

An Application-Specific and Adaptive Power Management Technique*

Bernhard Egger Jaejin Lee Heonshik Shin

School of Computer Science and Engineering
Institute of Computer Technology
Seoul National University, Seoul 151-744, Korea
egger@cselab.snu.ac.kr, jlee@aces.snu.ac.kr, shinhs@snu.ac.kr

Abstract—This paper describes an adaptive execution technique for saving energy in systems that support dynamic voltage (and frequency) scaling (DVS). We show that our method is most effective for periodic workloads such as video or audio decoding. It exploits both memory-bound code regions as well as idle time, while solely relying on time and power profiles of the system. These profiles are used to accurately estimate the execution time and the total system energy consumption for different code regions. To estimate the power consumption and execution time for a given code region, we analyze the characteristics of the executed code at run time by querying the performance counters provided by the CPU. We have implemented our method in a multitasking operating system (Microsoft Windows CE) running on an Intel XScale-processor. We achieved up to 9% of total system power savings over the standard power management policy that puts the CPU in a low power mode during idle periods.

I. INTRODUCTION

As battery-operated portable devices become increasingly popular, satisfying demands for effective energy reduction techniques becomes more important. The device should use as little energy as possible while still providing satisfactory performance; in other words, the quality of service (QoS) requirement must be met.

Dynamic Voltage Scaling (DVS) is an effective method to reduce power consumption of the CPU [4] and is supported by a number of processors on the market such as AMD's Mobile Athlon [2], Intel's XScale [11], and Transmeta's Crusoe [7]. Energy can be saved because not only is the CPU frequency scaled linearly, but so is the CPU voltage. Since energy is proportional to the square of the voltage, reducing the voltage along with the frequency yields significant energy savings [5].

There are basically two ways to save energy without severely degrading performance. One is detecting and exploiting slack and idle time. This approach is common for real-time systems where task deadlines are known in advance. By modifying the task scheduler or analyzing the (remaining) worst case execution time (WCET), slack and/or idle time is exploited to run the CPU at a lower frequency, thereby saving power [1][3][14][18][19]. In non real-time multitasking operating systems, there is no slack time and the idle time varies with the system load and is not known in advance. There are several approaches to estimate the future idle time. Weiser's original

algorithm, called PAST [21], predicts the idle time based on the idle time observed in the past. More sophisticated and accurate prediction algorithms have been developed in [16][20].

The second way to save energy without severely degrading performance is to use DVS in memory-bound and CPU-bound code regions. If the target architecture supports asynchronous memory accesses, the CPU stalls frequently during the execution of memory-bound code regions due to frequent cache misses. For these regions, reducing the CPU frequency will not affect performance significantly because the execution time is dominated by the memory access latency. This approach can be found in [8][17][22].

This paper introduces an application-specific and adaptive power management technique. Our goal is to design an adaptive execution algorithm that reduces the total system energy using DVS without modifying the scheduler. An adaptive power manager (APM) is linked to the operating system, and user programs call the power manager's API. Our work integrates both of the two methods mentioned above. It detects memory-bound code regions at run time and slows down the CPU in these regions whenever these regions are entered again. Moreover, it monitors the idle time to detect periodic behavior and then prolongs the execution time of these periodic regions by reducing the CPU clock frequency. Some previous work [6][9] specifically targets MPEG decoding and evaluates the system energy state at the frame level (i.e., the power APIs are called whenever decoding of a new video frame begins). However, these program points are difficult to detect at compile time without deep knowledge of the application. Our technique, in contrast, can detect periodic behavior even if the APM API calls do not occur at these specific program points.

II. ADAPTIVE POWER MANAGEMENT

Our adaptive power management scheme can be split into three processes: characterization of the system, inserting APM APIs, and adaptive execution.

A. Power and Execution Time Characterization

The CPU power consumption depends on the cache miss ratio and the memory-to-ALU instruction ratio. In CPU-bound code regions with few memory accesses and a low cache miss ratio, the CPU has a high power consumption because almost no CPU stalls occur, whereas in memory-bound code regions with many memory accesses and a high cache miss ratio, the CPU consumes less power due to memory stalls. On the other hand, the memory system consumes more power in

*This work was supported in part by Microsoft Research Innovation Excellence Awards for Embedded Systems, and by the Ministry of Education under the Brain Korea 21 Project in 2004.

memory-bound code regions and less power in CPU-bound code regions.

The execution time of CPU-bound code regions is dominated by the number of instructions, while that of memory-bound code regions is dominated by the number of memory operations and the cache miss ratio.

To obtain accurate characteristics of the target system, we execute synthetic benchmarks with different instruction mixes. We measure the overall system power consumption and the execution time for different instruction mixes, as well as the power consumption when the CPU is in idle mode. The power and execution time measurements are then approximated by linear functions. Using these functions, we construct power and execution-time estimation models that will be used by the APM at run time. We explain our estimation models in detail later.

B. Inserting APM APIs

The compiler or the user selects some interesting regions (say a loop-nest) with respect to power management and encloses them with a pair of APM API calls. Figure 1 shows an example. These regions can be identified by profiling or by analyzing the code using a static code partitioning technique such as the one described in [13] that basically selects loop-nests.

C. Adaptive Execution

The APM is invoked by applications which have been instrumented with APM calls. For each region, the APM reads the CPU performance counters and determines their characteristics. Depending on the data obtained, the APM operates in two energy optimization modes.

1) Optimizing energy consumption without idle time

The APM monitors the characteristics (instruction count, cache accesses, and cache misses) of each region separately and determines the best CPU frequency by estimating the execution time and energy consumption of the region. At the beginning of the region, the clock is switched to the best estimated frequency and, at the end of the region, back to the original frequency. The measured execution time is compared to the estimated time and a correction factor is computed and used in successive estimations to improve accuracy. Whenever the behavior of a region changes significantly, the APM recalculates the time and energy requirements of the different frequencies and switches to the one that minimizes energy consumption. A maximum relative performance penalty limit can be used to ensure QoS.

Our experiments show that this type of optimization cannot be exploited on the Intel PXA255 processor. Changing the CPU frequency also modifies the CPU-to-memory bus speed. Thus, reducing the CPU frequency in memory-bound regions also slows down memory operations, which results in an increased total energy consumption.

```

x = ...;
apmRegionStart(id);
for (i=1; i<N; i++) {
    a = i * ...;
    do {
        x += a + ...;
    } while (a);
}
apmRegionStop(id);
y = c(x);

```

Fig.1. A loop-nest marked with a pair of APM calls

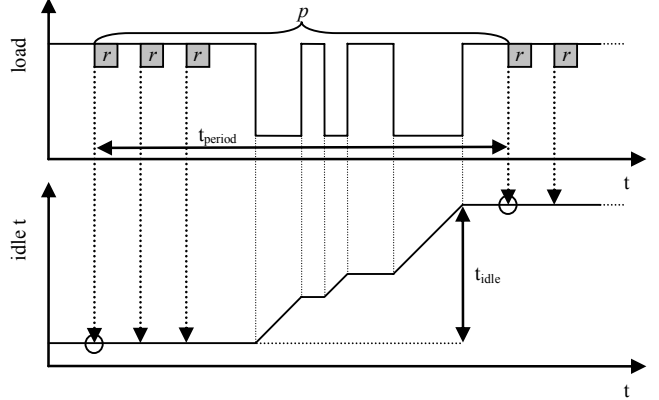


Fig.2. Detecting periods by looking at the total system idle time. The APM API calls are placed around the region r (grey boxes). The APM detects the periodic region p by looking at the idle time. In this example, a new period starts every 3 invocations of region r .

2) Optimizing energy consumption with idle time

Whenever the application with the highest priority calls the APM, the latter monitors the system idle time to detect periodic behavior. At the beginning of a region, the total system idle time is checked. If the idle time between the last and the current invocation increases, the system has been idle between the two invocations of the region. The current region is marked as the head of a periodic region p . The region p ends as soon as the idle time at the beginning of region r increases. Figure 2 illustrates the idea.

Once a periodic region p is detected, the characteristics of the code executed over a full period are measured. Then, the APM estimates the CPU frequency that minimizes the overall system energy consumption under the constraint of keeping the current QoS level (i.e., slowing down the CPU just as much such that there is no idle time left).

The APM constantly monitors and compares the measured execution time to the estimated execution time. For each frequency, it computes a correction factor. The next estimated execution time is then multiplied by this factor to improve accuracy.

III. ESTIMATING EXECUTION TIME AND POWER CONSUMPTION

Accurate estimation of execution time and power consumption is one of the most crucial parts of our method. We execute synthesized workloads and measure the overall system energy consumption as well as the execution time for all of these workloads. For each frequency, we vary the cache miss

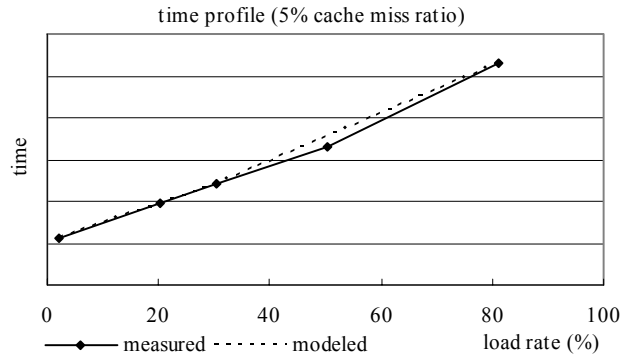


Fig.3. Measured and modeled execution time for varying load rates at 5% cache miss ratio

TABLE I: Power table

$f_{CPU} = 99\text{MHz}$	load rate 2%	load rate 20%	load rate 80%
cache miss ratio 0%	2.66500	2.62109	2.54678
cache miss ratio 3%	2.58255	2.50377	2.45716
cache miss ratio 5%	2.56164	2.51533	2.43093
cache miss ratio 10%	2.52168	2.49035	2.42781
...

ratio and the memory-to-ALU ratio. Figure 3 shows the execution time profile when the cache miss ratio is 5% with varying memory-to-ALU ratio (i.e., load rate). The measured data is approximated by several linear functions and stored in a table inside the APM (Table I).

To estimate the power consumption and execution time of a given region r , we use the following formulas:

$$P_r = P(f, instr, acc, miss)$$

$$T_r = T(f, instr, acc, miss)$$

where f , $instr$, acc , and $miss$ are the frequency, the number of instructions, the number of cache hits, and the number of cache misses respectively. By linearly interpolating the data in the tables, we can estimate the power and time consumption for any given load rate ($acc/instr$) and cache miss rate ($miss/acc$).

For a region r , the energy consumption is estimated by the following formula:

$$E_r = P(f_{new}, instr, acc, miss) \cdot T(f_{new}, instr, acc, miss) + E_{switch}(f_{old}, f_{new})$$

where $E_{switch}(f_{old}, f_{new})$ is the overhead to switch from f_{old} to f_{new} and back.

The energy consumption for a periodic region p can be computed by

$$E_p = n \cdot E_r(f_r, instr_r, acc_r, miss_r) + P(f_p, instr_p, acc_p, miss_p) \cdot T(f_p, instr_p, acc_p, miss_p) + P_{idle}(f_p) \cdot (t_p - n \cdot T_r - T(f_p, instr_p, acc_p, miss_p))$$

under the constraint

$$t_p \geq n \cdot T_r + T(f_p, instr_p, acc_p, miss_p)$$

where t_p is the measured period of p ; n is the number of regions per period; $instr_r$, acc_r , and $miss_r$ are the performance counters for the region r ; and $instr_p$, acc_p , and $miss_p$ are those for the entire period p without the counters for the regions.

IV. EVALUATION ENVIRONMENT

We implemented our APM for Microsoft Windows CE.NET 4.2 [15] running on a development board with an Intel XScale PXA255 processor [10]. The board is equipped with a PXA255 processor, 64MB of SDRAM, 64MB of Flash memory, Ethernet, USB (master and slave), Infrared, and a 320x160 pixel-wide LCD screen. The board is powered by two voltages; 3.3V and 5.0V. The power measurements were performed using a high-performance data acquisition device that can sample 3.3V and 5.0V at 100 KHz simultaneously.

The PXA255 supports a total of 10 different frequency settings, which differ not only in terms of CPU core clock frequency, but also in terms of the main processor bus speed and the SDRAM frequency (Table II).

We used *mpeg2dec* from Mediabench [12] and a synthesized

TABLE II: DVS frequencies of the PXA255 processor

index	f_{CPU}	$f_{CPU-MEM}$	$f_{MEM-LCD}$	f_{SDRAM}
0	99.5	50	99.5	99.5
1	199.1	50	99.5	99.5
2	298.6	50	99.5	99.5
3	132.7	66	132.7	66
4	199.1	99.5	99.5	99.5
5	298.6	99.5	99.5	99.5
6	398.1	99.5	99.5	99.5
7	265.4	132.7	132.7	66
8	331.8	165.9	165.9	83
9	398.1	196	99.5	99.5

benchmark program, *3dview*, for evaluating our techniques. *Mpeg2dec* is an MPEG2 decoder, and *3dview* simulates a three-dimensional view of a virtual world. *3dview* consists mainly of matrix transformations but does not draw the view on the screen.

The version of Windows CE that shipped with our development board does not support the system call *GetIdleTime()*. Because of this, the time between the end of processing one frame and the beginning of the next frame is treated as idle time. Since this approach cannot capture any other activity in the system (e.g., display activity) our simulated idle time is the upper bound of the actual idle time.

V. EXPERIMENTS

Our experimental results show that a frequency setting 9 with a CPU clock of 398MHz which also has the fastest CPU-to-memory bus speed is the optimal solution in terms of energy consumption for any given code region if there is no idle time (Figure 4). This is in contrast to the results obtained in [8]. The reason is that executing memory-bound regions under the maximum frequency will consume more total system energy due to the slower CPU-memory bus and thus longer execution time. However, idle time can still be exploited.

We identified the hotspots of the standard MPEG decoder *mpeg2dec* and inserted APM calls around them. The APM detects the periodic behavior of the APM calls and scales down to a lower frequency. At the beginning of the run, the execution time estimation is not very accurate. The APM under- or overestimates the execution time at a certain frequency, which is detected and corrected in later periods. Table III summarizes the results for *mpeg2dec*. Compared to the normal Windows CE power management policy (the CPU is in a low-power mode during idle periods), we are able to save 4% of total system power for a 256x192 pixel-wide, 7% for a 160x120 pixel-wide video, and 8% for an 80x60 pixel-wide video, all with jitter still within acceptable ranges.

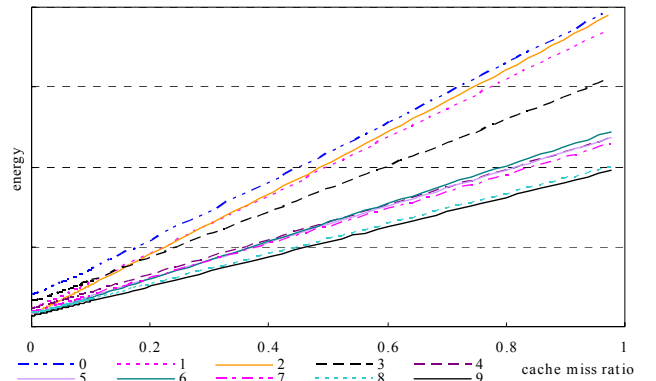


Fig. 4. Energy consumption per cache miss ratio

TABLE III: Power consumption for mpeg2dec

video size	Standard PM		APM		Power savings (%)
	avg. power [W]	avg. jitter [ms]	avg. power [W]	avg. jitter [ms]	
80x60	2.873	1.10	2.644	1.29	8.0
160x120	3.189	1.10	2.955	2.06	7.3
256x192	3.507	1.12	3.365	1.25	4.0

In *3dview*, the positions of objects are calculated using several matrix transformations. The number of objects varies over time resulting in a varying CPU workload. The APM calls are placed around the regions in which the matrix operations are performed.

Figure 5 (a) shows the complexity of the scene (i.e., number of objects) over time, (b) the idle time (in percentage of the entire period) of the system, and (c) the CPU frequencies chosen by the APM. While the achievable energy savings depend mostly on the complexity of the scene (i.e., number of objects), applying our technique to the varying workload depicted in Figure 5 (a) reduced the overall system energy consumption by 9% with an average jitter of 0.02ms per frame.

VI. CONCLUSIONS

We introduced an adaptive DVS algorithm that exploits memory-bound code regions as well as system idle time. We recognize memory-bound code regions by reading the performance counters provided by the CPU. Periodic regions are detected by checking the total system idle time at every invocation of the region.

Our estimation is based solely on a power/time characterization of the whole system that is measured beforehand.

Our experiments were performed on an XScale development platform that, as a whole, consumes a lot of energy compared to real battery-powered mobile systems. Nevertheless, we show energy savings of up to 9% for periodic tasks such as MPEG video decoding or 3D world rendering with varying system workload.

REFERENCES

- [1] N.AbouGhazaleh, *et al.* Collaborative Operating System and Compiler Power Management for Real-Time Applications. In *Proceedings of the 9th Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, May 2003.
- [2] Advanced Micro Devices. AMD Power Now Technology. 2000.
- [3] H.Aydin, *et al.* Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *the 22nd IEEE Real-Time Systems Symposium*, December 2001.
- [4] T.Burd, R.Brodersen. Energy efficient CMOS microprocessor design. In *the 28th Hawaii International Conference on System Sciences*, 1995.
- [5] A.Chandrakasan, and R.Brodersen. Low Power Digital CMOS Design. *Kluwer Academic Publishers*, 1995.
- [6] K.Choi, R.Soma, M.Pedram. Off-chip Latency-Driven Dynamic Voltage and Frequency Scaling for an MPEG Decoding. In *Design Automation Conference (DAC'04)*, June 2004.
- [7] M. Fleischmann. Longrun Power Management. White Paper of Transmeta Corporation, January 2001.
- [8] C-H. Hsu, U.Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *Programming Language Design and Implementation (PLDI'03)*, June 2003.
- [9] C.Hughes, J.Srinivasan, S.Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, December 2001.
- [10] Intel PXA255 Processor. Developer's Manual, January 2004.
- [11] Intel XScale Microarchitecture. <http://developer.intel.com/design/intelxscale/>

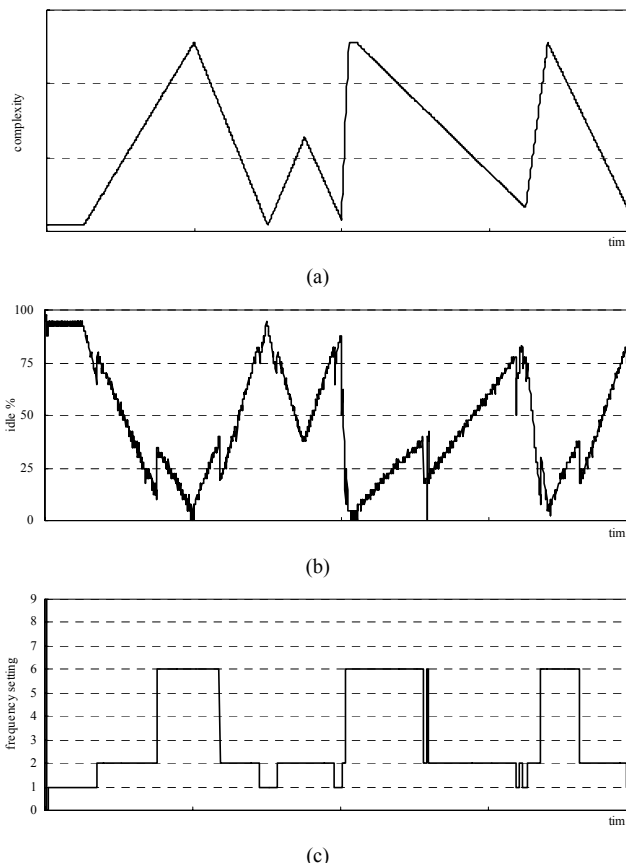


Fig. 5 Scene complexity (a), idle percentage (b), and CPU frequency settings (c)

- [12] C.Lee, M.Potkonjak, W.H.Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *30th Annual International Symposium on Microarchitecture (Micro '97)*, December 1997.
- [13] J.Lee, Y.Solihin, J.Torellas. Automatically Mapping Code on an Intelligent Memory Architecture. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)*, January 2001.
- [14] J.Lorch, A.Smith. Improving dynamic voltage algorithms with PACE. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2001)*, June 2001.
- [15] Microsoft. Windows CE.NET. <http://msdn.microsoft.com/embedded/ce.net/>
- [16] G.A.Paleologo, L.Benini, G.De Micheli. Policy Optimization for Dynamic Power Management. In *Design Automation Conference (DAC'98)*, June 1998.
- [17] H.Saputra, *et al.* Energy-Conscious Compilation Based on Voltage Scaling. In *Joint Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'02) and Software and Compiler for Embedded Systems (SCOPES'02)*, June 2002.
- [18] D.Shin, J.Kim, and S.Lee. Low-Energy Intra-Task Voltage Scheduling Using Static Timing Analysis. In *Design Automation Conference (DAC'01)*, June 2001.
- [19] V.Swaminathan, K.Chakrabarty. Investigating the effect of voltage switching on low-energy task scheduling in hard real-time systems. In *Asia South Pacific Design Automation Conference (ASP-DAC'01)*, January/February 2001.
- [20] A.Varma, *et al.* A Control-Theoretic Approach to Dynamic Voltage Scheduling. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'03)*, October 2003.
- [21] M.Weiser, *et al.* Scheduling for Reduced CPU Energy. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, Usenix Association, November 1994.
- [22] F.Xie, M.Martinosi, S.Malik. Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits. In *Programming Language Design and Implementation (PLDI'03)*, June 2003.