

Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-chip Access to On-chip Computation Times

Kihwan Choi, Ramakrishna Soma, and Massoud Pedram

Abstract— This paper presents an intra-process dynamic voltage and frequency scaling (DVFS) technique targeted toward non real-time applications running on an embedded system platform. The key idea is to make use of runtime information about the external memory access statistics in order to perform CPU voltage and frequency scaling with the goal of minimizing the energy consumption while translucently controlling the performance penalty. The proposed DVFS technique relies on dynamically-constructed regression models that allow the CPU to calculate the expected workload and slack time for the next time slot, and thus, adjust its voltage and frequency in order to save energy while meeting soft timing constraints. This is in turn achieved by estimating and exploiting the ratio of the total off-chip access time to the total on-chip computation time. The proposed technique has been implemented on an XScale-based embedded system platform and actual energy savings have been calculated by current measurements in hardware. For memory-bound programs, a CPU energy saving of more than 70% with a performance degradation of 12% was achieved. For CPU-bound programs, 15~60% CPU energy saving was achieved at the cost of 5-20% performance penalty.

Index Terms—Dynamic voltage and frequency scaling, Energy minimization, Low power, Energy and performance trade-off.

I. INTRODUCTION

DEMAND for low power consumption in battery-powered computer systems has risen sharply. This is because extending the service lifetime of these systems by reducing their power requirements is a key customer/user requirement. More recently, low power design has become a critical design consideration even in high-end computer systems, due to expensive cooling and packaging costs and lower reliability often associated with high levels of on-chip power dissipation.

Manuscript received February 26, 2004; revised June 20, 2004. This work was supported in part by DARPA PAC/C program under contract DAAB07-02-C-P302 and by NSF under grant no. 9988441.

K. Choi was with Samsung Electronics Corporation, KOREA. He is now with Department of Electrical Engineering-systems, University of Southern California, CA90089, USA (phone: 1-213-740-4472, fax: 1-213-740-9803, email: kihwanch@usc.edu).

R. Soma is with Department of Computer Science, University of Southern California, CA90089, USA (email: rsoma@usc.edu).

M. Pedram is with Department of Electrical Engineering, University of Southern California, CA90089, USA (email: pedram@usc.edu).

Dynamic voltage and frequency scaling (DVFS) technique has proven to be a highly effective method of achieving low power consumption while meeting the performance requirements [1]. The key idea behind DVFS techniques is to dynamically scale the supply voltage level of the CPU so as to provide “just-enough” circuit speed to process the system workload while meeting the total compute time and/or throughput constraints, and thereby, reducing the energy dissipation (which is quadratically dependent on the supply voltage level.) A number of modern microprocessors such as Intel’s XScale [2] and Transmeta’s Crusoe [3] are equipped with the DVFS functionality.

DVFS techniques may be used to reduce the energy consumption of an executed task while ensuring that the task meets its deadline. However, these techniques are not directly applicable to general-purpose operating systems because they assume that critical information about all tasks, such as the task arrival time, deadline, and workload, are known in advance. Moreover, the workload of a task is often represented by the number of CPU clock cycles required to complete the task regardless of whether the workload consists of mainly CPU-bound or memory-bound instructions. The latter information is, of course, critical in determining the idle time of the CPU.

In this paper, we propose an intra-process DVFS technique for non real-time operation in which finely tunable energy and performance trade-off can be achieved. The main idea is to lower the CPU frequency during the CPU idle times, which are, in turn, due to external memory stalls. Now if the task execution time is dominated by the memory access time, then the CPU speed can be slowed down with little impact on the total execution time. This could, however, result in potentially significant savings in energy consumption. To capture the CPU idle time at run time, several performance monitoring events, provided by performance monitoring unit (PMU) in the XScale processor, are used. The proposed technique has been implemented on an embedded system platform and actual energy savings have been calculated by current measurements in hardware. On this platform more than 70% CPU energy savings was achieved for memory-bound programs with a performance degradation of only 12%. In contrast, 15~60%

CPU energy savings was achieved for CPU-bound programs with a performance degradation of 5-20%.

The main contributions of our work are: (1) It presents one of the first actual implementations of an intra-process DVFS policy that exploits dynamic events at run time without any support from compiler or modification of the application program itself. (2) A simple, but effective, regression model is proposed to approximately determine the CPU idle time due to memory stalls by estimating the ratio of the total off-chip access time to the total on-chip computation time at runtime. (3) Evaluation of the proposed method is performed through actual hardware measurements for a number of different applications.

The remainder of this paper is organized as follows. Related work is described in Section 2. In Section 3 and 4, a new DVFS policy is presented. Details of the implementation, including both hardware and software, are described in Section 5. Experimental results and conclusions are given in Sections 6 and 7, respectively.

II. PRIOR WORK

Previous DVFS-related works may be divided into two categories based on the scaling granularity: coarse-grained and fine-grained. Coarse-grained voltage scaling is performed at the operating system (OS) or application level, whereas fine-grained voltage scaling is performed at the level of individual blocks/segments in an application task or software program. Many scheduling policies for hard real-time applications have coarse granularity. Multi-task scheduling in the OS is the focus of [4][5][6][7]. More precisely, scheduling is performed at task level by the OS so as to reduce energy consumption while meeting hard timing constraints for each task. In these coarse-grained DVFS approaches, it is assumed that the total number of CPU cycles needed to complete each task is fixed and known a priori. There are also a number of studies that implement fine-grained DVFS as part of compile-time optimization or by modifying the application program itself. In [8], an intra-task voltage scheduling technique was proposed in which the application code is divided into many segments and the worst-case execution time of each segment (which is obtained from a static timing analysis) is used to determine a suitable voltage for the next segment. In [9] a method based on a software feedback loop was proposed. In this method, a deadline for each time slot is provided. The authors calculate the operating frequency of the processor for the next time slot depending on the slack time generated in the current slot and the worst-case execution time of the next time slot. In [10], a checkpoint-based algorithm is proposed in which the scaling points are identified off-line by the compiler. In [11] and [12], compiler-assisted DVFS techniques were proposed, in which frequency is lowered in memory-bound region of a program with little performance degradation. In [13], an intra-task DVFS for multimedia application was proposed in which scaling is performed at each video frame based on the timing information given by the video server. In [14], an intra-task scheduling method for an

embedded multiprocessor system was proposed whereby tasks are dynamically scheduled based on a predefined schedule set during the compilation step. The goal is to save energy without incurring a large computational overhead at the runtime.

DVFS approaches that rely on micro-architecture or embedded hardware without any assistance from a compiler or a simulator have been reported. In [15] a microarchitecture-driven DVFS technique was proposed in which cache miss drives the voltage scaling. In [16] IPC (instruction per cycle) rate of a program execution was used to direct the voltage scaling. In [17], a voltage scaling approach, called process cruise control, was proposed, where dynamic events such as cache hit/miss ratio and memory access counts at run time from a performance monitoring unit (PMU) are used to determine the optimal frequency for a performance constraint.

In this paper, we propose a DVFS policy for non real-time application similar to the one presented in [17]. However, in our proposed DVFS approach, we use the performance events in a different way. Furthermore, our policy enables more precise control over energy-performance trade-off by using regression-based method in which performance events are used to recognize memory-bound region at runtime effectively. The proposed DVFS method can easily be extended to soft real-time applications such as multimedia processing. In such application, a large number of memory transactions take place, but at the same time, it is acceptable to miss the target deadline every now and then. For example, MPEG decoding which is one of most popular multimedia applications requires frequent memory accesses during some of its decoding steps (e.g., dithering). Simultaneously, it is allowed to miss the target frame rate for short periods of time (more precisely, an average, rather than the minimum, frame rate must be guaranteed.) Consequently, the CPU idle time due to memory transactions can be captured by using the proposed method, and the CPU voltage and frequency scaled so as to attain a significant energy saving. A preliminary version of this work appeared in [18].

III. PERFORMANCE-ENERGY TRADE-OFFS

A. Workload partitioning

Generally speaking, a task consists of a sequence of instructions to be performed. The execution time of a task is the sum of latencies of all instructions in the task. The instruction latencies can in turn be classified as on-chip latencies (data dependency, cache hit, branch prediction) or off-chip latencies (memory latency, PCI latency). The on-chip latencies are caused by events that occur inside the CPU. They are synchronized to the internal clock and may linearly be reduced by increasing the CPU frequency. The off-chip latencies, on the other hand, are independent of the internal frequency and are thus not affected by changing the CPU frequency. Accesses to external devices such as SDRAM and PCI peripheral devices are synchronized to the bus clock, which is independent of the CPU frequency.

Definition 1: *on-chip workload*, W_{onchip} is the number of

CPU clock cycles required to perform instructions which cause on-chip latencies.

Definition 2: *off-chip workload*, $W_{offchip}$, is the number of external bus clock cycles during off-chip accesses. Note that during these accesses, the CPU is stalled and waiting for transactions outside the CPU to complete.

Let T_{onchip} and $T_{offchip}$ denote the required time to process W_{onchip} and $W_{offchip}$. We have:

$$T_{onchip} = \frac{W_{onchip}}{f_{cpu}}, \quad T_{offchip} = \frac{W_{offchip}}{f_{mem}} \quad (1)$$

where f_{cpu} and f_{mem} denote the *current* clock frequency of the CPU and the clock frequency of the off-chip bus.

The total execution time of a program, T , is clearly written as:

$$T = T_{onchip} + T_{offchip} \quad (2)$$

Notice that this breakdown of the total execution is not exact when the target processor supports out-of-order execution whereby instructions after the instruction that caused an off-chip access may be executed during the off-chip access. In such a case, T_{onchip} and $T_{offchip}$ can overlap. However, in practice, the error introduced in this way is quite small considering that the memory access time is about two orders of magnitude greater than the instruction execution time. Therefore, out-of-order execution does not cause a large error in equation (2).

T_{onchip} and $T_{offchip}$ can be represented in terms of the CPU frequency and the amount of workload which is the CPI multiplied by the number of instructions being executed [19] as follows:

$$T_{onchip} = \frac{\sum_{i=1}^n CPI_{onchip}^i}{f_{cpu}} = \frac{n \cdot CPI_{onchip}^{avg}}{f_{cpu}}, \quad T_{offchip} = \frac{\sum_{j=1}^m CPI_{offchip}^j}{f_{mem}} = T - T_{onchip} \quad (3)$$

where n is the total number of instructions in the instruction stream, m is the number of off-chip accesses in that stream, CPI_{onchip}^i denotes the number of CPU clock cycles for the i^{th} instruction, $CPI_{offchip}^j$ denotes the number of memory clock cycles for the j^{th} off-chip access, and CPI_{onchip}^{avg} denotes the average on-chip CPI.

The CPU frequency for a task can be calculated differently depending on temporal distribution of W_{onchip} and $W_{offchip}$ as well as values of W_{onchip} and $W_{offchip}$. Consider a task which has W_{onchip} comprising of W_{onchip}^1 and W_{onchip}^3 and $W_{offchip}$ comprising of $W_{offchip}^2$ and $W_{offchip}^4$. Furthermore, assume that the four subtasks are executed in the order shown in Figure 1.

Then, there are two different scenarios, (I) and (II), according to whether we know the complete execution sequence of W_{onchip} and $W_{offchip}$ or not. In scenario (I), it is assumed that we know the temporal execution sequence of subtasks inside the task, i.e. $W_{onchip}^1 \rightarrow W_{offchip}^2 \rightarrow W_{onchip}^3 \rightarrow W_{offchip}^4$, whereas, this information is not available in scenario (II).

Now, the CPU frequencies for $W_{offchip}^2$ and $W_{offchip}^4$ can be set the minimum possible level in scenario (I) while it is not

possible to assign the minimum CPU frequency for $W_{offchip}$ in scenario (II). Thus, not surprisingly, more CPU energy can be saved in scenario (I) compared to scenario (II). More precisely, the CPU clock frequencies for the two scenarios are given next:

$$\text{scenario (I)} : f_{cpu}^{onchip} = \frac{W_{onchip}^1 + W_{onchip}^3}{D - \left(\frac{W_{offchip}^2 + W_{offchip}^4}{f_{mem}} \right)}, \quad f_{cpu}^{offchip} = f_{cpu}^{min} \quad (4)$$

$$\text{scenario (II)} : f_{cpu}^{onchip} = f_{cpu}^{offchip} = \frac{W_{onchip}^1 + W_{onchip}^3}{D - \left(\frac{W_{offchip}^2 + W_{offchip}^4}{f_{mem}} \right)} \quad (5)$$

where W_{onchip}^i ($W_{offchip}^i$) is on-chip (off-chip) workload of the i^{th} subtask, D is the deadline, f_{cpu}^{min} is the minimum CPU frequency, and f_{cpu}^{onchip} ($f_{cpu}^{offchip}$) is CPU frequency during the period of time that we are servicing on-chip (off-chip) accesses.

Notice that to set the minimum frequency during off-chip accesses in scenario (I), $W_{offchip}^2$ and $W_{offchip}^4$ should be sufficiently large compared to the frequency and voltage scaling overhead in actual hardware. For example, if a task results in a large number of small $W_{offchip}$'s that are scattered over the whole execution time of the task, then the CPU frequency for such a case is calculated as in scenario (II) even when the execution sequence is known.

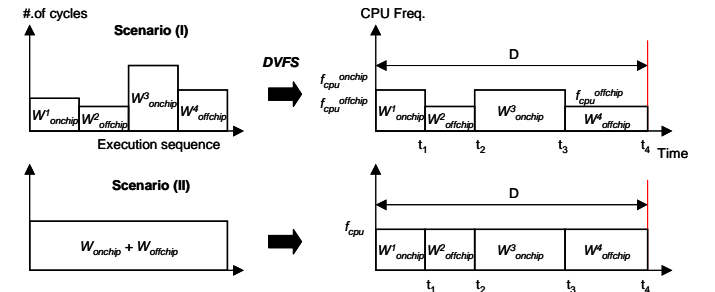


Fig. 1. DVFS with detailed knowledge about subtasks and their relative order and workload requirement (scenario I) and without this information (scenario II)

B. Performance degradation and energy saving

When the CPU frequency is changed for executing a task, the variation in the execution time, T , is solely dependent upon W_{onchip} of the task, because f_{mem} is independent of the f_{cpu} and is not scaled.

$$\frac{\Delta T}{\Delta f} = \frac{\Delta T_{onchip}}{\Delta f}, \quad \frac{\Delta T_{offchip}}{\Delta f} \approx 0 \quad (6)$$

The increased execution time of a program due to lowered clock frequency represents the performance loss (PF_{loss}), which is defined as follows:

$$PF_{loss} = \frac{(T_n - T_{f_{max}})}{T_{f_{max}}} \quad (7)$$

where f_{max} is the maximum frequency of the CPU, f_n is a frequency lower than f_{max} , T_{f_n} and $T_{f_{max}}$ are the total task execution times at CPU frequencies of f_n and f_{max} , respectively.

For a given program, different ratios of T_{onchip} and $T_{offchip}$ result in very different PF_{loss} over CPU frequencies. Figure 2 provides energy-performance trade-offs for various

applications. For example, in case of the “crc” and “djpeg”, lowering frequency introduces significant performance loss compared to other tasks implying that these programs are CPU-bound (i.e., $T_{onchip} \gg T_{offchip}$). On the contrary, it is known that “fgrep” and “qsort” are *memory-bound* (i.e., $T_{onchip} \ll T_{offchip}$) by observing little performance degradation with lowered frequency. Based on these observations, we conclude that the ratio of T_{onchip} to $T_{offchip}$ for a program is very important to the degree of energy saving and performance penalty attained by DVFS techniques.

Definition 3: The β value of a program is defined as the ratio $T_{offchip}/T_{onchip}$ for that program.

β represents the degree of potential energy saving because the larger β is, the more CPU energy saving can be achieved by a DVFS technique. Consequently, we need accurate information about β in order to sustain an effective DVFS technique. In regards to the two scenarios described in the previous section, because it is impractical to obtain the exact temporal distribution of W_{onchip} and $W_{offchip}$ at run time, we calculate the optimal CPU frequency as in scenario (II).

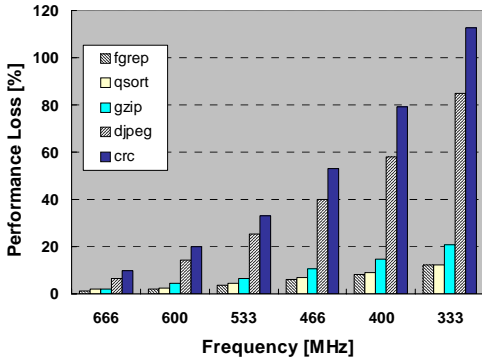


Fig. 2. Performance loss changes according to CPU frequency

From equations (1) and (7), the optimal frequency, f_{target} , for a given PF_{loss} value is calculated as follows:

$$f_{target} = \frac{f_{max}}{1 + PF_{loss} \cdot \left[1 + \beta \cdot \left(\frac{f_{max}}{f_{cpu}} \right) \right]} \quad (8)$$

As it can be seen from the above equation, f_{target} is closely related to β of a program. Consequently, accurate calculation of β is quite important to the effectiveness of our proposed DVFS approach.

C. Scaling granularity

The ideal DVFS can instantaneously change the voltage/frequency values. In reality, however, it takes time to change CPU frequency/voltage due to factors such as the internal PLL (phase lock loop) locking time and capacitances that exist in the voltage path. For the 80200 XScale processor, the latency for switching the CPU voltage/frequency is 6 μ sec at 333MHz [2]. The quantum of time for scaling the CPU frequency/voltage must be much larger than this switching overhead so that the overhead becomes negligible compared to the scaling time unit. At the same time, we would like to

minimize the overhead of the voltage/frequency scaling as far as the OS is concerned. Therefore, we use the start time of an (OS) *quantum* (approximately 50msec in Linux) used by the OS to schedule processes as DVFS decision points, that is, each time the OS invokes the scheduler to schedule processes in the next quantum, we also make a decision as to whether or not the CPU voltage/frequency is changed and if so, scale the voltage/frequency of the CPU. In addition to ease of the implementation, the advantage of using the OS quantum as a scaling unit is that the quantum is the smallest unit of time in which a process is executed atomically under a multitasking OS environment such as Linux.

D. Events monitored through the PMU on XScale

It is very difficult to calculate the exact β of a program in a static manner such as during the compilation time because on/off-chip latencies are severely affected by dynamic behavior such as cache statistics and different access overheads for different external devices. So, these unpredictable dynamic behaviors should be captured at run time. This can be achieved by using a performance-monitoring unit that is often available in modern microprocessors. In our target system, the CPU is Intel’s XScale, which supports monitoring of 20 performance events including cache hit/miss, TLB hit/miss, and number of executed instructions. The overhead for accessing PMU (read/write) is less than 1usec [17] and can be ignored. However, there is a limitation in using these events in the sense that only two events can be monitored at the same time along with the number of clock counts in a quantum (CCNT).

For our DVFS policy, we performed many experiments to figure out which events can give valuable clue about β and the following two events were proven to be most helpful based on experimental results: (i) the number of instructions being executed (INSTR) and (ii) the number of external memory accesses (MEM).

Using these two events, INSTR and MEM, along with CCNT, CPI_{onchip} can be extracted as in Figure 3. Figure 3 plots the combination of three events while executing (a) “fgrep” and (b) “gzip” applications at different frequencies from 733MHz to 333MHz at a fixed step of 66MHz. At the start of each quantum, the PMU reports the CCNT, INSTR, and MEM. From these three parameter values, we can calculate the average number of CPU cycles per instruction (CPI^{avg}) for the instruction stream as the ratio of CCNT to INSTR. Similarly, we can calculate the average number of memory accesses per instruction (MPI^{avg}) as the ratio of MEM to INSTR. The MPI value represents the degree of memory-bound of an application and quite useful when accurate numbers of clock cycles per memory access are not available. For example, the same memory instruction can cause different clock cycle numbers depending on its memory access pattern, either the access in the same row in the memory or not. In this figure, we have plotted CPI^{avg} on the y-axis and MPI^{avg} on the x-axis. Each dot in the plot represents one PMU report. From this figure, we can easily see that, at a fixed CPU clock frequency, CPI^{avg} is linearly related to MPI^{avg} as follows:

$$CPI^{avg} = b(f) \cdot MPI^{avg} + c \quad (9)$$

where $b(f)$ is frequency-dependent slope.

Notice that intercept c is equal to the average on-chip CPI, CPI_{onchip}^{avg} and is independent of frequency f . Therefore, Eq. (9) can be used to provide an accurate estimation of CPI_{onchip}^{avg} from which β can be determined from Eq. (1) and Definition 1.

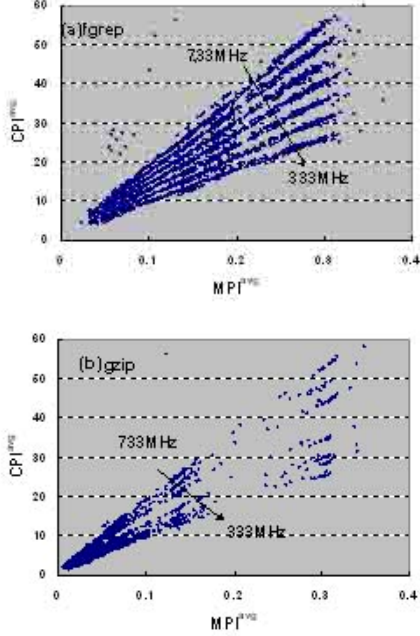


Fig. 3. Contour plots of CPI^{avg} versus MPI^{avg} for different CPU clock frequencies: (a) “fgrep” and (b) “gzip”

IV. REGRESSION-BASED FINE-GRAINED DVFS

A. Calculating β with a regression equation

In our proposed DVFS approach, monitored event values are used to estimate coefficient b and c of regression Eq. (9), and then to use this equation to predict β of a program. Voltage/frequency scaling is performed at the start of each quantum. Regression coefficients b and c are dynamically updated as explained below.

Let the linear equation for the regression be $y = b \cdot x + c$, where x and y denote MPI^{avg} and CPI^{avg} , respectively. Coefficients b and c at quantum $t \geq N$, are calculated from the last N PMU reports as follows:

$$b = \frac{N \cdot (\sum_{i=t}^{t-N+1} x_i \cdot y_i) - (\sum_{i=t}^{t-N+1} x_i) \cdot (\sum_{i=t}^{t-N+1} y_i)}{N \cdot (\sum_{i=t}^{t-N+1} x_i^2) - (\sum_{i=t}^{t-N+1} x_i)^2}, \quad c = \frac{\sum_{i=t}^{t-N+1} y_i}{N} - b \cdot \frac{\sum_{i=t}^{t-N+1} x_i}{N} \quad (10)$$

where x_i and y_i denote the MPI^{avg} and CPI^{avg} for the i^{th} quantum.

Note that we must choose N carefully since if N is chosen to be too small, we will be too sensitive to small changes in the program behavior and we may not have enough data points to do a good regression. On the other hand, if N is too large, then we may potentially filter out many important changes in the program behavior. The regression coefficients are updated at the start of every quantum. Recall that the regression equation

is maintained for each frequency because b is different for different frequencies.

The optimal frequency for the next quantum $t+1$ is calculated as follows. After quantum t , β of quantum t , β^t , is calculated as:

$$\beta^t = \frac{CPI_{onchip}^{avg,t}}{CPI_{onchip}^{avg,t}} - 1 \quad (11)$$

Once β^t is obtained, the target CPU frequency for the next quantum, f^{t+1} , is calculated from Eq. (8) with the specified PF_{loss} as follows:

$$f^{t+1} = \frac{f_{max}}{1 + PF_{loss} \cdot \left[1 + \beta^t \cdot \left(\frac{f_{max}}{f^t} \right) \right]} \quad (12)$$

B. Prediction error adjustment

We assumed that parameter β for the next quantum is the same as that for the current quantum. However, in reality, β can vary significantly even within one time quantum depending to the characteristics of the target application program (β variation tends to be higher for memory-bound applications.) The β variation is in fact due to the different off-chip latencies for the SDRAM and PCI-device accesses in our target system.

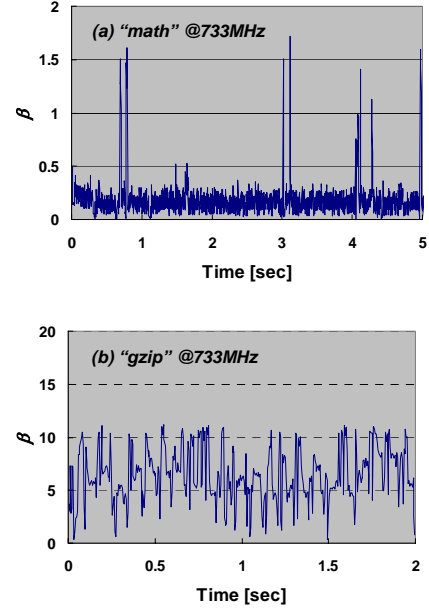


Fig. 4. β variation of applications: (a) “math” and (b) “gzip”

Figure 4 shows the actual distributions of β over time as measured at 5msec intervals during the execution of (a) “math” and (b) “gzip”. As expected, the β variation is larger (by nearly one order of magnitude) for “gzip” compared to “math”, which is a CPU-bound application. The average value, β_{avg} , and the standard deviation, σ_β , of β are calculated as follows:

$$\beta_{avg} = \frac{\sum_{i=1}^N T_{offchip}^i}{\sum_{i=1}^N T_{onchip}^i}, \quad \sigma_\beta = \sqrt{\frac{\sum_{i=1}^N (\beta_i - \beta_{avg})^2}{N-1}} \quad (13)$$

where N is the total number of quanta and β_i is the β value of

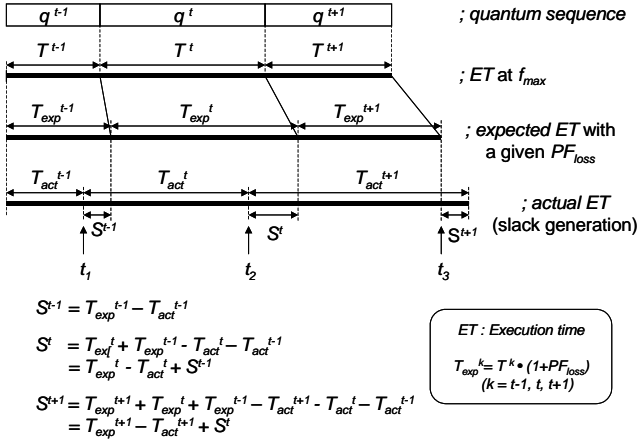
application program in the i^{th} quantum. Note that β_{avg} represents the amount of potential energy saving, i.e., the higher the β_{avg} , the higher the potential energy saving under a given timing constraint, whereas σ_β captures the degree of difficulty in achieving finely-controlled performance-energy tradeoffs. The β_{avg} and σ_β values are reported in Table 1.

TABLE 1.

STATISTICS FOR THE β VALUE SEEN FOR DIFFERENT APPLICATIONS AT A CPU CLOCK FREQUENCY OF 733MHZ

	math	bf	crc	djpeg	gzip	qsort	fgrep
β_{avg}	0.14	0.2	0.16	0.49	5.26	7.82	10.01
σ_β	0.21	0.64	0.56	0.45	2.80	10.94	9.44

The severe fluctuations in the β parameter, which tend to occur in memory-bound applications, may cause a large error when attempting to predict β for the next quantum based on β for the current quantum. This situation becomes worse when the quantum length is variable. For example, consider the case when a process performs an I/O operation (mostly file read/write functions.) In such a case, the CPU preempts the process; therefore, the length of the OS quantum is shortened compared to the "standard" quantum length of approximately 50msec. For example, for "gzip", the actual length of the quantum ranges from 2msec to 50msec (with an average value of 6msec), whereas it is nearly constant (with an average value of 50msec) for "math". Notice that the β prediction error is especially severe when the current quantum is quite short and the next quantum is quite long. So, we modify the proposed technique in order to handle the error in predicting β for the next quantum. The modification is shown in Figure 5, which depicts three consecutive quanta, q^{t-1} , q^t , and q^{t+1} , each with a distinct β value and quantum lengths T_{act}^{t-1} , T_{act}^t , and T_{act}^{t+1} . For the specified PF_{loss} , the expected execution time is denoted by T_{exp}^{t-1} , T_{exp}^t , and T_{exp}^{t+1} , respectively. Voltage/frequency scaling for q^t , q^{t+1} , and q^{t+2} is performed at t_1 , t_2 , and t_3 , respectively.

Fig. 5. Compensating for the error due to misprediction of β

When a frequency is chosen for the next quantum, there may exist some (positive or negative) slack time (i.e., the difference between T_{exp}^* and T_{act}^* .) These slack times come about due to the misprediction of β for the next quantum. With a positive (negative) slack, the frequency for the next quantum should be

made smaller (larger) compared to the case of zero slack. For example, at time t_2 , the actual execution time until t_2 is $(T_{\text{act}}^{t-1} + T_{\text{act}}^t)$ which is less than the expected time $(T_{\text{exp}}^{t-1} + T_{\text{exp}}^t)$, so there is a positive slack time $S^t = T_{\text{exp}}^t - T_{\text{act}}^t + S^{t-1}$. If S^t is added in the calculation of the frequency for the next quantum q^{t+1} , then the error that occurred in the previous quanta can be compensated for. Eq. (12) for calculating the target frequency for next quantum is thus modified as follows:

$$f^{t+1} = \frac{f_{\text{max}}}{1 + PF_{\text{loss}} \cdot \left[1 + \left(\beta^t + \frac{S^t}{PF_{\text{loss}} \cdot T_{\text{act}}^t} \right) \cdot \left(\frac{f_{\text{max}}}{f^t} \right) \right]} \quad (14)$$

Notice that for positive (negative) slack S^t , the denominator will be larger (smaller) than the zero slack case, and hence the target frequency f^{t+1} will be smaller (larger), which is of course the desired behavior.

V. IMPLEMENTATION

We implemented the proposed policy on a high-performance XScale-based testbed, which runs Linux (v2.4.17) including following components:

- Main module: Intel 80200 XScale microprocessor; PC100 SDRAM main memory (128 MByte, 64-bit bus); XScale microprocessor system bus interface and peripheral bus interface; SDRAM controller @100MHz (Sustained 800MB/sec of SDRAM bandwidth); PCI host bridge; Integrated FLASH memory controller and integrated UART16550 controller; DMA controller and interrupt controller
- FPGA module: Xilinx VirtexE/VirtexII FPGA companion chip
- LCD module: 10.4 inch 800×600 resolution color TFT LCD panel (LG-Philips LP064V1); Xilinx XC2S150 LCD controller; 16MB SDRAM frame buffer memory
- DSP module: TI TMS320C6713 floating point DSP; 1800 MIPS/1350 MFLOPS@225MHz
- Ethernet 10/100Mbps module
- USB 2.0 host module (NEC uPD720101) supporting up to 480 Mb/s data bandwidth
- PCMCIA host module (PCI RICOH R5C475II) supporting IEEE 802.11b WLAN protocol

The Intel 80200 Xscale processor configuration is summarized in Table 2.

TABLE 2.
INTEL 80200 XSCALE PROCESSOR CONFIGURATION

Unit	Configuration
Instruction Cache	32 Kbytes, 32 ways
Data Cache	32 Kbytes, 32 ways
Mini-Data Cache	2 Kbytes, 2 ways
Branch Target Buffer	2 Kbytes, 2 ways
Instruction Memory Management Unit	32 entry TLB, Full associative
Data Memory Management Unit	32 entry TLB, Full associative

The photo of the main PC Board of our target system, where XScale processor, SDRAM module, and memory controller are embedded, is shown in Figure 6.

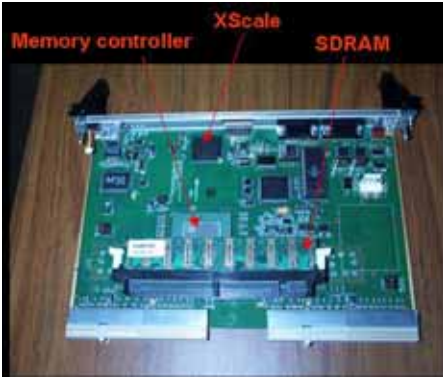


Fig. 6. Main board with the CPU, memory, and memory controller

A programmable clock multiplier (PLL) in the XScale processor generates the internal CPU clock, which can be adjusted from 200 up to 733MHz in steps of about 66 MHz with the development-board speeds only available from 333 MHz and up. The lower bound results from a constraint to the memory bus speed, which is at 100 MHz in our system. The bus speed has to be less than a third of the CPU clock speed. This would yield a minimum speed of 333 MHz. Running the system at CPU speeds slower than 333MHz causes immediate halts. The main PCB of our testbed includes an on-board variable voltage generator, which provides suitable operating voltage at each clock frequency level. A D/A converter was used as a variable operating voltage generator to control the reference input voltage to a DC-DC converter that supplies operating voltage to the CPU. Inputs to the D/A converter were generated using a customized CPLD (Complex Programmable Logic Device). When the CPU clock speed is changed, a minimum operating voltage level should be applied at each frequency to avoid a system crash due to increased gate delays. In our implementation, these minimum voltages are measured and stored in a table so that these values are automatically sent to the variable voltage generator when the clock speed changes. Voltage levels mapped to each frequency are obtained through extensive measurements and summarized in Table 3.

TABLE 3.
FREQUENCY AND VOLTAGE LEVELS IN THE SYSTEM

Frequency (MHz)	Voltage (V)
333	0.91
400	0.99
466	1.05
533	1.12
600	1.19
666	1.26
733	1.49

For the measurements, the system has a 100K samples/second data acquisition system in which the voltage drop across a precision resistor inserted between the external power line and the “design under test” (DUT) power line is used to measure the power consumption as shown in Figure 7.

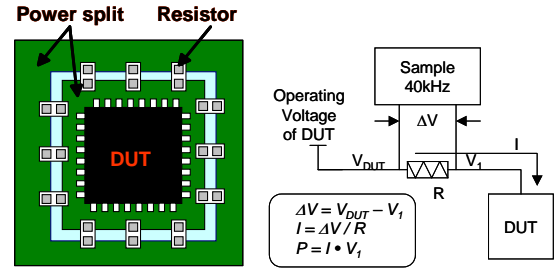


Fig. 7. Data acquisition system

As software works, we wrote a module in which the proposed policy is implemented and this module is hooked to the scheduler so that voltage scaling can occur during every context switch. Figure 8 shows the software architecture of DVFS implementation.

During the context switch, the PMU values for the previous process are read and the ideal frequency calculation for the next quantum is performed as described in section 4. A regression equation at each frequency is maintained for each process, which consists of no more than 5 long-type variables, resulting in little space overhead for implementing our DVFS policy. We measured the time overhead of our policy by using benchmark in the suite of the Lmbench [20] and found that the time overhead was about 100μsec. The original context switch time was also nearly 100 μsec. Although we almost doubled the context switch time, the overhead is still quite negligible in comparison to the quantum time of a few tens of millisecond.

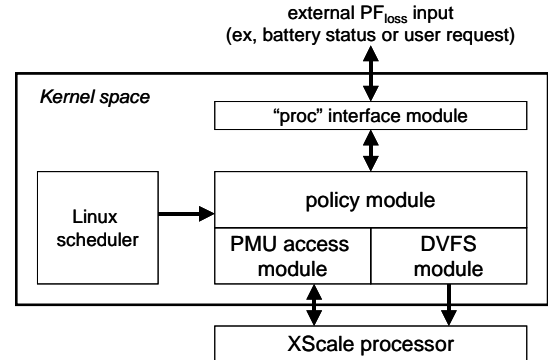


Fig. 8. Software architecture of our DVFS implementation

Our implementation supports a *proc-file* interface to the module such that the performance loss level and size of the window can be specified by writing the appropriate value to the this *proc-file*, which allows us to dynamically control the desired level of energy saving. Furthermore, the current values can be read from the *proc-file* interface. Another feature we have implemented to gain more accurate information (at the cost of higher overhead) is to measure the event values of PMU at every timer interrupt (1ms on our platform). This feature is disabled by default and is not exploited in the experimental results section.

VI. EXPERIMENTAL RESULTS

Our experiments are performed on the following

applications including two common UNIX utility programs (“gzip” and “fgrep”) and five representative benchmark programs available on the web [21]. They are summarized in Table 4. All the measurements are performed 10 times for each benchmark and the average performance loss and average energy saving values are reported. Size of the window, N , is set to 25 through exhaustive experiments. Based on the experimental results, it is found that N of 20 ~50 shows similar characteristics.

TABLE 4.
SUMMARY OF TEST APPLICATIONS

Benchmarks	Description
gzip	compressing a given input file
fgrep	searching for a given pattern in the files residing in a directory
math	floating-point calculations
bf (blowfish)	a symmetric block cipher with a variable length key from 32 to 448 bits
crc	32-bit cyclic redundancy check on a file
djpeg	decoding a jpeg image file
qsort	sorting a large array of strings in ascending order

Figure 9 represents the measured performance degradation with target performance loss ranging from 5% to 20% at steps of 5%. As seen in this figure, we obtained actual performance loss values very close to the target values for all programs (i.e., actual within 2% of the target) except for “fgrep” and “qsort” programs, which are memory-bound and PF_{loss} of these are saturated to ~12%, corresponding to data in Figure 2.

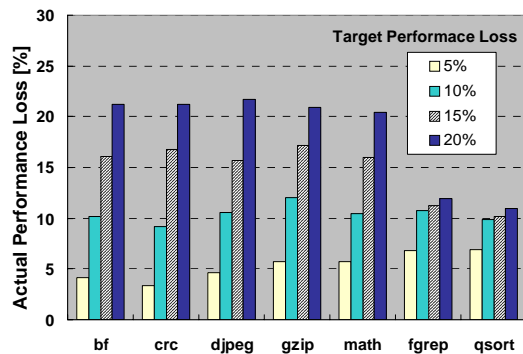
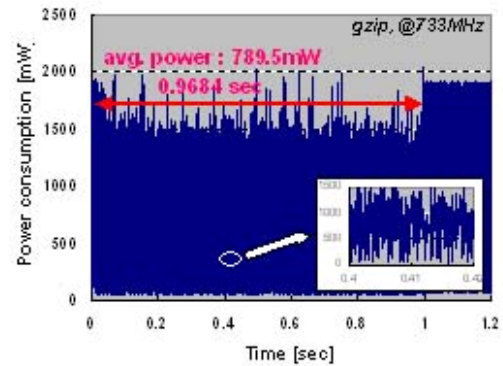
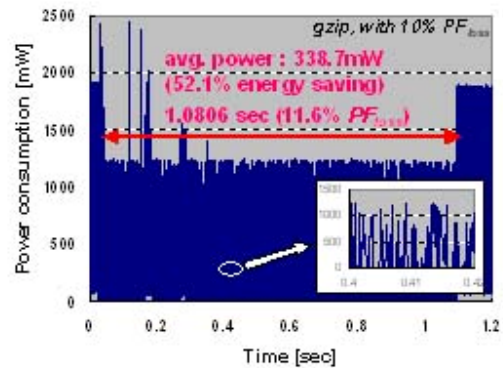


Fig. 9. Performance loss with different target values

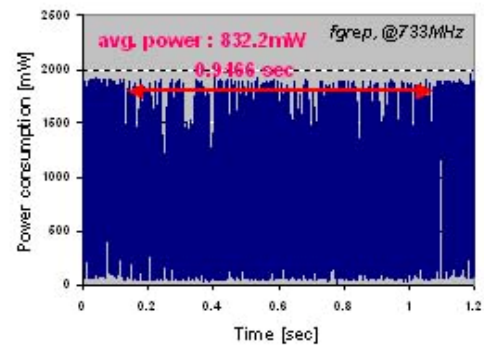
In Figure 10, actual power consumptions (including both CPU and DC-DC converter power) for two cases: (i) without DVFS and (ii) with DVFS are reported when running “gzip” and “fgrep”. In case (a) and (c), two programs are run at the maximum frequency (733MHz) and 10% target PF_{loss} is given consistent with case (b) and (d). By applying the proposed policy, 52.1% of the CPU energy is saved at the cost of 11.6% performance loss for “gzip”, whereas 77.6% of CPU energy saving with 10.3% performance loss is achieved for “fgrep”.



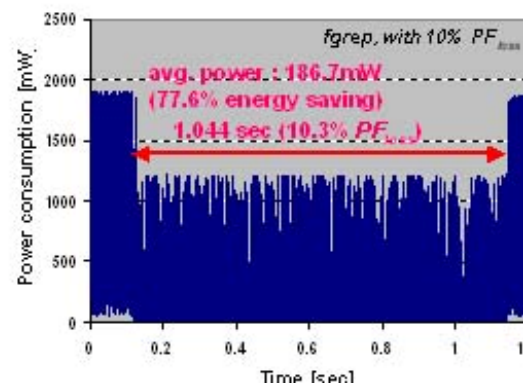
(a) “gzip”: without DVFS - at maximum frequency



(b) “gzip”: with DVFS - at a 10% performance loss constraint



(c) “fgrep”: without DVFS - at maximum frequency



(d) “fgrep”: with DVFS - at a 10% performance loss constraint

Fig. 10. CPU power consumption of with/without DVFS

Measured energy savings for all benchmarks appear in Figure 11. From these measurements, we conclude that a CPU

energy saving of more than 70% is achieved for memory-bound applications (“fgrep” and “qsort”) with about 10% performance loss. The energy saving saturates after that, i.e., we cannot increase the amount of energy savings by tolerating a larger performance loss value. For CPU-bound applications, the degree of energy saving is smaller, but our approach allows a finely tuned energy-performance tradeoff. For example, in the case of “djpeg” program, we obtain a 42% CPU energy saving with a 20% performance loss constraint or a 26% energy saving with a 5% performance loss constraint.

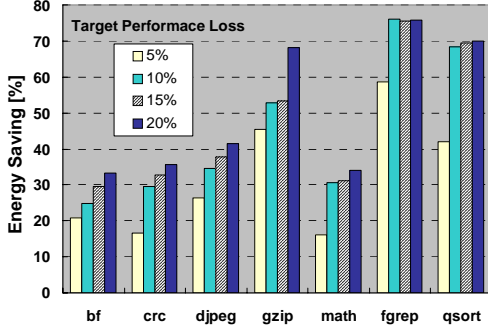


Fig. 11. CPU Energy saving for various application programs

Next, we compared our proposed DVFS method with a static approach in which the CPU frequency for an application program is calculated based on the off-line data profiling in a uni-tasking system and the calculated frequency is set during the whole execution. The percentage of T_{onchip} and $T_{offchip}$ in the total execution time T at the maximum CPU frequency (733MHz) are measured for three benchmarks, “djpeg”, “gzip”, and “qsort” and provided in Table 5.

TABLE 5.

CALCULATED CPU FREQUENCY FOR TEST APPLICATIONS AFTER PROFILING					
Bench- marks	β_{avg}	f_{opt} (calculated frequency)		f_{app} (applied frequency)	
		[MHz]		[MHz]	
		10% PF _{loss}	20% PF _{loss}	10% PF _{loss}	20% PF _{loss}
djpeg	0.49	637.9	564.7	666	600
gzip	5.26	450.8	325.5	466	333
qsort	7.82	389.5	265.2	400	333

Based on the measured T_{onchip} and $T_{offchip}$, i.e., β_{avg} , the optimal CPU frequency, f_{opt} , for each application is calculated using Eq. (8) with the target performance of 10% and 20%. Due to discreteness of the available CPU frequency in our target system, we used the closest CPU frequency, f_{app} , larger than f_{opt} and measured the CPU energy consumption. Figure 12 shows the CPU energy saving difference between our dynamic method and the static approach. As shown in this Figure, it was found that there is little difference in the energy savings (less than 5%) between the two approaches.

As the final experiment, we compared energy saving according to a scaling granularity whereby the PMU readings and the voltage/frequency scaling operation occur at intervals of 1msec, 5msec, 20msec, and 50msec, respectively. The results for “gzip” (memory-bound) and “djpeg” (cpu-bound)

applications are shown in Figure 13. We can see that too frequent a scaling, i.e., 1msec time interval, causes higher energy consumption for 5% and 10% target performance loss specs in “djpeg”. This phenomenon is due to additional overhead associated with handling the timer interrupt (~80usec). In addition, in such a case, it was observed that the external memory access count, MEM, increases due to changes in the cache content, which makes the total execution time longer, resulting in higher overall energy consumption. However, these ill effects become insignificant once the scaling intervals becomes longer than 5msec. Finally, notice that there is little difference in the CPU energy saving between the OS quantum unit (which can be different from 50msec depending on the I/O interrupt generation or task completion) and other time units.

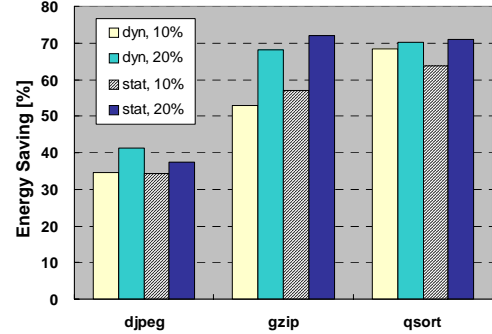


Fig. 12. Comparison of dynamic (the proposed work) vs. static approach (profiling)

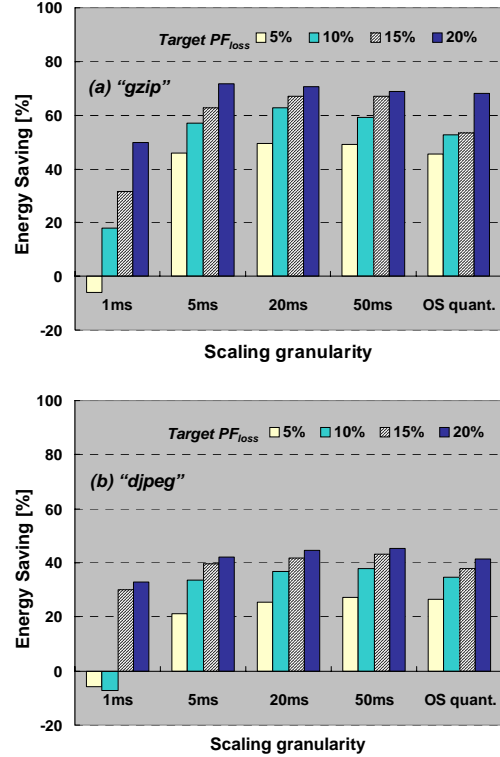


Fig. 13. Energy saving comparison according to scaling granularity (1msec, 5msec, 20msec, 50msec, and OS quantum): (a) “gzip” and (b) “djpeg”

VII. CONCLUSION

In this paper, a regression-based DVFS policy for finely tunable energy-performance trade-off was proposed and implemented on an XScale-based platform. In the proposed DVFS approach, a program execution time is decomposed into two parts: on-chip computation and off-chip access latencies. The CPU voltage/frequency is scaled based on the ratio of the on-chip and off-chip latencies for each process under a given performance degradation factor. This ratio is given by a regression equation, which is dynamically updated based on runtime event monitoring data provided by an embedded performance-monitoring unit. Through actual current measurements in hardware, we demonstrated a CPU energy consumption of saving of more than 70% for memory-bound programs with about 12% performance degradation. For CPU-bound programs, 15~60% energy saving was achieved with fine-tuned performance degradation, ranging 5% to 20%.

REFERENCES

- [1] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," *IEEE Symp. on Low Power Electronics*, 1994, pp.8-11
 - [2] "Intel 80200 Processor Based on Intel XScale Microarchitecture," <http://developer.intel.com/design/io/manuals/273411.htm>
 - [3] "Cruso SE Processor TM5800 Data Book v2.1," http://www.transmeta.com/everywhere/products/embedded/embedded_efamily.html.
 - [4] F. Yao, A. Demers, and S. Shenker, "A Scheduling model for reduced CPU energy," *IEEE Annual Foundations of Computer Science*, 1995, pp.374-382
 - [5] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proc. Int'l Symp. on Low Power Electronics and Design*, 1999, pp.197-202
 - [6] G. Quan and X. Hu, "Minimum energy fixed-priority scheduling for variable voltage processors," *Proc. Design Automation and Test in Europe*, March 2002, pp.782-787
 - [7] I. Hong, G. Qu, M. Potkonjak, and M.B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processor," *Proc. of the IEEE Real-Time Systems Symp.* December 1998, pp.178-187
 - [8] D. Shin, J. Kim, and S. Lee, "Low-energy intra-task voltage scheduling using static timing analysis," *Proc. Design Automation Conf.*, 2001, pp. 438-443.
 - [9] S. Lee and T. Sakurai, "Run-time power control scheme using software feedback loop for low-power real-time applications," *Proc. Asia-Pacific Design Automation Conf.*, 2000, pp.381-386.
 - [10] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau, "Profile-based dynamic voltage scheduling using program checkpoints in the COPPER framework," *Proc. Design Automation and Test in Europe Conference*, March 2002, pp.168-176
 - [11] C. Hsu and U. Kremer, "Compiler-directed dynamic voltage scaling for memory-bound applications," *Technical Report DCS-TR-498*, Department of Computer Science, Rutgers University, August 2002.
 - [12] C. Hsu and U. Kremer, "Single region vs. multiple regions: A comparison of different compiler-directed dynamic voltage scheduling approaches," *Proc. Workshop on Power-Aware Computer Systems*, February 2002.
 - [13] E-Y. Chung, L. Benini, G. De Micheli, "Contents provider-assisted dynamic voltage scaling for low energy multimedia applications," *Proc. IEEE Int'l Symp. On Low Power Design*, Monterey, CA, August 2002, pp. 42-47
 - [14] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, R. Lauwereins, "Energy-aware runtime scheduling for embedded multi-processor SoCs", *IEEE Design and Test of Computers*, vol. 18, No. 5, 2001, pp. 46-58
 - [15] D. Marculescu, "On the use of microarchitecture-driven dynamic voltage scaling," *Workshop on Complexity-Effective Design*, 2000.
 - [16] S. Ghiasi, J. Casmira, and D. Grunwald, "Using IPC variation in workloads with externally specified rates to reduce power consumption," *Proc. Workshop on Complexity Effective Design*, 2000.
 - [17] A. Weissel and F. Bellosa, "Process Cruise Control," *Proc. Compilers, Architectures and Synthesis for Embedded Systems*, October 2002, pp.238-246.
 - [18] K. Choi, R. Soma and M. Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-chip Access to On-chip Computation Times," *Proc. of Design Automation and Test in Europe*, Feb. 2004.
 - [19] J. Hennessy and D. Patterson, "Computer Architecture—A Quantitative Approach," 2nd, Morgan Kaufmann Publishers, 1996
 - [20] L. McVoy and C. Staelin, "Imbench: Portable Tools for Performance Analysis," *Proc. of the USENIX 1996 Technical Conf.*, January 1996, pp. 279-294
 - [21] <http://www.eecs.umich.edu/mibench>
- Kihwan Choi** (S'01) received B.S. degree in Electronics from the Hanyang University, Korea in 1992 and M.S. degree in Electronics from the Pohang University of Science and Technology (POSTECH), Korea in 1994. He joined the Low Power Design research group led by Dr. Pedram at the University of Southern California in 2000. His area of research is low-energy system design.
- Ramakrishna Soma** (S'03) received B.S. degree in Computer Science from the University of Mysore, India in 1999. He is pursuing M.S. degree in Computer Science at University of Southern California since 2003. His research interests are in software architecture and wireless sensor networks.
- Massoud Pedram** (S'88-M'90-SM'98-F'01) received a B.S. degree in Electrical Engineering from the California Institute of Technology in 1986 and M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley in 1989 and 1991, respectively. He then joined the department of Electrical Engineering-Systems at the University of Southern California where he is currently a professor. Dr. Pedram has served on the technical program committee of a number of conferences, including the Design Automation Conference (DAC), Design and Test in Europe Conference (DATE), Asia-Pacific Design Automation Conference (ASP-DAC), and International Conference on Computer Aided Design (ICCAD). He served as the Technical Co-chair and General Co-chair of the International Symposium on Low Power Electronics and Design (SLPED) in 1996 and 1997, respectively. He is the Technical Program Chair of the 2002 International Symposium on Physical Design. Dr. Pedram has published three books, 50 journal papers, and more than 130 conference papers. His research has received a number of awards including two ICCAD Best Paper Awards, a Distinguished Paper Citation from ICCAD, a DAC Best Paper Award. He is a recipient of the NSF's Young Investigator Award (a.k.a PECASE Award) (1996).
- Dr. Pedram is a Fellow of the IEEE, a member of the Board of Governors for the IEEE Circuits and systems Society, and IEEE Solid State Circuits Society Distinguished Lecturer, a board member of the ACM Interest Group on Design Automation, and an associate editor of the IEEE Transactions on Computer Aided Design and the ACM Transactions on Design Automation of Electronic Systems.
- His current work focuses on developing computer aided design methodologies and techniques for low power design, system-level dynamic power management, smart battery design and management, and integrated RT-level synthesis and physical design.