# Off-chip Latency-Driven Dynamic Voltage and Frequency Scaling for an MPEG Decoding

Kihwan Choi, Ramakrishna Soma, and Massoud Pedram
Dept. of EE-Systems, Univ. of Southern California, Los Angeles, CA 90089
{kihwanch, rsoma, pedram}@usc.edu

## Abstract

*This paper describes a dynamic voltage and frequency scaling (DVFS) technique for MPEG decoding to reduce the energy consumption using the computational workload decomposition. This technique decomposes the workload for decoding a frame into on-chip and off-chip workloads. The execution time required for the on-chip workload is CPU frequency-dependent, whereas the off-chip workload execution time does not change, regardless of the CPU frequency, resulting in the maximum energy savings by setting the minimum frequency during off-chip workload execution time, without causing any delay penalty. This workload decomposition is performed using a performance-monitoring unit (PMU) in the XScale-processor, which provides various statistics such as cache hit/miss and CPU stall, due to data dependency at run time. The on-chip workload for an incoming frame is predicted using a frame-based history so that the processor voltage and frequency can be scaled to provide the exact amount of computing power needed to decode the frame. To guarantee a quality of service (QoS) constraint, a prediction error compensation method, called inter-frame compensation, is proposed in which the on-chip workload prediction error is diffused into subsequent frames such that run time frame rates change smoothly. The proposed DVFS algorithm has been implemented on an XScale-based Testbed. Detailed current measurements on this platform demonstrate significant CPU energy savings ranging from 50% to 80% depending on the video clip.*

## Categories and Subject Descriptors

C.4 [Information systems]: Special-purpose and application-based systems.

## General Terms

Algorithms, Measurement, Experimentations

## Keywords

Low power, MPEG decoding, voltage and frequency scaling

## 1 Introduction

Demand for portable computing and communication devices has been increasing rapidly. Because portable devices are battery-operated, a design objective is to minimize the energy dissipation (to thus maximize the battery service time) without any appreciable degradation in the QoS. DVFS is a highly effective method to achieve this design goal. This is because energy consumption in CMOS VLSI circuits is quadratically proportional to the supply voltage [1]. Therefore, reducing the supply voltage can result in large energy savings. Reducing the voltage level, however, slows the circuit down. The key idea behind DVFS techniques is to perform dynamic voltage scaling so as to provide "just-enough" circuit speed to process the workload while meeting the total compute time and/or throughput constraints, thereby, reducing the energy dissipation. A number of modern microprocessors such as Intel's XScale [2] and Transmeta's Cruso [3] are equipped with the DVFS function.

Many DVFS techniques may be used to reduce the energy consumption of an executed task while ensuring that the task meets its deadline in real-time scenarios [4][5][6][7][8]. However, all of these techniques assume that critical information about all tasks, such as task arrival time, deadline, workload, and worst-case execution time (WCET) are known in advance. These assumptions are not directly applicable to the case when a task's workload exhibits a large variability. An archetypal example of such a task is MPEG decoding in which the computational workload fluctuates greatly depending on the frame type. Moreover, in all of the previous DVFS approaches for real-time applications, either hard real-time or soft real-time, 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 present a DVFS technique for low power MPEG decoding in which CPU energy savings is maximized through workload partitioning into either CPU-bound or memory-bound workload. The intuition for workload partitioning is that memory is asynchronous with the processor and often has its own clock. More precisely, we propose to lower the CPU frequency during the CPU idle times, which are in turn due to external memory stalls. To capture the CPU idle time at run time, the performance-monitoring unit (PMU) in the Intel's XScale processor is used. In addition, in attempt to guarantee a user-specified QoS for the video playback, we describe a method, called "inter-frame compensation", in which the frame rate fluctuation, due to workload prediction error, is effectively localized to a small number of subsequent frames.

The proposed DVFS technique has been implemented on an XScale-based embedded system platform and detailed energy savings have been obtained by actual current measurements in hardware. On this platform, a significant CPU energy savings has been achieved which ranges from 50% to 80%, depending on the test video sequence.

The main contributions of our work are: (1) The work presents one of the first actual implementations of a DVFS policy for low power MPEG decoding that exploits the different characteristic of CPU-bound and memory-bound instructions in the computational workload required to decode a frame (2) Recognition of the CPU stalls is performed dynamically by using the PMU, thus allowing us to take into account the effect of various events at run time. (3) It presents an effective error compensation method to guarantee a QoS constraint by eliminating severe frame rate fluctuations. (4) Evaluation of the proposed method is performed through actual hardware measurements for a number of different video sequences.

This research was supported in part by DARPA PAC/C program under contract DAAB07-02-C-P302 and by NSF under grant no. 9988441.



*DAC'04*, June 7-11, 2004, San Diego, California, USA

The remainder of this paper is organized as follows: Related work is described in Section 2. In Section 3, a method to separate on-chip and off-chip execution times for MPEG decoding using PMU is considered. Details of the proposed DVFS policy for an MPEG decoding are presented in Section 4. Experimental results and conclusions are given in Sections 5 and 6, respectively.

## 2 Related work

A number of researchers have applied DVFS to MPEG video decoding in order to achieve lower energy consumption [9][10][11][12][13]. In [9] and [12], DVFS with interval-based prediction is performed based on the ratio of the number of idle and busy cycles of the CPU while the MPEG stream is decoded. Although significant energy reduction has been reported, there is no guarantee that the deadline for each frame is met. A method using feedback control is proposed in [10] in which decoding time is predicted based on encoded code size of a frame. This code size prediction scheme is inaccurate, however, and may frequent miss deadlines. Furthermore, the linear prediction equation must be changed when different resolutions of the video image or different frame pixel sizes are encountered. In [13] a frame-based workload prediction is used for DVFS in which the different steps of decoding sequence are divided into frame-independent and frame-dependent parts. The prediction error for the frame-dependent part is compensated during the frame-independent part which consists of memory-intensive work and the execution time during this part can be scaled by the CPU frequency. However, this approach is inapplicable to the high performance processors such as XScale and Cruso in which external memory clock cycle is asynchronous to the CPU. In [11], the estimation of decoding time is performed in units of group of picture (GOP) that consists of 12 or 15 frames, in general. In this approach, sizes and types of the frames of an incoming GOP are observed and the time needed to decode the next GOP is estimated based on statistics of the previous GOPs. It is highly probable that severe QoS degradation may occur when the prediction is inaccurate because the same frequency (voltage) is applied for all frames in a GOP. There have been studies on using buffers in multimedia processing [14][15]. One of the most important advantages of using buffers is that no explicit frame-decode time prediction is needed, and therefore, missed deadlines due to prediction errors are avoided. These techniques, however, suffer from underflow/overflow of the finite buffer when the decoding time variation is high [14] or for improper gain of the proportional-integral controller [15]. None of the previous works on low-power MPEG decoding consider the decomposition of the computational workload, as proposed in this paper.

There are different DVFS approaches that make use of the asynchrony of memory access to the CPU clock during task execution. In [16] and [17], compiler-assisted DVFS techniques were proposed, in which frequency is lowered in the memory-bound region of a program with little performance degradation. DVFS approaches that rely on micro-architecture or embedded hardware without any assistance from a compiler or a simulator have also been reported. In [18] a microarchitecture-driven DVFS technique was proposed in which a cache miss drives the voltage scaling. In [19] the IPC (instruction per cycle) rate of a program execution was used to direct the voltage scaling. Reference [20] presented a policy to choose the optimal CPU clock frequency under a fixed performance degradation constraint (of say 10%) based on dynamic program behavior such as the number of executed instructions and memory access counts during the whole execution time using a performance-monitoring unit (PMU). The authors defined the optimal frequency domains in 2-D space comprising of points of the monitored events by exhaustive simulation, resulting in a table lookup scheme for frequency scaling. This scheme comprises of using the PMU to obtain certain run time information, which is then used as a key in the table lookup, to recover and apply the pre-computed frequency level stored in the table. Unfortunately, the technique of [20] cannot be applied to real-time applications where the performance loss constraint changes rapidly over time (for example, the performance loss constraint for an I-frame is much tighter than that for a B-frame in MPEG stream) because in that case the lookup table cannot provide the optimal frequency. To handle such a situation, frequency and voltage level calculation must be done at run time in response to the dynamically changing performance loss constraint value. In addition, an error compensation method must be put into effect in order to soften the effect of any misprediction, which was not the case in [20].

In this paper, we propose a DVFS method for MPEG decoding in which the time for memory-bound operations is accurately singled out of the whole decoding time such that CPU energy savings can be maximized under a given frame rate by setting lower CPU frequency during memory-bound operations. The calculation of memory-bound operation time is performed at run time based on the dynamic events reported by the PMU without any help from an off-line simulator or compiler.

## 3 Workload Partitioning in MPEG Decoding

### 3.1 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.

As a motivating example, Figure 1 shows the different degrees of execution time increases for two applications as CPU frequency varies.

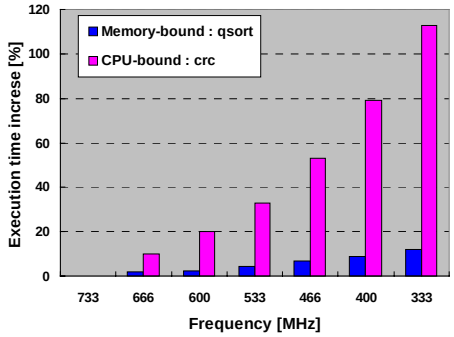



**Figure 1: Execution time changes according to CPU frequency**

For example, in the case of the “crc”, lowering frequency introduces significant performance losses compared to “qsort” implying that these programs are CPU-bound. On the contrary, it is known that “qsort” is *memory-bound* by observing little execution time increase with lowered frequency. Based on these observations, we found that, when the same amount of timing constraint is provided, a lower CPU clock frequency can be applied for memory-bound programs when compared to CPU-bound programs. This, in turn, results in higher relative energy savings for DVFS when it is applied to memory-bound programs.

**Definition 1**: *on-chip workload*, $W^{ON}$, is the number of CPU clock cycles required to perform instructions which cause on-chip latencies.

**Definition 2**: *off-chip workload*, $W^{OFF}$, 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 to complete outside the CPU.

Let $T^{ON}$ and $T^{OFF}$ denote the required time to process $W^{ON}$ and $W^{OFF}$. We have:

$$T^{ON} = \frac{W^{ON}}{f^{CPU}} = \frac{\sum_{i=1}^{n} CPI_i^{ON}}{f^{CPU}} = \frac{n \cdot CPI_{AVG}^{ON}}{f^{CPU}} \quad (1)$$

$$T^{OFF} = \frac{W^{OFF}}{f^{EXT}} = \frac{\sum_{j=1}^{m} CPI_j^{OFF}}{f^{EXT}} = \frac{m \cdot CPI_{AVG}^{OFF}}{f^{EXT}} \quad (2)$$

where $n$ is the total number of instructions in the instruction stream, $m$ is the number of off-chip accesses in that stream, $CPI_i^{ON}$ denotes the number of CPU clock cycles for the i[th] instruction due to on-chip transactions, $CPI_j^{OFF}$ denotes the number of memory clock cycles for the j[th] off-chip access, $CPI_{AVG}{}^{ON}$ and $CPI_{AVG}{}^{OFF}$ denote the average on-chip and off-chip CPI, $f^{CPU}$ and $f^{EXT}$ denote the *current* clock frequency of the CPU and the clock frequency of the off-chip bus.

Intuitively, the on-chip CPI denotes the CPI when no off-chip accesses occur. When the CPU frequency is changed for executing a task, the variation in the execution time is solely dependent upon $W^{ON}$ of the task, because $f^{EXT}$ is independent of the $f^{CPU}$ and is not scaled. The CPU frequency for a task can be calculated differently depending on temporal distribution of $W^{ON}$ and $W^{OFF}$ as well as values of $W^{ON}$ and $W^{OFF}$. Consider a task, which has $W^{ON}$ comprising of $W_1{}^{ON}$ and $W_3{}^{ON}$ and $W^{OFF}$ comprising of $W_2{}^{OFF}$ and $W_4{}^{OFF}$. Furthermore, assume that the four subtasks are executed in the order shown in Figure 2. Then, there are two different scenarios, (I) and (II), according to whether we know the complete execution sequence of $W^{ON}$ and $W^{OFF}$ or not. In scenario (I), it is assumed that we know the temporal execution sequence of subtasks inside the task, i.e. $W_1{}^{ON} \rightarrow W_2{}^{OFF} \rightarrow W_3{}^{ON} \rightarrow W_4{}^{OFF}$, whereas, this information is not available in scenario (II). Now, the CPU frequencies for $W_2{}^{OFF}$ and $W_4{}^{OFF}$ can be set to the minimum possible level in scenario (I) while it is not possible to assign the minimum CPU frequency for $W^{OFF}$ 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 as:

$$scenario\ (I)\ :\ f_{ON}^{CPU} = \frac{W_1^{ON} + W_3^{ON}}{D - \left( \frac{W_2^{OFF} + W_4^{OFF}}{f^{EXT}} \right)},\quad f_{OFF}^{CPU} = f_{MIN}^{CPU} \quad (3)$$

$$scenario\ (II)\ :\ f_{ON}^{CPU} = f_{OFF}^{CPU} = \frac{W_1^{ON} + W_3^{ON}}{D - \left( \frac{W_2^{OFF} + W_4^{OFF}}{f^{EXT}} \right)} \quad (4)$$

where $W_i^{ON}$ ($W_i^{OFF}$) denote the on-chip (off-chip) workload of the i[th] subtask, D is the deadline, $f_{MIN}{}^{CPU}$ is the minimum CPU frequency, and $f_{ON}{}^{CPU}$ ($f_{OFF}{}^{CPU}$) denote the CPU frequency during the period of time that we are servicing on-chip (off-chip) accesses.

The definition of these two scenarios is useful for MPEG decoding as will be shown in a later section because different steps in the MPEG decoding sequence can be mapped to one of these two scenarios. Notice that to set the minimum frequency during off-chip accesses in scenario (I), $W_1{}^{OFF}$ and $W_2{}^{OFF}$ should be large compared to the frequency and voltage-scaling overhead in actual hardware. For example, if a task results in a large number of small $W^{OFF}$'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.

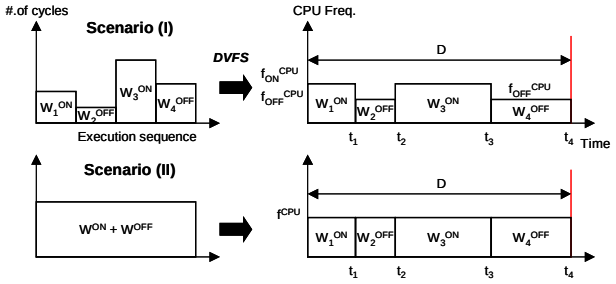



**Figure 2: DVFS with detailed knowledge of subtasks and their relative order and workload requirement (scenario I) and without this information (scenario II)**

## 3.2 Using PMU on XScale for Identifying the Off-chip Access Times

It is very difficult to get the exact $W^{ON}$ and $W^{OFF}$ of a program in a static manner such as during the compilation time. This is because on/off-chip latencies are severely affected by dynamic behavior of the program 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 [20] 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. We performed many experiments to determine which events can give valuable clues about $W^{ON}$ and $W^{OFF}$ and the following two events were proven to be the most helpful based on experimental results: (i) the number of instructions being executed (INSTR) and (ii) the number of memory accesses (MEM).

## 3.3 MPEG Decoding

Two objectives of DVFS in MPEG decoding are to maximize CPU energy savings and to guarantee a given QoS constraint such as a given frame rate. There are three frame types I-, P-, and B-frame in an MPEG video stream and each frame type results in a different workload. It takes several steps in decoding a frame as shown in Figure 3. Careful examination of what operations are performed in each step is quite helpful in partitioning the MPEG decoding workload into on-chip and off-chip. For example, the inverse discrete cosine transform (IDCT) is a CPU-intensive operation in which iterative multiplication-accumulation computations over an 8x8 array of integer or floating-point values are required, so the IDCT step is classified as $W^{ON}$, whereas the dithering and display steps are memory-intensive, requiring a frame-size data movement between the processed video stream and display frame buffer causing frequent cache misses, which can be considered as $W^{OFF}$.

To empirically confirm this observation, we played a test video clip using "*mpeg_play*" software decoder program [21] and recorded the MEM event reported by the PMU. From this experiment, we found that high MEM counts occurred during "dithering" and "display" step compared to all other steps. Furthermore, the MEM value was nearly the same for all types of frames. While "dithering" and "display" are clearly classified as operations that are intensive in terms of the off-chip accesses, it is difficult to extract $W^{OFF}$ for the remaining steps since MEM counts are scattered over repeated short loops as shown in Figure 3.

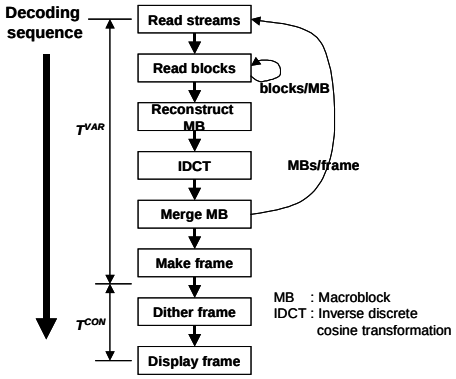



**Figure 3: MPEG decoding sequence**

The situation is complex and more closely resembles scenario (II) of Figure 2. We therefore opted to divide the whole decoding time of a frame into two parts, $T^{CON}$ and $T^{VAR}$, where $T^{CON}$ is CPU frequency-independent and comprises of the "dithering" and "display" times while $T^{VAR}$ is the elapsed time for the remaining steps, which are CPU frequency-dependent. Figure 4 shows the actual experimental results of $T^{VAR}$ and $T^{CON}$ of each frame type while changing CPU frequency from 733MHz to 333MHz. As we expected, $T^{CON}$s of all frames are independent of the CPU frequency, regardless of frame type, while $T^{VAR}$ changes according to the CPU frequency. This means that we can assign the minimum frequency during $T^{CON}$, i.e., "dithering" and "display" steps as in scenario (I).

To calculate the target CPU frequency during $T^{VAR}$, it is required to know the accurate ratio of the on-chip and off-chip times during $T^{VAR}$, which can, in turn, be collected by using dynamic events from the PMU.

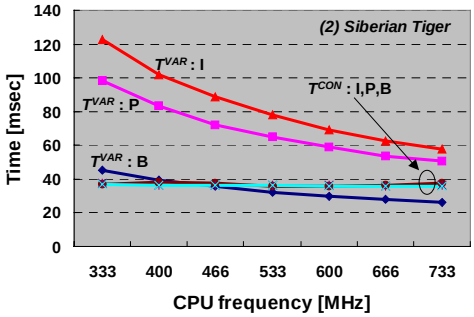



**Figure 4: Decoding time variation as a function of the CPU clock frequency**

## 4 Proposed DVFS Policy

The off-chip time, $T^{OFF}$, can be obtained by making use of the fact that it is independent of the CPU frequency. To relate a PMU event with $T^{OFF}$, we plotted many combinations of PMU events and measured $T^{VAR}$ with changing CPU frequency and found that INSTR, the number of executed instructions, can give quite accurate information about $T^{OFF}$ in $T^{VAR}$. In Figure 5, we have plotted $T^{VAR}$ on the y-axis and INSTR on the x-axis at a CPU frequency of 333MHz and at 733MHz. Each dot in the plot represents one PMU report for a B-type frame at the corresponding clock frequency. From this figure, we can see that $T^{VAR}$ for all B-frames in the test video form a line and that $T^{OFF}$ can be obtained as y-axis intercept point in a linear equation as follows:

$$T^{VAR} = \left(\frac{CPI_{AVG}^{ON}}{f^{CPU}}\right) \cdot INSTR + T^{OFF} \qquad (5)$$

Based on the equation (5), $CPI_{AVG}{}^{ON}$ is calculated as about 2.7, regardless of the CPU frequency, and $T^{OFF}$ at both frequencies converged to 7.5msec. $T^{OFF}$ for each frame type is different with B-frame having the largest $T^{OFF}$ while the I-frame has the smallest $T^{OFF}$. This observation can be justified by recalling that predictive frames (P- and B-frame) need macroblocks that have already been reconstructed and decoded in the previous I-frames; thereby, causing more off-chip access delays due to frequent data cache-misses. Finally, the proposed DVFS method is quite effective in MPEG decoding application if we consider that an MPEG video clip usually has 10 times more P- and B-frames than I-frames. In Table 1, the obtained ratios of $T^{OFF}$ and $T^{VAR}$ at 733MHz for each frame type of six different video clips are reported.

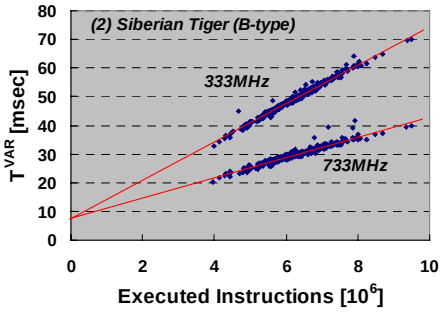



**Figure 5: Contour plots of $T^{VAR}$ versus *INSTR* for different CPU clock frequencies**

**Table 1. The ratio of $T^{VAR}$ and $T^{OFF}$ of each frame type in each video clip**

| Test video | Frame size | Frame type | | |
|---|---|---|---|---|
| | | I | P | B |
| (1) Terminator2 | 352 X 240 | 3.49 % | 11.60 % | 40.58 % |
| (2) Siberian Tiger | 320 X 240 | 7.96 % | 11.87 % | 25.74 % |
| (3) Deploy | 352 X 288 | 15.01 % | 58.01 % | 47.19 % |
| (4) Wg_wt | 304 X 224 | 10.12 % | 43.95 % | - |
| (5) Badboy2 | 480 X 208 | 20.64 % | 38.85 % | 50.76 % |
| (6) Final3 | 160 X 120 | 26.11 % | 36.80 % | 59.34 % |

Let the linear equation for the regression be $y=a*x+b$, where $x$ and $y$ denote INSTR and $T^{VAR}$ of some frame type, respectively. Coefficients $a$ and $b$ at frame $t \geq N$, are calculated from the last N PMU reports as follows:

$$a = \frac{N\cdot\left(\sum_{i=t}^{t-N+1} x_i \cdot y_i\right) - \left(\sum_{i=t}^{t-N+1} x_i\right)\cdot\left(\sum_{i=t}^{t-N+1} y_i\right)}{N\cdot\left(\sum_{i=t}^{t-N+1} x_i^2\right) - \left(\sum_{i=1}^{t-N+1} x_i\right)^2}, \quad b = \frac{\sum_{i=t}^{t-N+1} y_i}{N} - a\cdot\frac{\sum_{i=t}^{t-N+1} x_i}{N} \qquad (6)$$

The regression coefficients are updated at the end of every frame. Recall that the regression equation is maintained for each frame type because MEM varies for different frame types, resulting in different execution times for off-chip accesses.

For varying $T^{ON}$ of each frame, we maintained a moving-average of the last M INSTRs for each frame type (three averages, one per frame type). Here, M can be the same as N, that is, the number of data for the regression equation. The expected decoding time for an incoming frame under a given frame rate, R, is thus determined based on the following: the moving average of INSTR and $CPI_{AVG}{}^{ON}$ from the regression equation for on-chip latency, the y-axis intercept of the regressed equation for off-chip latency, $T_{EXP}{}^{OFF}$, and constant $T^{CON}$ which is easily obtained after decoding the first frame for a given video clip. Then, the CPU frequency for t+1[th] frame, $f_{t+1}{}^{CPU}$ is calculated as:

$$f_{t+1}^{CPU} = \frac{INSTR_{t+1}^{EXP}\cdot CPI_{AVG}^{ON}}{D - T^{CON} - T_{EXP}^{OFF}} \qquad (7)$$

where $INSTR_{t+1}{}^{EXP}$ is the average of INSTR (until the t[th] frame) of the frame type that matches frame type at time t+1.

In MPEG decoding, meeting a QoS constraint such as a given frame rate is quite important. In fact, the proposed DVFS method is based on the prediction for on-chip and off-chip times for a frame. This kind of prediction may not be perfect when each frame exhibits severe variation in the computational workload such that target frame rate cannot be maintained. So, a method that can compensate for the prediction error and effectively maintain the user-specified QoS is required.

There is a commonly used technique in video rendering called *error diffusion* [22] in which the quantization error of previously quantized pixel is filtered and distributed forward to unquantized pixels in the neighborhood such that a smooth image can be achieved. This same idea can be used to eliminate severe fluctuations in frame rate due to prediction error. In inter-frame compensation methods, the amount of error is *diffused* over the subsequent frames and the CPU frequencies for the following frames are calculated by considering not only their own predicted decoding times, but also accounting for the timing slack that occurred due to the imperfect prediction in the previous frames. This *error diffusion* makes the prediction error *localized* into a small number of neighboring frames, thereby, it can effectively compensated for by decreasing (increasing) the CPU frequency in case of over-prediction (under-prediction), resulting in soft and stable variation in the frame rate. In some way, and indirectly, the proposed inter-frame compensation method is analogous to considering "excess cycles" from the previous time slots in interval-based workload prediction techniques [23].

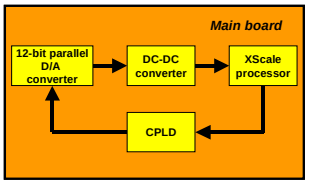
Adopting inter-frame compensation, the equation (7) is modified as follows;

$$f_{t+1}^{CPU} = \frac{INSTR_{t+1}^{EXP} \cdot CPI_{AVG}^{ON}}{D - T^{CON} - T_{EXP}^{OFF} + T_t^{SLACK}} \qquad (8)$$

where $T_t^{SLACK}$ is the time difference between D and actually elapsed time expended on decoding the $t^{th}$ frame.

## 5 Experimental Results

We implemented the proposed DVFS technique, called OL-DVFS which stands for off-chip latency driven DVFS) for MPEG decoding with on-chip vs. off-chip workload partitioning on an XScale-based system which includes an on-board variable voltage generator to generate a suitable CPU voltage at each frequency level. The block diagram of the XScale-based system as well as the allowed CPU clock frequencies with the corresponding minimum voltage levels are shown in Figure 6. Sizes of window, N and M, are set to 25 through exhaustive experiments. For the actual measurement, a data acquisition system (DAQ) with a sampling rate up to 40 KHz is used.

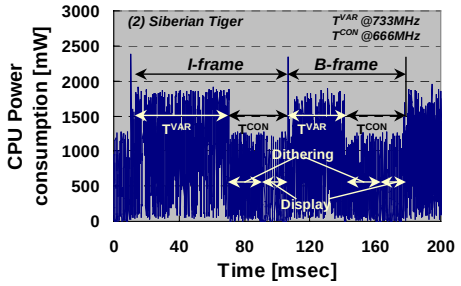



| Freq. [MHz] | Volt. [V] |
|---|---|
| 333 | 0.91 |
| 400 | 0.99 |
| 466 | 1.05 |
| 533 | 1.12 |
| 600 | 1.19 |
| 666 | 1.26 |
| 733 | 1.49 |

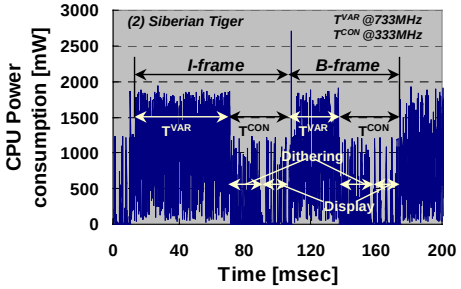
**Figure 6: Block diagram of XScale-based system**

Figure 7 depicts the CPU power consumption while decoding an I-frame followed by a B-frame in which two different frequencies are set during $T^{CON}$ (a) 666MHz and (b) 333MHz. A 733MHz is used for $T^{VAR}$. As mentioned in the previous section, $T^{CON}$, which contains the off-chip access latencies during "dithering" and "display", does not change with the CPU frequency, i.e., it remains at 37msec at both frequencies. The average power consumption during $T^{CON}$ is significantly reduced from 510mW to 186mW (64% reduction) as a result of voltage scaling.

We measured the actual CPU power consumptions while playing back six test video clips on the XScale-based system with the proposed DVFS method (OL-DVFS) and compared the results with the case of conventional DVFS without workload partitioning (CON-DVFS). The proposed inter-frame compensation is used for both OL-DVFS and CON-DVFS. CON-DVFS refers to state-of-the-art work prior to OL-DVFS and comprises of the following: The computational workload (i.e., the number of CPU clock cycles needed to decode the frame) is calculated as the elapsed total decoding time divided by the current CPU frequency. Voltage and frequency scaling is done as a function of this calculated workload.




**(a) $T^{VAR}$ : 733MHz, $T^{CON}$ : 666MHz**




**(b) $T^{VAR}$ : 733MHz, $T^{CON}$ : 333MHz**

**Figure 7: Decoding time and power consumption at different CPU frequencies and voltage levels**

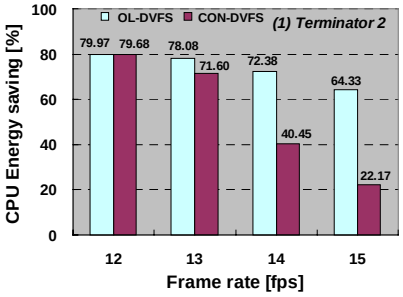



**Figure 8: CPU energy savings using proposed DVFS**

Figure 8 shows the CPU energy savings of a test video for both OL-DVFS and CON-DVFS compared to no DVFS. As we can see, the OL-DVFS method enables much higher energy savings as the frame rate becomes higher compared to CON-DVFS. Results for other test videos are summarized in Table 2, demonstrating a CPU energy savings ranging from 50% to 80% for various frame rates.

We also compared the OL-DVFS method with a DVFS technique (called MIX-DVFS) that uses the minimum CPU clock frequency for $T^{CON}$ (this is similar to OL-DVFS) and a policy similar to the CON-DVFS for $T^{VAR}$. The results are reported in Figure 9. Notice that in this experiment, the minimum CPU frequency is set during $T^{CON}$ for both OL-DVFS and MIX-DVFS in order to clearly highlight the effect of considering $T^{OFF}$ during $T^{VAR}$. Inter-frame compensation is not used in this experiment for both cases. As in Figure 9, $T^{OFF}$ identification becomes more effective as the frame

rate goes higher. In particular, with off-chip latency separation during $T^{VAR}$, a 6.5% higher energy savings at a frame rate of 14 is achieved for the test video (clip 5). Finally, Figure 10 shows the effectiveness of inter-frame compensation method.

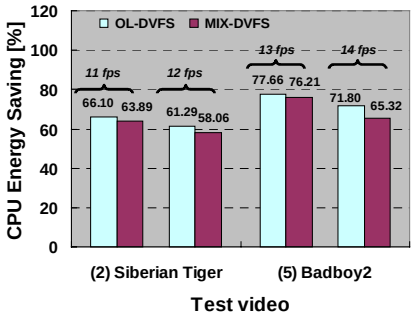



**Figure 9: CPU energy savings with off-chip latency separation during $T^{VAR}$**

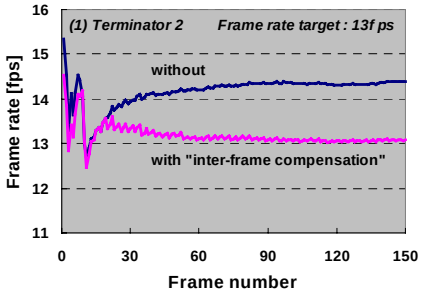



**Figure 10: Frame rate variation with the proposed DVFS**

With this compensation scheme, the run time frame rate smoothly converges to the target frame rate (here, 13 fps). Notice that the frame rate diverges from the target rate without this compensation, resulting in wasted CPU energy. The reason that the divergent rate is higher (rather than lower) than the target frame rate is that the I- and P-frames need maximum frequency to meet the deadline, and are unaware of positive timing slacks that are carried over from the previous B-frames.

## 6 Conclusions

A DVFS for MPEG decoding was proposed and implemented on the XScale-based portable system. In this DVFS, the computational workload in decoding a frame is partitioned as on-chip and off-chip workload by using a dynamic event from PMU and which results in significant CPU energy savings. To avoid QoS degradation due to misprediction of on-chip and off-chip latencies, an inter-frame compensation method was proposed in which an error occurring in a frame was diffused into a small number of subsequent frames and compensated for with a negligible fluctuation in the frame rate. On this platform the significant CPU energy savings ranges from 50% to 80% depending on the test video sequence under which various frame rates were achieved.

## References


[1] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," *IEEE Symp. On Low Power Electronics*, 1994, pp. 8-11.

[2] Developer manual: "Intel 80200 Processor Based on Intel XScale Microarchitecture," http://developer.intel.com/design/iio/manuals/273411.htm

[3] "Cruso SE Processor TM5800 Data Book v2.1," http://www.transmeta.com/everywhere/products/embedded/embedded_sefamily.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*, Monterey, CA, Aug. 1998, pp.197-202.

[6] 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.

[7] I. Hong, G. Qu, M. Potkonjak, and M.B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol.18, No.12, December 1999, pp. 1702-1714

[8] 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.

[9] T. Pering, T. Burd, and R. Broderson, "The simulation and evaluation of dynamic voltage scaling algorithms," *Proc. Int'l Symp. on Low Power Electronics and Design*, 1998, pp.76-81.

[10] J. Pouwelse, K. Langendoen, R. Lagendijk, and H. Sips, "Power-aware video decoding," *22nd Picture Coding Symp.*, Seoul, Korea, 2001.

[11] D. Son, C. Yu, and H. Kim, "Dynamic voltage scaling on MPEG decoding," *Int'l Conf. of Parallel and Distributed System*, June 2001

[12] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld, "Policies for dynamic clock scheduling," *Symp. on Operating Systems Design & Implementation*, Oct. 2000

[13] K. Choi, K. Dantu, W. Cheng, and M. Pedram, "Frame-based dynamic voltage and frequency scaling for a MPEG decoder," *Proc. Int'l Conf. on Computer Aided Design*, November 2002, pp. 732-37

[14] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," *Proc. Int'l Symp. on Low Power Electronics and Design*, Aug. 2001, pp.34-39

[15] Z. Lu, J. Lach, M. Stan, K. Skadron, "Reducing multimedia decode power using feedback control," *Proc. Int'l Conf. on Computer Design* San Jose, CA, Oct. 2003.

[16] 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, Aug. 2002.

[17] 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*, Feb. 2002.

[18] D. Marculescu, "On the use of microarchitecture-driven dynamic voltage scaling," *Proc. Workshop on Complexity-Effective Design*, June 2000.

[19] S. Ghiasi, J. Casmira, and D. Grunwald, "Using IPC variation in workloads with externally specified rates to reduce power consumption," *Workshop on Complexity Effective Design*, June 2000.

[20] A. Weissel and F. Bellosa, "Process Cruise Control," *Proc. Compilers, Architectures and Synthesis for Embedded Systems*, October 2002, pp.238-246

[21] http://bmrc.berkeley.edu/frame/research/mpeg.

[22] R. Floyd and L. Steinberg, "An adaptive algorithm for spatial grayscale," *Proc. the Society for Information Display*, 17 (2), 1976, pp. 75-77

[23] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proc. 1st Symp. on Operating Systems Design Implementation*, 1994, pp. 13-23.


**Table 2. CPU Energy saving comparison - OL: OL-DVFS, CON: CON-DVFS. (*: numbers in parenthesis are for (6))**

| fps* | (1) Terminator2 | | (2) Siberian Tiger | | (3) Deploy | | (4) Wg_wt | | (5) Badboy2 | | (6) Final3 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CON | OL | CON | OL | CON | OL | CON | OL | CON | OL | CON | OL |
| 10 | - | - | 73.15 % | 77.78 % | - | - | - | - | - | - | - | - |
| 11 (27) | 80.46 % | 80.75 % | 55.49 % | 71.39 % | - | - | - | - | - | - | 80.88 % | 82.62 % |
| 12 (28) | 79.68 % | 79.97 % | 43.39 % | 60.66 % | - | - | - | - | 79.33 % | 79.45 % | 82.04 % | 82.63 % |
| 13 (29) | 71.60 % | 78.08 % | 25.36 % | 49.54 % | - | - | 75.27 % | 77.74 % | 78.85 % | 79.48 % | 81.85 % | 81.96 % |
| 14 (30) | 40.45 % | 72.38 % | - | - | 57.94 % | 75.69 % | 60.59 % | 73.18 % | 71.34 % | 75.16 % | 81.65 % | 81.99 % |
| 15 | 22.17 % | 64.33 % | - | - | 35.53 % | 64.44 % | 41.33 % | 66.99 % | 46.99 % | 61.64 % | - | - |