Contents Provider-Assisted Dynamic Voltage Scaling for Low Energy Multimedia Applications *

Eui-Young Chung
CSL Stanford University
eychung@stanford.edu

Luca Benini
DEIS, University of Bologna
lbenini@deis.unibo.it

Giovanni De Micheli
CSL Stanford University
nanni@stanford.edu

ABSTRACT

This paper presents a new concept of DVS (*Dynamic Voltage Scaling*) for multimedia applications. Many multimedia applications have a periodic property, but each period shows a large variation in terms of its execution time. Exact estimation of such variation is a crucial factor for low energy software execution with DVS technique. Previous DVS techniques focused only on end users (client sites) and their quality heavily depends on the accurateness of the worst case execution time estimation. This paper proposes that contents providers (server sites) supply the information of the execution time variations in addition to the content itself. This makes it possible to perform DVS independent to worst case execution time estimation. The extra work required to the contents provider for this purpose is fully compensated by the benefits for the end users because single content is often provided to many users. Experimental results show that our method greatly reduces the energy consumption of client systems compared to previous DVS techniques.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms

Algorithms, Management

Keywords

DVS(Dynamic Voltage Scaling), contents provider, low-power, worst case execution time, characterization, multimedia

1. INTRODUCTION

Energy consumption has become one of the most important constraints on modern system design, especially for portable and battery powered embedded system. Processor-based architecture is one of the common choices in portable embedded system design due to its flexibility and time-to-market constraint. In this architecture, the most critical component in energy consumption is the processor because a large fraction of overall computations is performed by the programs running on the processor.

DVS (*Dynamic Voltage Scaling*) is one of the most promising techniques to reduce the processor energy consumption by adjusting the clock frequency and/or supply voltage level, while meeting the required throughput constraint. DVS basically exploits the idle interval (slack time) of the processor for energy reduction, thus the variation of idle interval critically affects the effectiveness of DVS algorithm. In practice, the idle interval depends on not only the application program running on the processor, but also the type of workload processed by the application program.

The multimedia application programs such as MPEG decoder, one of the most common programs executed on portable embedded systems, show the large variation of idle interval depending on the characteristic of the workload. For example, MPEG decoder periodically processes a picture frame, but the processing time of each frame varies depending on the frame type, picture size and so on. For example, the worst case, P-type frame decoding takes 1.6 times longer than the worst case I-type frame decoding and 3.7 times longer than the best case P-type decoding in [8].

Previous work on DVS can be classified into two categories. The first category considered hard real-time constraint applications for DVS, while soft real-time constraint applications were considered in the second category.

The first generation of hard real-time DVS techniques mostly focused on OS level multi-task scheduling problem for energy reduction [18, 9, 11]. In these methods, they assume that each task has a fixed amount of execution time regardless of the workload characteristics, therefore the full potential of energy reduction cannot be exploited when the execution time of each task varies significantly. This problem was partially solved by enhancing the fixed priority scheduling algorithm to consider both best case execution time and worst case execution time of each task in [15].

While these approaches controlled the clock frequency and supply voltage in task level (appropriate for OS level implementation), the second generation hard real-time DVS techniques proposed the finer grain control of clock frequency and supply voltage (appropriate for application program level implementation) to exploit the variation of slack time more efficiently. In [10], the *timeslot* concept was introduced. They sliced a task into several pieces called *timeslots*, which enables to adjust the processor speed (clock frequency and supply voltage) in the middle of a task execution. Another technique in this category is the compiler assisted voltage scheduling technique proposed in [16]. This technique first builds the control flow graph of the given program and identifies the worst case execution time path using static timing analysis. Then, the processor can be slowed down when it executes non timing-critical paths of the control flow graph. They showed the applicability of this technique only for single procedure, but inter procedural analysis is required when multiple procedures are involved.

Recently, the second category, soft real-time DVS techniques, has been received large attention to achieve higher

*This research was supported in part by GSRC and NSF under contract CCR-9901190.

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

degree of energy reduction [5, 12].They adopt idle length prediction schemes and select optimal voltage and frequency pair based on the prediction result. Most of the techniques in this category made a prediction based on the ratio of idle time to busy time. But these techniques often miss the deadline due to the inaccuracy of the prediction.

The most critical common assumption of previous techniques is that the worst case execution time of a task (timeslot or path in a control flow graph) is known a priori or predicted accurately. Hard real-time DVS techniques claimed that the worst case execution time could be obtained by profiling (simulation, measurement or static timing analysis). Namely, they defined single worst case execution time for each application. But in some application programs like MPEG decoder, the worst case execution time shows a large variation depending on the video clips (workloads). Soft real-time DVS techniques also have the problem of prediction quality, especially when the workload has the non-stationary property in the ratio of idle time to busy time.

In this paper, we propose a new concept of DVS for MPEG decoder to overcome such inefficiency with the help of contents provider. Our method is a soft real-time DVS technique based on the prediction, but differs from the previous soft real-time DVS techniques because the information used for prediction is supplied by contents providers, which guarantees the high quality of prediction even when the workload shows the non-stationarity. The contents provider supplies the decoding time information of each frame in conjunction with the video clips. The timing information should be architecture independent to consider various types of architectures in client sites. The MPEG decoder in client site performs DVS by translating the architecture independent timing information into architecture dependent timing information. Timing characterization, the extra work required to the contents provider is compensated by the energy reduction in client sites because it is typical that single content is provided to more than thousands of users.

2. MOTIVATION

2.1 Multimedia streaming environment

In this paper, we consider the real-time MPEG video streaming environment, *i.e.* real-time transmission of stored data. Typical MPEG streaming environment consists of two parts - servers and clients as shown in Figure 1. In download mode (or off-line mode) MPEG environment, the server is replaced by fully downloaded video clip or DVD title and data is read from the disk instead of the network. The major issues in this streaming environment can be classified into six areas - video compression, application-layer QOS control, continuous media distribution services, streaming servers, media synchronization mechanisms, and protocols for streaming media [17]. Some of previous techniques targeting multimedia environment regarded the energy reduction as a QOS problem. Thus, they requested the server system to help the energy reduction of the client systems. For example, data fidelity such as picture size is controlled depending on the remaining battery capacity [4].

On the other hand, many previous DVS techniques only have focused on the client site and the biased focus causes the inaccurate worst case frame decoding time estimation because client site has no knowledge of future frames to be decoded. The environment shown in Figure 1 does not impose any timing constraints on the encoding process, which

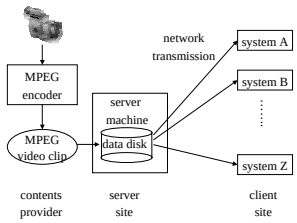

Figure 1: MPEG streaming environment

MPEG clips	size	frame decoding time (ms)			variance
		best	average	worst	
hubble	80x60	7.382	8.431	8.902	0.798
joel	80x60	5.526	5.643	5.731	0.660
klein	384x288	20.624	32.506	36.881	26.695
cindy	80x60	1.255	2.702	3.898	1.325
cindy	160x120	4.874	8.282	11.842	5.043
cindy	320x240	18.987	26.854	33.315	15.805

Table 1: Frame decoding time variation in MPEG decoder measured by strong-ARM simulator

allows the contents provider to characterize the decoding time of each frame for a given video clip. If there exists only single system architecture in client site, the timing characterization is a trivial problem because the contents provider can characterize the decoding time of each frame based on the same system architecture. In this case, the contents provider can supply not only the actual worst case frame decoding time for a given video clip, but also the exact decoding time of each frame. Thus, DVS techniques can fully exploit the slack time because MPEG decoder has the perfect knowledge of the decoding time of each frame a priori. Due to the large volume of requests for single video clip, the extra work (timing characterization) required to the contents provider can be fully compensated by the energy saving of numerous clients.

2.2 Frame decoding time variation

To illustrate our basic idea, we first show the variation of frame decoding time of a MPEG decoder in Table 1 by running MPEG decoder on strong ARM simulator. Table 1 shows two important execution time variations in MPEG frame decoding procedure.

First, the execution time variation of frame decoding in single MPEG video clip is very large. It means that DVS technique can more efficiently exploit idle intervals if we can track the execution time variation over the frame execution. Most of previous approaches have focused on exploiting the slack time occurred due to this variation.

Second, the worst case execution time of each video clip varies more significantly. One of the major reasons for such variation is the picture size (*horizontal size* × *vertical size*) because picture size affects the number of macro blocks which eventually controls the number of iterations for many computationally expensive loop constructs. Even when several video clips have the same size (`hubble`, `joel`, and `cindy(80x60)`) the worst case execution time varies significantly due to other encoding parameter values. For this reason, the worst case execution time should be decided depending on the video clip to be decoded for more efficient DVS. However, previous approaches considered single worst case execution time without the consideration of the workload characteristics. In other words, they decided single worst case execution time for a MPEG decoder program. For this reason, actual worst case execution time (*i.e.* the

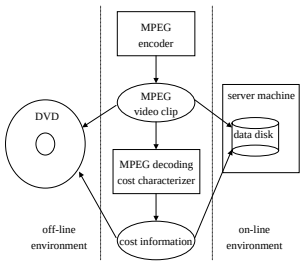

Figure 2: MPEG video clip characterization flow

worst case execution time experienced while decoding the given video clip) can be much smaller than they decided.

The following example clearly shows the importance of the exact worst case frame decoding time estimation to consider the workload characteristics.

EXAMPLE 1. *Suppose that the worst case frame decoding time of MPEG decoder is decided as* 3.898 *ms in strong ARM processor environment by profiling the program using the video clip,* `cindy (80x60)`. *In this setting, DVS technique will severely violate the required performance when MPEG decoder experiences other MPEG video clips due to the underestimation. On the other hand, suppose that the worst case frame decoding time of MPEG decoder is decided as* 36.881 *ms by profiling the program using all video clips shown in Table 1. Even though MPEG decoder is trained for all video clips, the largest worst case frame decoding time (*`klein`*) is applied to all video clips in previous approaches, which wastes large portion of slack time in other video clips. For example, when MPEG decoder experiences the video clip,* `cindy (80x60)`*,* 32.983 *ms of idle time is wasted due to the overestimated worst case frame decoding time.* ■

In practice, there exist many different system architectures in client site. For this reason, the timing information should be abstracted as architecture independent and each client system needs a method to translate the architecture independent timing information into the timing information for its own system architecture. We call the abstracted frame decoding time and translation method *decoding cost* and *decoding cost translation*, respectively. Also, we call the translated decoding cost *actual decoding cost*.

To summarize, our method requires that contents providers supply architecture independent timing information with the contents. Each system in client site translates it into the architecture specific timing information and DVS is performed based on this information.

3. OVERALL FLOW

At the server site, the decoding time of each frame for the reference system can be easily obtained by simulation or measurement. This information is only meaningful when the client system is identical to the reference system due to the architecture-dependent property. To support heterogeneous client systems, we reduce the architecture dependent property of the timing information by introducing decoding cost which is the decoding time of each frame normalized to that of the first frame (reference frame). Figure 2 shows the decoding cost characterization procedure performed by the contents provider. The decoding cost file also informs the best worst decoding costs. The reference system architecture is arbitrarily decided by the contents provider, thus each contents provider may have different reference system

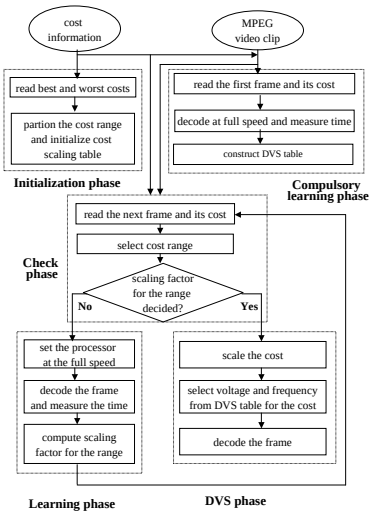

Figure 3: MPEG decoding procedure with the proposed DVS technique

architecture. However, if the client machine identifies the decoding time of the reference frame in the client machine, it is possible to approximate the decoding time of other frames using the decoding costs by assuming that the decoding time ratio among the frames are rarely sensitive to system architecture. The advantage of this abstraction is that contents providers do not need to provide the exact decoding time of each frame. Instead, they just need to provide the relative computation costs of the frames, which is possible using the computation cost estimation tool. This characterization step increases the transmission latency in streaming mode to transmit the decoding cost file before the video clip. But the latency is tolerable because the decoding cost of each frame requires only 4 bytes for floating point representation. The details of decoding cost model will be described in Section 4.

Figure 3 shows the overall procedure of MPEG decoding in cooperating with our decoding cost translation method. MPEG decoding with our technique consists of five phases.

Initially, MPEG decoder goes into `initialization phase` to initialize the cost scaling table which partitions the range from the best decoding cost to the worst decoding cost into several pieces called *cost segments*. For each cost segment, a scaling factor is assigned for translating the decoding cost into the architecture specific timing information, *i.e.* actual decoding cost. The scaling factors are identified in `learning phase` while the decoding is in progress.

After `initialization phase`, the MPEG decoder goes into `compulsory learning phase`, in which the first frame of the video clip is decoded by the processor running at full speed. At the same time, the decoding time of the first frame is measured. Using this information, the DVS table is constructed. DVS table represents the mapping relationship between actual decoding cost and appropriate clock frequency and supply voltage pair. This table is used as a lookup table in the later phase.

Next, MPEG decoder moves to `check phase`. MPEG decoder reads the decoding cost of the next frame and check the cost scaling table whether the corresponding scaling factor is identified. If the scaling factor has not been identified, MPEG decoder starts the `learning phase`. Otherwise, the `DVS phase` is started.

In `learning phase`, the frame is decoded at full speed and

Frame #	SimpleScalar		Strong ARM	
	decoding time (ms)	decoding cost	decoding time (ms)	decoding cost
1	0.221	1.000	1.227	1.000
5	0.327	1.480	1.895	1.544
10	0.653	2.955	3.920	3.194

Table 2: Decoding time and cost for `cindy(80x60)`

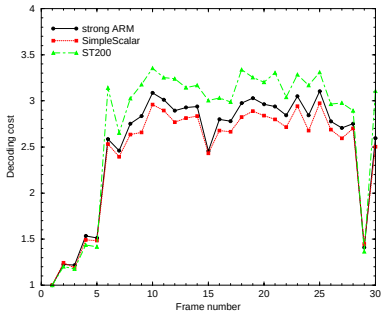

Figure 4: Decoding cost comparison on different architectures for the video clip, `cindy (80x60)`

the decoding time is measured like in `compulsory learning phase`. However, at this time, the scaling factor is computed using the decoding cost and decoding time and recorded in the cost scaling table for future reuse.

In `DVS phase` the actual decoding cost is first computed using the cost scaling table and then, DVS table is looked up to find the appropriate frequency and voltage setting. After finishing either `learning phase` or `DVS phase`, MPEG decoder returns to `check phase` to process the next frame.

Our method performs DVS only in DVS phase. But `initialization phase` and `compulsory learning phase` are executed only once during the entire decoding process. Also, for reasonable long video clips, the `DVS phase` significantly dominates learning phase visiting frequency.

4. DECODING COST CHARACTERIZATION

We simulated MPEG decoder with the video clip, `cindy (80x60)`, for three different system architectures. SimpleScalar is a super-scalar microprocessor [2], ST200 is a VLIW microprocessor [6], and strong ARM is a typical RISC microprocessor [7]. Table 2 shows the decoding time and decoding cost of three frames selected from SimpleScalar and Strong-ARM simulations. The decoding cost of each frame is its decoding time normalized to that of the frame 1. Figure 4 shows the decoding cost of the first 30 frames of the video clip, `cindy (80x60)`. In Table 2 and Figure 4, it is observed that even though the same video clip is decoded in different architectures, the decoding cost in each system varies in the similar direction. In other words, the decoding time ratio between two different frames is much less sensitive to system architecture compared to the absolute decoding time. This observation implies that decoding cost is more appropriate information rather than decoding time to consider various client system architectures for DVS.

Suppose that the reference system of the contents provider is SimpleScalar and the client systems are ST200 and strong ARM. Then, the decoding costs of two client systems, *actual decoding costs* can be obtained by scaling the decoding cost of SimpleScalar. The scaling factors can be computed by measuring the decoding times of a few frames on each client system. Although their decoding costs show similar variations, the difference at each frame is not a constant, *i.e.* they are non-linearly related. For instance, the scaling factor to compute the decoding cost of frame 1 is 1 and that of frame 10 is 1.08. The difference of scaling factors is even larger when the client system is ST200. Thus, decoding cost translation should consider the non-linear relation between decoding cost and actual decoding cost. But, this is not the duty of the contents provider and should be treated by the clients and will be discussed in detail in Section 5.

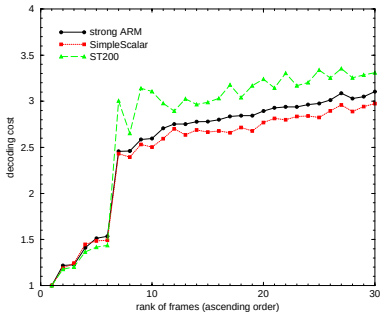

Figure 5: Sorted decoding costs of frames (`cindy (80x60)`)

5. DVS WITH DECODING COSTS

5.1 Initialization phase

The construction of cost scaling table is the main purpose of this phase. This table is used to translate the decoding cost into the actual decoding cost. Due to the non-linear relation between the decoding cost and actual decoding cost as described in Section 4, single scaling factor is not sufficient for the translation and piece-wise linear model is appropriate for the translation. Cost scaling table consists of *cost segments* and a scaling factor is determined for each *cost segment* in later phase (`learning phase`).

Figure 5 shows that piece wise linear model is appropriate for the translation. In Figure 5, the frames in Figure 4 are sorted in ascending order in terms of the strong ARM decoding cost. The important observation in this graph is that similar decoding costs can use the same scaling factor to approximate their actual decoding costs with acceptable error ratio. For example, the frames ranked in between the first and 7th can translate their decoding costs to its actual decoding costs by single scaling factor. And the frames ranked in between 8th and 30th also need single scaling factor.

The entire range of the decoding cost for the given video clip is identified by the best decoding cost and worst decoding cost from the decoding cost information file provided with the video clip. A client-defined parameter called *decoding cost partitioning factor* (denoted as N_p) decides the number of *cost segments* in the cost scaling table. Thus, *ith cost segment* is denoted as $Segment(i)$ and expressed:

$$d_{best} + (i-1) \times \Delta d < Segment(i) < d_{best} + i \times \Delta d \quad (1)$$

where, d_{worst} and d_{best} are worst decoding cost and best decoding cost, respectively, and Δd is $\frac{d_{worst} - d_{best}}{N_p - 1}$ which is the distance between the segment.

Next, from the cost scaling table, MPEG decoder selects the $Segment(i)$ corresponding to the decoding cost of the first frame (the decoding cost of the first frame is always 1.) and its scaling factor is set to 1.

Index	Frequency	Voltage	Power	Threshold
0	200MHz	3.3V	1W	10.7
1	180MHz	3.1V	0.9W	8.2
⋮	⋮	⋮	⋮	⋮
N_p-1	10MHz	0.5V	0.1W	0.2

Figure 6: An example of DVS table

5.2 Compulsory learning phase

MPEG decoder constructs the DVS table and decodes the first frame of the video clip at full speed. Also, the decoding time of the first frame is measured and denoted as t_{first}. Next, MPEG decoder constructs a DVS table to map each clock frequency and supply voltage pair to the appropriate actual decoding cost as shown in Figure 6. Using t_{first}, DVS table is constructed using the following equation.

$$threshold(i) = \frac{D - t_o}{t_{first}} \times \frac{t_p(0)}{t_p(i)} \quad (2)$$

where, D is a deadline to decode a frame, t_o is the overhead time to switch the voltage and frequency, $t_p(0)$ is the clock period at the full speed, and $t_p(i)$ is the clock period at the selection of row i from the DVS table. $threshold(i)$ represents the minimum actual cost of the given frame to select the corresponding voltage and clock frequency pair.

5.3 Check phase

First, MPEG decoder selects the row corresponding to the decoding cost of the current frame from the cost scaling table. If the scaling factor for this segment is not computed yet, MPEG decoder goes to learning phase to learn the scaling factor for this segment. Otherwise, MPEG decoder goes into DVS phase to set the processor with the appropriate voltage and clock frequency pair.

5.4 DVS phase

In this phase, MPEG decoder first computes the actual cost of the current frame using the scaling factor selected in Check phase. Next, the appropriate voltage and clock frequency pair is selected from the DVS table. If the actual cost of the frame is in between $threshold(i+1)$ and $threshold(i)$, the row i is selected as the appropriate voltage and clock frequency pair to decode the frame.

5.5 Learning phase

To learn the scaling factor for the given decoding cost, MPEG decoder decodes the frame i at full speed and measures the decoding time as we obtain t_{first}. We denote the decoding time of the frame i as t_i. $Segment(i)$ corresponding to the given decoding cost is selected and its scaling factor is set to $\frac{t_i}{t_{first}}$. This phase is visited $N_p - 1$ times.

6. EXPERIMENTAL RESULTS

We verified our method by considering three different system architectures. SimpleScalar was selected as the reference system and ST200 and strong ARM were chosen as the client systems.

As our evaluation metric, we use *idle time usage* (ITU) which is the fraction of idle time used for DVS over the total idle time. If the *idle time usage* is smaller than 1, the available idle time is under-utilized. On the other hand, if *idle time usage* is larger than 1, it is over-utilized and decoding process violates the deadline. We call the former *under-utilized idle time usage* (UITU) and the latter *over-utilized idle time usage* (OITU). Also, we call unity ITU *ideal idle time usage* (IITU). It is very important to classify the decoded frames into two categories based on the timing violation and analyze ITU for each category because ITU for all frames is insufficient to appreciate the severeness of timing violation and the efficiency of exploiting idleness without timing violation. Notice that MPEG decoder imposes soft real-time constraint rather than hard real-time constraint on each frame decoding, therefore marginal violation is acceptable for energy reduction. Because we are especially more interested in the timing-violated frames. *Over-utilized Frame Ratio* (OFR) and *Execution time over the Deadline Ratio* (EDR) are defined. OFR indicates the number of timing violated frames over total number of decoded frames and EDR is the average ratio of the execution time of over-utilized frame over the deadline.

We compared our method to *oracle DVS* (ODVS) which is an ideal method and only available by off-line analysis. *Oracle DVS* always provides the largest ITU (but less than or equal to IITU) compared to any other methods. OFR of ODVS is always 0 according to its definition, thus OITU of ODVS is also always 0 and its UITU is same to its ITU. However, ODVS does not always provide IITU when the system provides discrete voltage and frequency ranges.

To analyze how the number of voltage/frequency pairs affects our method, we consider two set of voltage/frequency pairs. Table 3 (a) and (c) were obtained using 5 voltage/frequency pairs and Table 3 (b) and (d) were obtained using 20 voltage/frequency pairs. [1] The deadline to decode a frame, D, was set to $20ms$ and the overhead time to change the frequency and voltage was set to $140us$ [13]. Finally, N_p (decoding cost partitioning factor) is set to 10. While the voltage and frequency are changed, we assume that processors consume their maximum power.

In Table 3, UITU of our method is comparable to ITU of ODVS - our method is as efficient as OVDS in idle time utilization. Also, OFR of our method is negligible except for `cindy(320x160)` in Table 3 (d). However, OITU in Table 3 (d) is only 1.061, which indicates that the amount of over-utilized idle time is marginal and acceptable under soft-realtime constraint, even if the number of over-utilized frames is non-trivial. EDR in this table supports this claim by showing that the the average execution time of the over-utilized frames is only 2.9% larger than the deadline.

It is also interesting to appreciate Table 3 with the change of the number of voltage/frequency pairs (Compare (a) to (b) and (c) to (d) in Table 3). As the number of voltage/frequency pairs is increased, so are ITU of both ODVS and our method because a widen set of possible selections of frequencies enables to control the frame decoding time more precisely. Also, OFR of our method is increased as the number of voltage/frequency pairs is increased (`klein` and `cindy(320x160)`. This is because the accuracy of the piece-wise linear model (scaling factor) becomes significant. As the number of voltage/frequency increases, the distance

[1]For each set, we assume that the frequency (f) range is from 200MHz to 10MHz and the frequencies are equally spaced. Each corresponding voltage is computed by $f = \frac{(V_{DD} - V_{TH})^\alpha}{V_{DD}}$ [14]. Also, we assume that the voltage at 200MHz is 3.3V, α is 2.0, and V_{TH} is 0.7V. When the voltage/frequency set is defined for a specific architecture like Xscale [3], the set can be directly applied.

MPEG clips	Our method					*ODVS*
	OFR	OITU	EDR	UITU	ITU	ITU
hubble(80x60)	0.000	0.000	0.000	0.634	0.634	0.642
joel(80x60)	0.000	0.000	0.000	0.454	0.454	0.462
klein(384x288)	0.000	0.000	0.000	0.199	0.199	0.216
cindy(80x60)	0.000	0.000	0.000	0.516	0.516	0.522
cindy(160x120)	0.000	0.000	0.000	0.604	0.604	0.614
cindy(320x240)	0.000	0.000	0.000	0.545	0.545	0.558

(a) ST200 (5 voltage / frequency pairs)

MPEG clips	Our method					*ODVS*
	OFR	OITU	EDR	UITU	ITU	ITU
hubble(80x60)	0.007	1.074	1.058	0.799	0.801	0.820
joel(80x60)	0.000	0.000	0.000	0.733	0.733	0.743
klein(384x288)	0.000	0.000	0.000	0.813	0.813	0.833
cindy(80x60)	0.042	1.028	1.025	0.837	0.845	0.829
cindy(160x120)	0.085	1.116	1.090	0.788	0.816	0.822
cindy(320x240)	0.000	0.000	0.000	0.858	0.858	0.905

(b) ST200 (20 voltage / frequency pairs)

MPEG clips	Our method					*ODVS*
	OFR	OITU	EDR	UITU	ITU	ITU
hubble(80x60)	0.000	0.000	0.000	0.518	0.518	0.525
joel(80x60)	0.000	0.000	0.000	0.323	0.323	0.329
klein(384x288)	0.000	0.000	0.000	0.789	0.789	0.804
cindy(80x60)	0.000	0.000	0.000	0.422	0.422	0.428
cindy(160x120)	0.000	0.000	0.000	0.513	0.513	0.522
cindy(320x240)	0.000	0.000	0.000	0.556	0.556	0.583

(c) Strong ARM (5 voltage / frequency pairs)

MPEG clips	Our method					*ODVS*
	OFR	OITU	EDR	UITU	ITU	ITU
hubble(80x60)	0.000	0.000	0.000	0.837	0.837	0.845
joel(80x60)	0.000	0.000	0.000	0.739	0.739	0.749
klein(384x288)	0.017	1.007	1.002	0.870	0.873	0.884
cindy(80x60)	0.000	0.000	0.000	0.688	0.688	0.695
cindy(160x120)	0.000	0.000	0.000	0.824	0.824	0.836
cindy(320x240)	0.128	1.061	1.029	0.859	0.885	0.903

(d) Strong ARM (20 voltage / frequency pairs)

Table 3: *Idle time usage* **comparison between our method and** *oracle DVS*

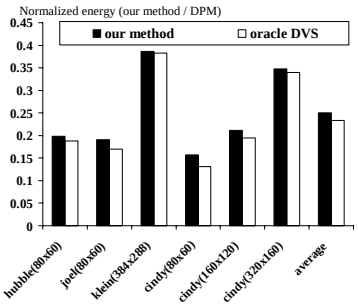

Figure 7: Energy consumption comparison between our method and *oracle DVS* **for Table 3 (d)**

between the thresholds in DVS table is decreased. Thus, even small error in scaling factor may alter the row selection in DVS table. However, such undesirable selection has small impact on OITU and EDR because the frequency step in Table 3 (b) and (d) is 4 times smaller than that in Table 3 (a) and (c). This limitation can be overcome by increasing N_p. When N_p is 50, OFR of `cindy(320x160)` becomes 0.012.

Finally, we compared the energy saving of our method to *ODVS*. Figure 7 shows the energy consumption of each method normalized to the energy consumption when *Dynamic Power Management* (DPM) [1] is adopted for Table 3 (d). We assume that the power consumption of each client system is zero in sleep state and there is no overhead in power state transition. Nevertheless, our method and ODVS outperform DPM and the actual energy improvements of both methods will be larger than Figure 7. Also, our method is comparable to *ODVS*, which is impossible without the decoding cost information used in our approach.

7. CONCLUSIONS

We proposed a novel DVS technique for multimedia application programs. Our method requires that contents providers (server site) supply the information of the execution time variations in addition to the content itself. Therefore, it is possible to make DVS techniques be independent of the worst case execution time estimation, which is one of the most critical factors affecting the quality of DVS techniques. The extra work required to the contents provider for this purpose is fully compensated by the benefits for the end users, because single content is typically provided to more than thousands users. The experimental results show that our method is comparable to the ideal method - *oracle DVS* - in terms of both *idle time usage* and energy saving when the client systems are different from the server system. The next generation of MPEG standard format may support the energy related information like the decoding cost information proposed in this paper for convenience and to reduce the latency to transmit the information.

8. REFERENCES

[1] L. Benini and G. De Micheli, *Dynamic Power Management of Circuits and Systems: Design Techniques and CAD Tools*, Kluwer, 1997.

[2] D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2.0", *Computer Architecture News*, pp. 13-25, June, 1997

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

[4] J. Flinn and M. Satyanarayanan, "Energy-aware Adaptation for Mobile Applications", *ACM Symposium on Operating Systems Principles*, pp. 48-63, 1999

[5] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld, "Policies for Dynamic Clock Scheduling", *Symposium on Operating Systems Design & Implementation*, Oct. 2000

[6] http://www.st.com

[7] http://www.arm.com

[8] C. Im, H. Kim, S. Ha, "Dynamic Voltage Scaling Techniques for Low-Power Multimedia Applications Using Buffers", *International Symposium on Low Power Electronics and Devices*, pp. 34-39, 2001

[9] T. Ishihara and H.Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors", *International Symposium on Low Power Electronics and Design*, pp. 197-202, 1999

[10] S. Lee and T. Sakurai, "Run-time Voltage Hopping for Low-power Real-time Systems", *Design Automation Conference*, pp.806-809, 2000

[11] A. Manzak and C.Chakrabarti, "Variable Voltage Task Scheduling Algorithms for Minimizing Energy", *International Symposium on Low Power Electronics and Design*, pp. 279-282, 2001

[12] T. Pering, T. Burd, and R. Broderson, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms", *International Symposium on Low Power Electrnoics and Design*, pp. 76-81, 1998

[13] J. Pouwelse, K. Langendoen, and H. Sips "Energy Priority Scheduling for Variable Voltage Processors", *International Symposium on Low Power Electronics and Devices*, pp. 28-33, 2001

[14] T. Sakurai and A. Newton, "Alpha-power Law MOSFET Model and its Application to CMOS Inverter Delay and Other Formulas", *IEEE Journal of Solid State Circuits*, vol.25, no.2, pp.584-594, Apr, 1990

[15] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Scheduling", *Design Automation Conference*, pp.134-139, 1999

[16] D. Shin and J. Kim, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications", *IEEE Design & Test of Computers*, Vol. 18, No. 2, pp. 20-30, 2001

[17] D. Wu, Y. Hou, W. Zhu, Y, Zhang, J. Peha, "Streaming Video over the Internet: Approaches and Directions", *IEEE trans. on Circuits and Systems for Video Technology*, vol. 11, no. 1, , pp.1-20, Feb. 2001

[18] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy", *IEEE Annual Foundations of Computer Science*, pp. 374-382, 1995