurring at 162 MHz and 236 MHz.

These break downs are not caused by the processor; the lower curve in Figure 5 plots the power consumption of the processor core only. Figure 6 shows that the obtained bandwidth at each frequency has similar dips. The explanation of this phenomenon is the limited capability of the StrongARM to generate high resolution DRAM timing waveforms. Moreover the DRAM waveform generator is driven by the system clock. The memory timing waveforms must be programmed by specifying a bit sequence that is used at subsequent clock pulses. Since the length of a clock pulse is not too fine compared to the (constant) DRAM timings, an optimal waveform can not be generated at each frequency.

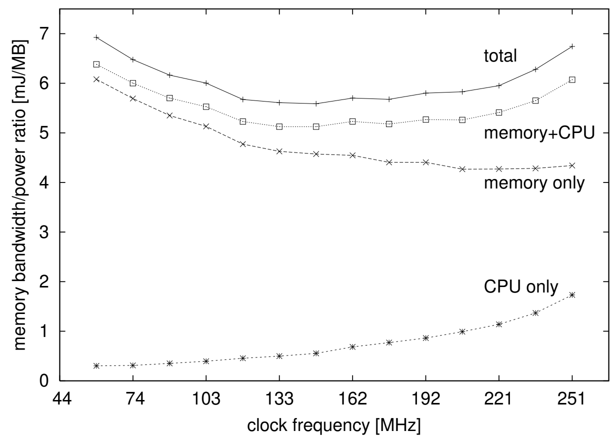

Figure 7: Energy breakdown for memory read.

It is instructive to combine the bandwidth and power consumption curves to show the relative cost at each frequency. Figure 7 gives the energy required to read one megabyte from memory. The "memory only" curve represents the energy drawn from the fixed 3.3 V, and shows that reading memory becomes relatively cheaper when the frequency increases. The "total" curve is the energy drawn from the batteries and includes both the memory, CPU activities, and voltage regulators. Initially the total energy drops, just as the "memory only" curve, but at higher frequencies the power consumed by the CPU increases considerably and forces the total energy to rise again. The difference between the "memory+cpu" curve and the "total" curve is the constant loss factor of the two voltage regulators. The best result is obtained at 148 MHz, where the bandwidth is 92 MB/s and the power consumption is 514.2 mW, with a cost of 5.6 mJ/MB.

4.4 Application performance

The power consumption of applications depends on the ratio between instructions and memory references. Figure 8 shows the power consumed by a publicly available Telenor H.263 video decoder in relation to the clock frequency. It also gives a breakdown in processor and memory power-

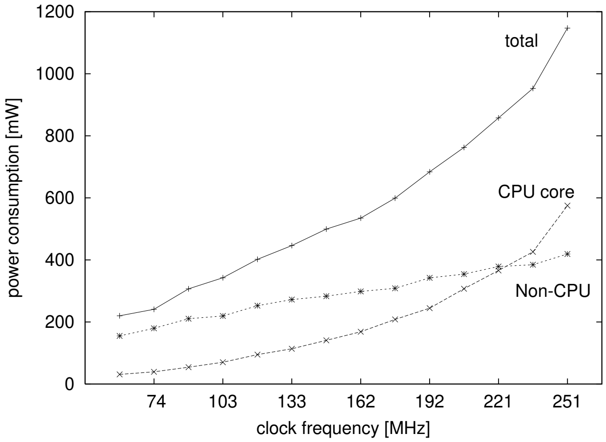

Figure 8: Power breakdown for H.263 decoder.

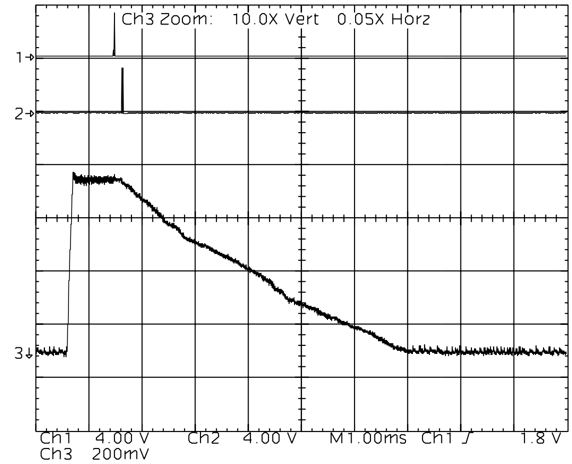

Figure 9: Scope image of voltage scaling.

consumption. At low frequencies the decoder is memory bound; at high frequencies the processor dominates.

5. DYNAMIC VOLTAGE SCALING

The results from the previous section were obtained under static conditions. In a real system, however, the frequency and voltage have to be set dynamically to match the changing demands for processing power. This is the responsibility of the Operating System (OS). The difficulty is that the OS has no direct knowledge of the workload generated by (bursty) applications, and must derive the optimal settings from external observations, for example, by monitoring the system load and estimating the future demand. This is a non-trivial task, for example, for the Mac OS it is hard to determine when no useful computation is occurring [13].

Predicting the future workload from the current situation is difficult, and mispredictions can seriously reduce the gains of voltage-scaling as observed in several simulation studies [6, 11, 17, 23]. These simulations use an interval-based scheduler with a time window of 5 to 100 ms. When idle time is detected within a window the clock speed is reduced proportionally in the next window. This reduces the idle-time to zero by running the applications as slow as possible. Whenever a window has no idle time, the speed is increased to accommodate for the unfinished work in the next window. Within a real-time OS where applications register deadlines and (worst-case) computational needs, the OS knows which amount of work must be compensated for in the next window and selects the appropriate processor speed. Most OSes, however, cannot determine the amount of unfinished work due to lack of real-time information, and increase the speed according to some heuristic. This is no problem for fairly constant workloads, but poor schedules result for bursty applications. For example, simulations with an MPEG decoder show that an additional 36 % energy reduction remains possible with a better workload prediction [17]. In this section we show that such gains can actually be achieved by making applications power-aware.

5.1 System support

A first requirement for any scheme to successfully exploit dynamic voltage scaling is that the processor frequency (and voltage) can be changed without much delay. To assess the responsiveness of the LART we added voltage scaling to the kernel module already providing frequency scaling. Whenever the frequency is changed, the module also sets the input voltage to the minimum level reported in Figure 3. The required frequency is read from /proc. /proc is a Linux pseudo-filesystem that is used as an interface to kernel data structures. The frequency and voltage scaling module creates an entry in /proc, which is readable and writable from user space. The kernel module was also modified to generate a pulse on an output pin just *before* changing the frequency and voltage, and another pulse on another output pin just *after* these changes.

We used a digital oscilloscope to measure the time required to change to a new frequency/voltage. The two upper lines on the scope image in Figure 9 show the pulses marking the beginning and the end of a clock/voltage change. The time difference is 140 μs. This time is insensitive to the frequency, and is needed to stabilize the internal clock at the new frequency.

The bottom line on the scope (labeled 3) shows how the processor-core voltage regulator responds to a steep increase, followed by a similar decrease in clock speed. The benchmark starts with 59 MHz at 0.8 V, then jumps to 221 MHZ at 1.5 V, stays at this level for about 1 ms, and finally returns back to 59 MHz at 0.8 V. The required processor-voltage increase is rapidly handled (40 μs), but the decrease takes a long time (5.5 ms). This is caused by the high capacitance of the regulator and the low power demand of the processor at 59 MHz. During the long delay, however, the energy is consumed from the regulator capacitance rather then the battery.

5.2 Power-aware applications

To overcome the problems of interval-based schedulers we propose that applications provide additional information about their future demands to the OS, so it does not need to work with questionable predictions. The drawback of this approach is that it requires modifications of the application source. This is, however, only required for bursty applications and, more importantly, applications running on a resource-scarce wearable device must be modified anyway. It is generally accepted that limitations imposed by low weight, small size, extensive battery life, wearable user interfaces, and wireless connectivity all have a profound effect on applications [1]. Without extensive adaptation of the applications to the wearable environment, *no* valuable service can be given, therefore modification of applications is inevitable.

Applications must be made aware of their processing demands and inform the OS about it, so the optimal processor speed can be selected that minimizes power consumption and still meets the application's deadlines. As a first step the application could indicate the required number of clock cycles (instructions) to the next deadline (absolute time). Combining the cycle count with the power-consumption curve in Figure 4 allows the OS to compute the lowest speed at which this application could meet its deadline. When multiple applications are time sharing the processor, the OS should take all constraints (deadlines) into account. A refinement is to have the applications express their demands in both instructions and memory references, which would yield better power consumption approximations.

6. POWER-AWARE VIDEO PLAYBACK

To determine how much effort is required to make applications power-aware, and the benefits that can be achieved, we conducted an experiment with a video playback application. In essence, we adapted the Telenor H.263 encoder to annotate video frames with information about the decoding complexity, so that the corresponding decoder can set the processor frequency considering the current status of the LART. In our experiment we used the 12.6 s carphone benchmark video, which was encoded at a rate of 15 fps with all compression optimizations enabled. The encoded file is rather small (98,480 bytes) and can be stored in the RAM-disk of the LART. Figure 10 gives the frame type and size for the carphone benchmark video. Note the large initial I-frame.

An interesting question is which complexity information to associate with a frame. It must be detailed enough to allow for good estimates of the decoding time, yet it must be simple enough to induce little overhead in computing and transmission. We found that the combination of frame type (I, P, or PB) and frame size (i.e., number of bits in the encoded stream) yields a complexity measure that is simple yet accurate. Figure 11 shows the results of measuring the decoding time on the LART of each frame in the carphone video. We measured the decoding times at different processor frequencies, 59 and 221 MHz, to take the non-linear memory bandwidth into account. At each frequency the results show a strong correlation between frame size and decoding time, with PB frames being more expensive to process than P frames. A slight complication with our complexity measure is that the frame size is not part of the standard H.263 headers. Fortunately, H.263 allows for additional signalling through the (optional) PEI field in the frame (picture) header, so a modified encoder can pass the frame size on to the decoder running on the LART.

Combining the decoding times from Figure 11 with the the strong variation of frame sizes in Figure 10 demonstrates that video decoding is a bursty application. Furthermore, video decoding is a demanding application since none of the frames can be decoded within the required 67 ms (i.e. 15 frames per second) at the lowest frequency of 59 MHz. On the other hand, running the CPU at high speed (221 MHz)

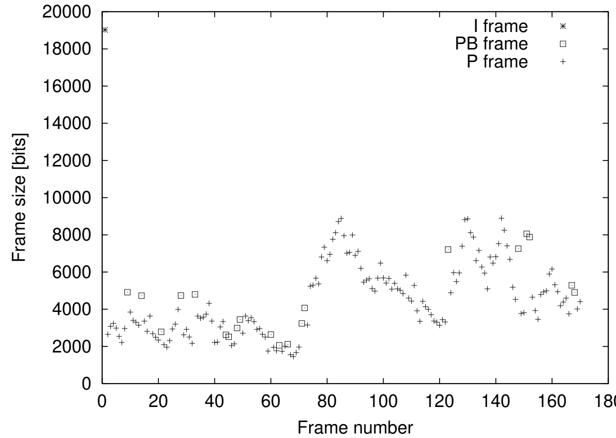

Figure 10: Frame size variation over time.

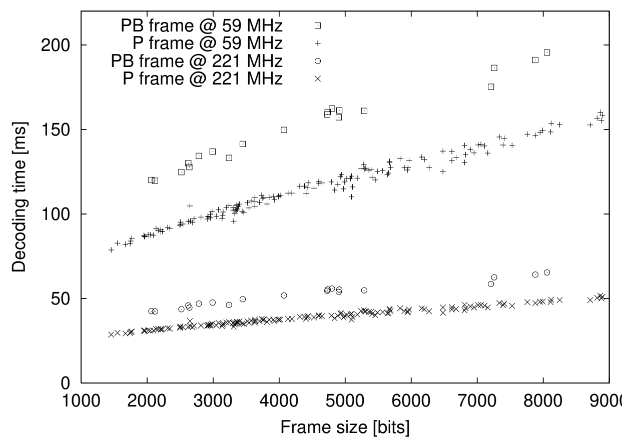

Figure 11: Decoding time vs. frame size&type.

is only necessary for the two largest PB frames. Video decoding is thus a good test case for dynamic voltage scaling.

We used the experimental setup discussed in Section 3.2 to measure the power consumption of a modified H.263 decoder that reads a play list specifying the frequency at which each individual frame should be decoded. Since subsequent frames in the carphone video often differ considerably in size and sometimes in type (see Figure 10), the decoder switches 167 times to another frequency out of a possible 191 (190 frames and 1 return to idle). We sampled the voltage and current at a rate of 2.5 kHz and calculated the energy consumption. In case of the carphone video, the average power for decoding is 304 mW (100 mW for the CPU, and 204 mW for the memory subsystem). For comparison, we also measured power when decoding at a fixed frequency of 236 MHz, which averages to 405 mW (198 mW for the CPU, and 207 mW for the memory subsystem). Thus, dynamic voltage scaling reduces the CPU power with a factor of two, but total system power only with 25 % since the cost of memory accesses is nearly the same with the identical workload. We expect that the overall effect of voltage scaling can be increased by properly optimizing the H.263 decoder to take the size of the cache, which is part of the scalable processor core, into account. For example, large

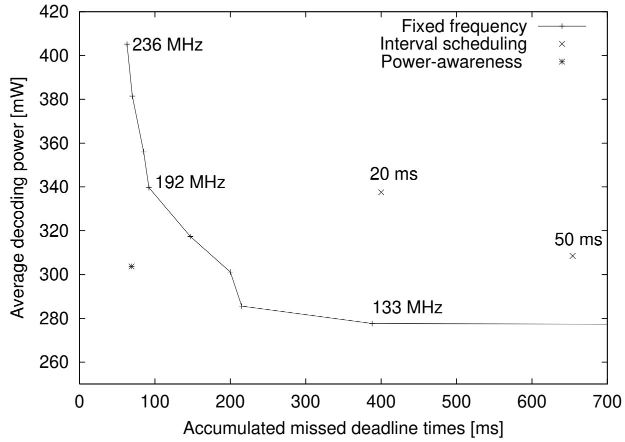

Figure 12: Power-quality tradeoff.

look-up tables are ineffective on the LART with its small data cache of 8 KB, and degrade performance.

The carphone video can also be decoded at lower (fixed) frequencies, but this results in missed frame deadlines. We modified the H.263 decoder to record missed deadlines, and report the accumulated miss times at the end. With this quality measure it is possible to study the trade-off between power and quality. Our accumulated miss times metric is similar to the clipped-delay metric in the simulations by Pering [17]. Figure 12 shows the trade-off when decreasing the (fixed) frequency from 236 MHz (304 mW, 63 ms) down to 133 MHz (278 mW, 388 ms). All measurements have an equal decoding time. Note that in all cases deadlines are missed. This is caused by the initial I-frame that cannot be decoded within 67 ms, even at the highest frequency. This also explains why the power-aware application with its perfect knowledge misses deadlines.

Running the application at some fixed frequency is not a practical solution, hence, we programmed an interval-based scheduler. It is implemented as a user-space process that periodically reads the idle times from the /proc system statistics and instructs the kernel to change the frequency (and voltage) by writing into /proc. The policy is to lower the frequency by one step whenever the fraction of idle time exceeds a threshold (currently set at 50 %), and to increment it when no idle time is recorded in the last interval. At successive increments the step size is doubled to scale up rapidly. The performance for two different interval lengths (20 ms and 50 ms) is shown in Figure 12. Note that in comparison to the fixed-frequency schemes the interval scheduler performs worse. For example, the 20 ms scheduler operates with an average power of 337 mW and causes 400 ms of missed deadlines; running at 192 MHz requires the same power, but reduces the missed deadlines to only 92 ms, while running at 133 MHz incurs a similar miss time, but requires less power (278 mW). Significant improvements need to be made before interval scheduling can handle bursty applications as well as basic fixed-frequency scheduling, let alone power-aware applications.

7. RELATED WORK

With the development of experimental voltage scaling platforms such as the Itsy by Compaq [4], the ARM8 implementation by Berkeley [16], and our LART system it becomes finally possible to measure the effectiveness of various approaches to dynamic voltage scaling previously developed on the basis of theoretical analysis [9] and simulation [6, 11, 17, 23]. Practical experience is vital to determine the best approach for exploiting commercial processors with voltage scaling support that are currently appearing on the market: the Transmeta Crusoe processor, Intel's XScale, and the K6-IIIE by AMD.

The various approaches to managing variable voltage processors differ in the amount of information that is available. We distinguish four classes: hardware-based (no information), interval-based (load information only), integrated schedulers (all OS statistics), and application-specific (complete knowledge). The more information is available, the better results will be obtained.

The Transmeta Crusoe processor is the prime example of the hardware-based approach. It has built-in support for clock scheduling in the "microcode" of the processor [21]. Unfortunately little information is made available about the exact workings of the "LongRun" technology, but it is clear that it operates in isolation, that is, without any help form the OS or application. Our limited experience with the Crusoe shows that it, consequently, exhibits an all-or-nothing behavior switching rapidly between full and minimal speed. This results in wasting energy compared to running at intermediate speeds.

Adding an interval-based clock scheduler to the OS is the topic of many simulation papers. A typical example are the simulations by Pering [17] showing that interval-based clock scheduling at the OS-level reduces processor power considerably. The realized efficiency, however, is extremely dependent on the interval length and application. It is difficult to choose an interval such that all applications can be scheduled efficiently. In particular bursty applications with a variable resource requirement "fall significantly short of optimal". This observation is confirmed by recent experimental work on the ARM8 [18] and Itsy [7], and matches the (poor) performance results obtained by the interval-based scheduler developed for the LART when running the H.263 video decoder.

To improve the class of applications that can be handled transparently by the OS, other information than the processor load can be used to predict the future demands of the application. This requires a tight integration of the variable-voltage scheduler and the OS. For example, Flautner et al. [5] describe an integrated scheduler that maintains processor usage statistics of every process, observes the communication pattern between processes, keeps track of input/output device usage by processes, and tries to extract deadlines from periodic tasks such as video decoding. Unfortunately the simulation results presented do not include a comparison with a traditional interval-based scheduler, so the advantage of using additional information is still to be determined. To increase the scope for improvements at the OS level, several researchers have proposed to use soft deadlines [5, 14]. For example, Lorch et al. exploit the observation that a reaction time of 50 ms for interactive applications is below the perception threshold of the user. Therefore, the application processing time for, say, a mouse click can be increased to 50 ms (by slowing down the CPU) without noticeable performance degradation. Off-line simulations show that the upper bound on the additional energy saving is in the order of 20%. It remains to be seen how much energy can be actually saved in a real implementation.

The approach that will save most energy is to involve the application in determining the lowest possible clock speed. This is the approach taken in this paper, and good results are demonstrated for bursty workloads as generated by a modified H.263 video decoder. The main drawback is that applications become more complex, because they must provide hints about their future behavior (e.g., processing demands, deadlines) to the OS. To alleviate the task of the application developer Shin et al. discuss an approach based on automatic compile-time analysis [20]. Their software tool analyses the source code of an MPEG 4 decoder and adds a clock schedule that is guaranteed to meet hard real-time deadline requirements (i.e. frame deadlines). The worst case execution times, however, are rarely encountered in natural video sequences and, consequently, the clock speed is set too high wasting energy.

8. CONCLUSIONS AND FUTURE WORK

Power consumption is the limiting factor for the functionality of future wearable devices. Since interactive applications like wireless information access generate bursts of activities, it is important to match the performance of the wearable device accordingly. A popular approach is using power-down modes to minimize the power consumption of unused hardware like disks, screen, etc. In the case of the processor, better results can be obtained by scaling the speed and voltage to match the required performance level, since power consumption is quadratically related to the supply voltage. Simulations have pointed out the potentials of voltage scaling.

Within the UbiCom project we have assembled a wearable system that supports dynamic voltage scaling. It is based upon the low-power embedded SA-1100 processor, whose frequency can be varied from 59 MHz to 251 MHz. The required supply voltage varies from 0.8 V to 2.0 V. Measurements show that the power consumption of the processor at 59 MHz is 33.1 mW, while it consumes 696.7 mW at 251 MHz. It follows that the energy per instruction at minimal speed is 1/5 of the energy required at full speed. This result reconfirms the importance of voltage scaling.

We have added kernel support to Linux that allows running applications to scale the frequency and voltage from user space in only 140 μs. This allows power-aware applications to quickly adjust the performance level of the processor whenever the workload changes. An experiment with power-aware video playback showed that a simple complexity measure (frame size & type) associated with each frame allows for an efficient control of the frequency (and voltage). The power-quality results show that the power-aware decoder outperforms both a static fixed-frequency policy as well as a dynamic interval-based scheduler that responds to the load average.

Our future plans are to extend the application-specific approach for dynamic voltage scaling to power management in general. Within UbiCom we are developing a framework that is based on the explicit exchange of performance and power consumption information between hardware devices (CPU, hard disk, wireless link, etc.) OS, and applications. The explicit exchange of information will allow us to perform intelligent and efficient power management for the complete wearable Ubicom system.

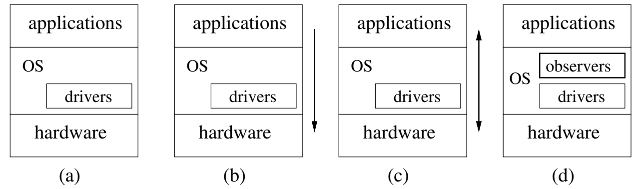

Figure 13: Four power management frameworks.

Figure 13 shows four power management frameworks with three different layers: hardware, OS, and application. Framework (a) is the traditional framework without performance-power consumption exchange, the situation for interval-based clock schedulers. Within Framework (b) applications specify their future requirements to the lower layer and hardware devices can be scheduled more efficiently, as shown in this paper. Framework (c) demonstrates interaction between applications and hardware. An example of such interaction would be a power-aware video decoder that meets almost all frame decoding deadlines, yet misses the deadlines for the complex and power expensive frames. Such a decoder would extend the single power-awareness data point in Figure 12 into a curve that is more power efficient than the fixed frequency curve. Framework (d) adds observers that log all application requests and try to predict future requests for applications that do not specify their hardware needs (similar to integrated schedulers). The purpose of including observers is to improve the energy efficiency of legacy codes.

Currently we are working on an implementation of framework (d) on our LART platform. The target application is wireless audio and video playback with a guaranteed battery lifetime that is specified by the user. Using the power consumption information of the hardware devices (CPU, hard disk, wireless link) and the application's ability to scale the image and sound quality, we can infer the control settings that provide the best quality without draining the battery completely before the user-defined target time.

Acknowledgements

We would like to thank Jan-Derk Bakker and Erik Mouw for providing us with an excellent low-power platform, and assisting us with the measurements and their interpretation. We thank Hylke van Dijk and the anonymous reviewers for commenting on the draft version of this paper.

9. REFERENCES

[1] O. Angin, A. Campbell, M. Kounavis, and R. Liao. The MobiWare toolkit: Programmable support for adaptive mobile networking. *IEEE Personal Communications*, Aug. 1998.

[2] J.-D. Bakker, J. Mouw, and M. Joosen. Linux Advanced Radio Terminal design page. `http://www.lart.tudelft.nl/`

[3] T. Burd and R. Brodersen. Processor design for portable systems. *Journal of VLSI Signal Processing*, Aug. 1996.

[4] Compaq, Western Research Lab. The Itsy (version 2) pocket computer, overview slides. `http://research.compaq.com/wrl/projects/itsy/`

[5] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. In

7th ACM Int. Conf. on Mobile Computing and Networking (Mobicom), Rome, Italy, July 2001.

[6] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *MobiCom*, Berkeley, CA, Nov. 1995.

[7] D. Grunwald, P. Levis, K. Farkas, C. Morrey, and M. Neufeld. Policies for dynamic clock scheduling. In *OSDI*, San Diego, CA, Oct. 2000.

[8] Intel StrongARM SA-1100 microprocessor developer's manual. `http://developer.intel.com/design/strong/manuals/278088.htm`

[9] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISPLED*, Aug. 1998.

[10] R. Lagendijk. The TU-Delft Research Program Ubiquitous Communications. In *21st Symp. on Information Theory in the Benelux*, Wassenaar, The Netherlands, May 2000.

[11] Y. Lee and C. Krishna. Voltage-clock scaling for low energy consumption in real-time embedded systems. In *6th Int. Conf. on Real-Time Computing Systems and Applications*, 1998.

[12] J. Lorch. The complete picture of the energy consumption of a portable computer. Master's thesis, UC Berkeley, Dec. 1995.

[13] J. Lorch and A. Smith. Scheduling techniques for reducing processor energy use in MacOS. *Wireless Networks*, 1997.

[14] J. Lorch and A. Smith. Improving dynamic voltage scaling algorithms with pace. In *Sigmetrics 2001*, Cambridge, MA, June 2001.

[15] L. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conference*, San Diego, CA, Jan. 1996.

[16] T. Pering, T. Burd, and R. Brodersen. Dynamic voltage scaling and the design of a low-power microprocessor system. In *ISCA*, 1998.

[17] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *ISPLED*, Aug. 1998.

[18] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lpARM microprocessor system. In *ISPLED*, July 2000.

[19] J. Pouwelse, K. Langendoen, and H. Sips. A feasible low-power augmented-reality terminal. In *2nd IEEE and ACM Int. Workshop on Augmented Reality (IWAR'99)*, San Francisco, CA, Oct. 1999.

[20] D. Shin, S. Lee, and J. Kim. Intra-task voltage scheduling for low-energy hard real-time applications. In *IEEE Design & Test of Computers*, Mar. 2001.

[21] Transmeta-corporation. The technology behind the Crusoe processor. `http://www.transmeta.com/crusoe/download/pdf/crusoetechwp.pdf`

[22] Transmeta-corporation. Tm5400 processor specifications. `http://www.transmeta.com/crusoe/download/pdf/TM5400_ProductBrief_5-23-00.pdf`

[23] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *OSDI*, pages 13–23, Nov. 1994.