Dynamic Voltage Scaling for Portable Systems

Tajana Simunic Luca Benini* Andrea Acquaviva* Peter Glynn† Giovanni De Micheli

Computer Systems Laboratory
Stanford University

†Management Science and Engineering Department
Stanford University

*DEIS
University of Bologna

ABSTRACT

Portable systems require long battery lifetime while still delivering high performance. Dynamic voltage scaling (DVS) algorithms reduce energy consumption by changing processor speed and voltage at run-time depending on the needs of the applications running. Dynamic power management (DPM) policies trade off the performance for the power consumption by selectively placing components into low-power states. In this work we extend the DPM model presented in [2, 3] with a DVS algorithm, thus enabling larger power savings. We test our approach on MPEG video and MP3 audio algorithms running on the SmartBadge portable device [1]. Our results show savings of a factor of three in energy consumption for combined DVS and DPM approaches.

1. INTRODUCTION

Battery-operated portable systems impose tight constraints on energy consumption. Portable systems often consist of one or more microprocessors and a set of devices with multiple low-power states. Many microprocessors support dynamic clock frequency adjustment, and some newer devices also support dynamic supply voltage setting [4]. Thus at system level it is possible to reduce energy by changing the frequency and voltage level of the microprocessor (dynamic voltage scaling) and by transitioning components into low-power states (dynamic power management).

In this work we extend the DPM model presented in [2, 3] with a DVS algorithm, thus enabling larger power savings. The algorithm is implemented for the SmartBadge portable device [1]. The SmartBage consists of a StrongARM processor, memory, RF link and display. All components have four main power states: *active, idle, standby* and *off*. In addition, the processor can operate over a range of frequencies. For each frequency, there is a minimum allowed voltage of operation. If the processor is run at the minimum frequency and voltage required to sustain the performance level required by the application, it is possible to save power even when the system is active, in addition to the savings that can be obtained by DPM during idle periods. This principle is exploited by the recently announced Transmeta's Crusoe processor [4].

A DVS algorithm sets the microprocessor voltage and frequency at run time depending on the behavior of applications currently running. Early DVS algorithms set processor speed based on the processor utilization of fixed intervals [5, 6]. The individual requirements of the tasks running were not considered, resulting in poor behavior for more complex workloads [13]. There has been a number of voltage scaling techniques proposed for real-time systems. The approaches presented in [7, 9, 8, 10] assume that all tasks run at their worst case execution time (WCET). The workload variation slack times are exploited on task-by-task basis in [11], and are fully utilized in [12]. Work presented in [14] introduces a voltage scheduler that determines the operating voltage by analyzing application requirements. The scheduling is done at task level, by setting processor frequency to the minimum value needed to complete all tasks. For applications with high frame-to-frame variance, such as MPEG video, schedule smoothing is done by scheduling tasks to complete twice the amount of work in twice the deadline.

In all DVS approaches presented in the past, scheduling was done at the task level, assuming multiple threads. The prediction of task execution times was done either using worst case execution times, or heuristics. Such approaches neglect that DVS can be done within a task or for single-application devices. For, instance, in MPEG decoding, the variance in execution time on frame basis can be very large: a factor of three in the number of cycles [15], or a range between 1 and 2000 IDCTs per frame [16] for MPEG video.

The first contribution of this work is to develop and verify a stochastic model for prediction of execution times for streaming multimedia applications on a frame-by-frame basis. Our model is based on the change-point detection theory used for ATM traffic detection among other applications [17]. We compare our model to prefect prediction and to exponential moving average used in [14]. The prediction algorithm developed is then used as a part of a power control strategy that merges DVS and DPM.

As opposed to DVS, power management algorithms aim at reducing energy consumption at the system-level by selectively placing components into low-power states during idle periods. DPM algorithms presented in the past can

be classified into deterministic and stochastic (for more detailed overview see [3]). Deterministic algorithms include basic timeout and predictive schemes. Stochastic models can give optimal DPM policies, as long as the basic assumptions made in the formulation of the model are true. A simple stochastic model for DPM assumes that idle times for system resources follow exponential distributions. Unfortunately, it has been shown in [2] that approaches purely based on exponential distributions do not model well real system behavior.

Two stochastic approaches recently presented allow usage of general distributions instead of just exponential. As a result, large power savings were observed. The first approach is based on renewal theory [2]. This model assumes that the decision to transition to low power state can be made in only one state. Another method developed is based on the Time-Indexed Semi-Markov Decision Process model (TISMDP) [3]. This model is more complex, but also has wider applicability because it assumes that a decision to transition into a lower-power state can be made from any number of states. Both approaches assume a single system active state and assume that energy consumption in that state is constant.

The second contribution of this work is to merge the DPM and the DVS approaches, by expanding the active state definition to include multiple settings of frequency and voltage, thus resulting in a range of performance and power consumptions available for tradeoff at run time. In this way, the power manager can control performance and power consumption levels both by using DVS when the system is active, and by transitioning components into low-power states when the system is idle.

The rest of the paper is organized as follows. Section 2 describes the stochastic models of the system components. The models are based on experimental observations. In Section 3 we present the theoretical basis for detection of rate change together with dynamic selection of CPU frequency and voltage. We show simulation and measurement results for MPEG video and MP3 audio running on the SmartBadge in Section 4. Finally, we summarize our findings in Section 5.

2. SYSTEM MODEL

The systems can be modeled with three components: the user (a source of external events), the device (SmartBadge) and the queue (the buffer associated with the device) as shown in Figure 1. The power manager observes all event occurrences of interest and takes decisions on what state the system should transition to next, in order to minimize energy consumption for a given performance constraint. While the device is active, the power manager selects the most appropriate execution frequency and voltage for the processor. As our work was motivated by a real design of the SmartBadge, in all our examples we use the SmartBadge hardware with MPEG video and MP3 audio.

Each system component is described probabilistically. The user behavior is modeled by a *request interarrival distribution*. For streaming multimedia applications, requests represent frame arrivals from the network. Similarly, the *service time distribution* describes the behavior of the device in the active state. In multimedia case, it represents the time needed for processing a frame (decompressing it and sending to the output interface). The *transition time distribution* models the time taken by the device to transition between its power states. Finally, the combination of interarrival time distribution (incoming frame arrivals) and service time distribution (frame decoding times) characterizes well the behavior of the queue (frame buffer). The details of each system component are described in the next sections.

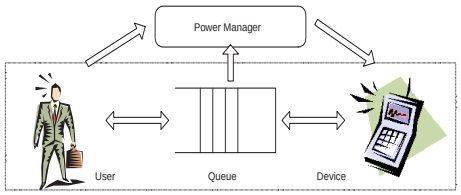

Figure 1: System Model

2.1 Portable Device

Portable devices typically have multiple power states. Each device has one active state in which it services user requests, and one or more inactive low-power states. The active state can further be characterized by a set of sub-states differentiated by performance (e.g. CPU frequency) and power consumption (e.g. CPU voltage). In addition, the power manager can trade off power for performance by placing the device into low-power states. Each low power state can be characterized by the power consumption and the performance penalty incurred during the transition to or from that state. Usually higher performance penalty corresponds to lower power states.

2.1.1 The SmartBadge Device

The SmartBadge, shown in Figure 2, is an embedded system consisting of Sharp's display, Lucent's WLAN RF link, StrongARM-1100 processor, RAM, FLASH, sensors, and modem/audio analog front-end on a PCB board powered by the batteries through a DC-DC converter. Note that the SmartBadge has two types of data memory – slower SRAM (1MB, 80ns) from Toshiba and faster DRAM (4MB, 15ns) from Micron that is used only during audio or video decode. Components in the SmartBadge, the power states and the transition times of each component from standby (t_{sby}) and off (t_{off}) state into active state are shown in the Table 1.

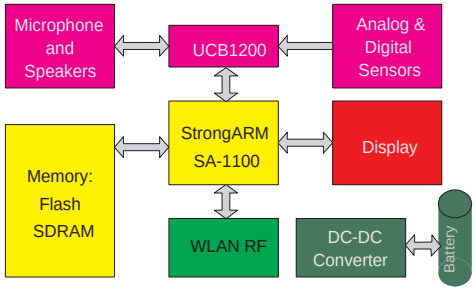

Figure 2: SmartBadge

The initial goal in designing the SmartBadge was to allow a computer or a human user to provide location and environmental information to a location server through a heterogeneous network. The SmartBadge could be used as a corporate ID card, attached (or built in) to devices such as PDAs and mobile telephones, or incorporated in computing systems. In this work we focus on using the SmartBadge as a PDA capable of MPEG video and MP3 audio playback.

Table 1: SmartBadge components

Component	Active P (mW)	Idle P (mW)	Stdby P (mW)	t_{sby} (ms)	t_{off} (ms)
Display	1000	1000	100	100	240
WLAN RF	1500	1000	100	40	80
SA-1100	400	170	0.1	10	35
FLASH	75	5	0.023	0.6	160
SRAM	115	17	0.13	5.0	100
DRAM	400	10	0.4	4.0	90
Total	3500	2200	200	110	705

The StrongARM processor on the SmartBadge can be configured at run-time to execute at a set of different frequencies. We measured the transition time between two different frequency settings at 150 microseconds. Since typical decoding time for MPEG video or MP3 audio is much longer than the transition time, it is possible to change the CPU frequency without perceivable overhead. For each frequency, there is a minimum voltage the SA-1100 needs in order to run correctly, but with lower energy consumption. Figure 3 shows the frequency-voltage tradeoff.

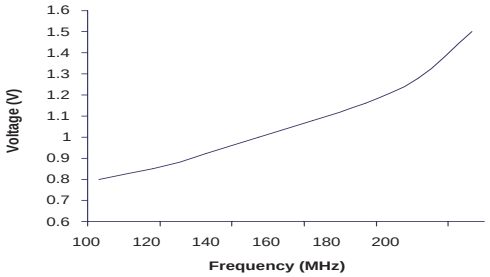

Figure 3: Frequency vs. Voltage for SA-1100

In addition to the active state, the SmartBadge supports three lower power states: *idle, standby and off*. The idle state is entered immediately by each component in the system as soon as that particular component is not accessed. The standby and off state transitions can be controlled by the power manager. The transition from standby or off state into the active state can be best described using the uniform probability distribution.

2.1.2 The Active State Model

Service times (decoding times for video or audio frames) on the SmartBadge in the active state are modeled by an exponential distribution. The average service time is defined by $\frac{1}{\lambda_D}$ where λ_D is the average service rate (measured in frames/second for MPEG video and MP3 audio). Equation 1 defines the cumulative probability of the device servicing a user request within time interval t.

$$F_D(t) = 1 - e^{-\lambda_D t} \quad (1)$$

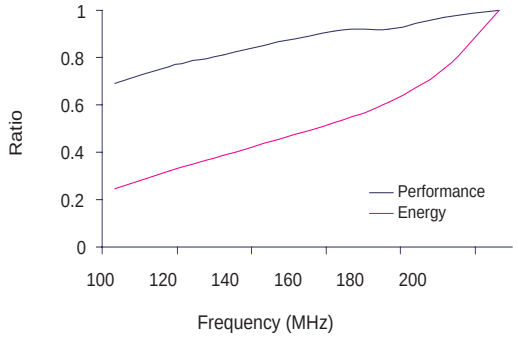

Figure 4: Performance and energy for MP3 audio

Figure 4 shows the tradeoff between performance and energy when running MP3 audio decode on the SmartBadge hardware at allowable frequency and voltage setting for the SA-1100 processor, and Figure 5 shows the same results for MPEG video. The shape of the performance curve versus processor frequency setting depends on the application and on the underlying hardware. MP3 audio was decoded using slower SRAM on the SmartBadge. Since memory access time does not depend on processor clock frequency, performance improvements at high processor frequencies are memory-bound, and speedup is not linear. MPEG video decode ran on much faster SDRAM and thus its performance curve is almost linear as it is more limited by the processor speed. In both figures all values are normalized to the data points obtained for the fastest frequency.

The basic rationale for DVS is that for frames that take a shorter time to decode, processor frequency and voltage can be lowered, and for longer frames, increased. In addition, the decoding speed needs to be adjusted to frame arrival frequency, so that the frame buffer does not contain too many or too few frames. The detection of changes in decoding speed and arrival frequency are thus critical for optimal setting of CPU frequency and voltage. We present an optimal way for detection in section 3.

2.2 User Model

User is a source of external events to the device. The requests to the multimedia application during the decoding are in form of audio or video frame arrivals through the WLAN. Thus, the user's stochastic model in the active state can be defined by the frame interarrival time distribution. We measured MPEG video and MP3 audio frame arrival times (user requests in the abstract system model) by monitoring the accesses to the WLAN card. The frame interarrival times in the active state for both applications can be approximated with an exponential distributions. Figure 6 shows exponential cumulative distribution fitted to the measured results for the MPEG video. Similar results have been observed for the MP3 audio. Frame arrival rate in the active state is

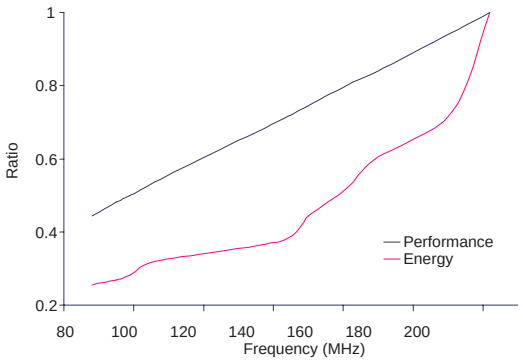

Figure 5: Performance and energy for MPEG video

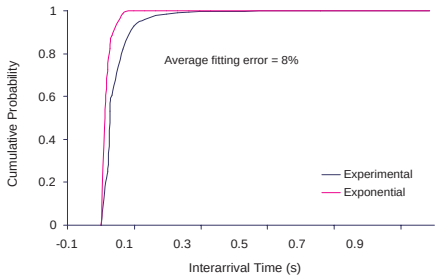

Figure 6: MPEG video arrival time distribution

defined as λ_U and the mean frame interarrival time is $\frac{1}{\lambda_U}$. The probability of the SmartBadge receiving a frame within time interval t follows the cumulative probability distribution shown below.

$$F_U(t) = 1 - e^{-\lambda_U t} \tag{2}$$

Note that the exponential distribution is *not* used to model the arrivals in the idle state. In the idle state, audio or video frames have all been decoded and no new requests have arrived yet from the user. This is when the power manager can make a decision on what low-power state to place the device in as discussed in [2]. Remember that the full optimization model should not only decide when to transition the device into one of the low-power states (standby or off) but should also perform dynamic voltage scaling in the active state.

2.3 Queue

Portable devices normally have a buffer for storing requests that have not been serviced yet. For multimedia requests such as MPEG video and audio it is convenient to describe queue in terms of the number of frames waiting in the frame buffer. As the frames arriving to the SmartBadge do not have priority, our queue model contains only the number of frames waiting service (decoding). In the active state, where the exponential distributions is used to describe frame arrivals and service times, the behavior of the system can be modeled using M/M/1 queue model. More details on this model and its application to dynamic voltage scaling are given in the following section.

3. THEORETICAL BACKGROUND

In the work presented in [2, 3], the power manager's only job is to decide when to transition the device into one of the low-power states. Power management policies are obtained using one of two models: renewal theory model [2] and time-indexed semi-markov process model [3]. It was observed that in the idle state we need to accurately model the tail of the interarrival time distribution, which does not follow a perfect exponential distribution. As a result, the time elapsed since the last entry into the idle state had to be accounted for in the model in order to obtain the optimal power management policy. Renewal theory naturally accounts for the time elapsed in the idle state through formulation of the system renewal time. In the TISMDP model, instead of the simple state model shown on the left in Figure 7, it was necessary to expand the idle and the sleep states with time index representing elapsed time since the last entry into the idle state as shown on the right. Note that in both renewal and TISMDP models there is only one active state (with one or more elements in the queue).

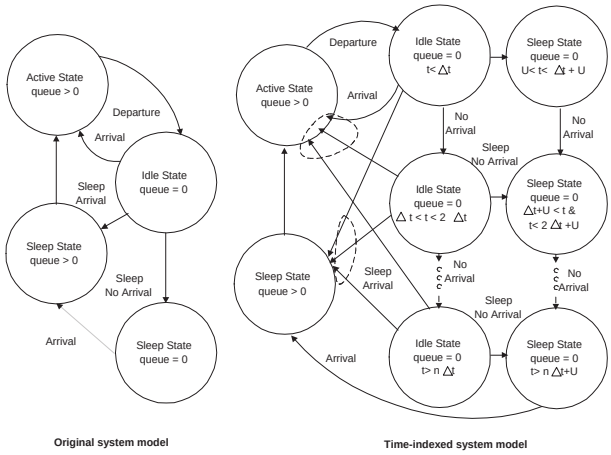

Figure 7: Time-indexed SMDP states

In this work we have extended the function of power manager (PM) to include decisions on the CPU frequency and voltage setting while in the active state. Thus, instead of having only one active state as shown in Figure 7, now there is a set of active states, each characterized by different performance (CPU frequency) and power consumption (CPU voltage) as shown in Figure 8. Since TISMDP and renewal models both assumed that active state can be described using the exponential distribution, the transformation from one active into multiple active states is completely compatible with the rest of the model. As a result, the power management policies we develop can make decisions for both dynamic voltage setting and the transition into the low-power states.

At run-time, the PM observes user request arrivals and service completion times (in our case frame arrivals and decoding times), the number of jobs in the queue (the number of frames in the buffer) and the time elapsed since last entry into idle state. When in the active state, the PM checks if the rate of incoming or decoding frames has changed, and then adjusts the CPU frequency and voltage accordingly.

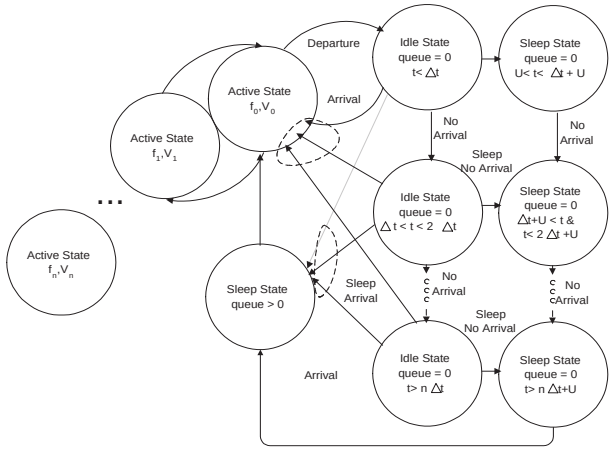

Figure 8: Expansion of the active state

Once the decoding is completed, the system enters idle state. At this point the power manager observes the time spent in the idle state, and depending on the policy obtained using either renewal theory or TISMDP model, it decides when to transition into one of the sleep states. When a request from the user arrives for more audio or video decoding, the power manager transitions the system back into the active state and starts the decoding process.

We next present the optimal approach for detecting a change in the frame arrival or decoding times. Once a change is detected, a decision has to be made on how to set the CPU frequency and voltage. We present results based on M/M/1 queue theory that enable power manager to make this decision.

3.1 Dynamic Voltage Scaling Algorithm

The DVS algorithm consists of two main portions: detection of the change in request arrival or servicing rate, and the policy that then adjusts the CPU frequency and voltage. The detection is done using maximum likelihood ratio that guarantees optimal detection for exponential distributions. Policy is implemented using M/M/1 queue results to ensure constant average delay experienced by buffered frames.

Detecting the change in rate is a critical part of optimally matching CPU frequency and voltage to the requirements of the user. For example, the rate of MP3 audio frames coming via RF link can change drastically due to changes in the environment. The servicing rate can change due to variance in computation needed between MPEG frames [15, 16], or just by changing the audio source currently decoded by the MP3 audio. The request (frame) interarrival times and servicing (decoding) times follow exponential distribution as discussed in the previous section. The two distributions are characterized by the arrival rate, λ_U, and the servicing rate, λ_D.

The change point detection is performed using maximum likelihood ratio, P_{max}, as shown in Equation 3. Maximum likelihood ratio computes the ratio between the probability that a change in rate did occur (numerator in Equation 3) and the probability that rate did not change (denominator).

The probability that the rate changed is computed by fitting the exponential distribution with an old rate, λ_o, to the first $k-1$ interarrival or decoding times (x_j), and another exponential distribution with a new rate, λ_n, to the rest of the points observed in window of size m. The probability that the rate did not change is computed by fitting the interarrival or decoding times with the exponential distribution characterized by the current (or old) rate, λ_o.

$$P_{max} = \frac{\Pi_{j=1}^{k-1} \lambda_o e^{-\lambda_o x_j} \Pi_{j=k}^{m} \lambda_n e^{-\lambda_n x_j}}{\Pi_{j=1}^{m} \lambda_o e^{-\lambda_o x_j}} \qquad (3)$$

A more efficient way to obtain the maximum likelihood ratio is to calculate the natural log of P_{max} as shown below:

$$ln(P_{max}) = (m-k+1) ln\frac{\lambda_n}{\lambda_o} - (\lambda_n - \lambda_o) \sum_{j=k}^{m} x_j \qquad (4)$$

Note that in this equation, only the sum of interarrival (or decoding) times needs to be updated upon every arrival (or service completion). A set of possible rates, Λ, where $\lambda_n, \lambda_o \in \Lambda$ is predefined, as well as the size of the window m. Variable k is used to locate the point in time when the rate has changed. Stochastic simulation is done to obtain the value of $ln(P_{max})$ that is sufficient to detect the change in rate. The results are accumulated in a histogram, and then the value of maximum likelihood ratio that gives very high probability that the rate has changed is chosen. In our work we selected 99.5% likelihood. At run time the interarrival time sums are collected and maximum likelihood ratio is calculated. We found that a window of $m = 100$ is large enough. Larger windows will cause longer execution times, while much shorter windows do not contain statistically large enough sample and thus give unstable results. In addition, the change point can be checked every $k = 10$ points. Larger values of k interval mean that the changed rate will be detected later, while with very small values the detection is quicker, but also causes extra computation. If the maximum likelihood ratio computed is greater than the one obtained from the histogram, then there is 99.5% likelihood that the rate change occurred, and thus the CPU frequency and voltage need to be adjusted.

The adjustment of frequency and voltage is done using M/M/1 queue model. Using this model we try to keep the average total delay for processing frames in the queue constant (Equation 5).

$$Frame_{delay} = \frac{\lambda_D}{\lambda_U(\lambda_U - \lambda_D)} \qquad (5)$$

When either interarrival rate, λ_U, or the servicing rate, λ_D, change, the frame delay is evaluated and the new frequency and voltage are selected that will keep the frame delay constant. For example, if the arrival rate for MP3 audio changes, Equation 5 is used to obtain required decoding rate in order to keep the frame delay (and thus performance) constant. The decoding rate can be related back to the processor frequency setting using Figure 4 or an equivalent table. On the other hand, if a different frame decoding rate is detected while processor is set to the same frequency, then piece-wise linear approximation based on the application frequency-performance tradeoff curve (Figures 4 and 5) is used to obtain the new processor frequency setting. In

either case, when CPU frequency is set to a new value, the CPU voltage is always adjusted according to Figure 3.

4. RESULTS

We implemented the change point detection algorithm as a part of the power manager for both MPEG video and MP3 audio examples. When the system is in the active state (the state where audio and video decoding occur), the power manager (PM) observes changes in the frame arrival and decoding rates using change point detection algorithm described in the previous section. Once a change is detected, the PM evaluates the required value of the processor frequency that would enable the frame delay expressed in Equation 5 to remain constant. The CPU voltage is set using results shown in Figure 3. Figure 9 shows the relationship between CPU frequency and MPEG video frame arrival and decoding rates for average buffered frame delay of 0.1 seconds, which then corresponds to an average of 2 extra frames of video buffered. This example is for a clip of football video decoded on the SmartBadge. Similar results can be obtained for other clips, but with different decoding rates, as the rates depend on the content and on the hardware architecture.

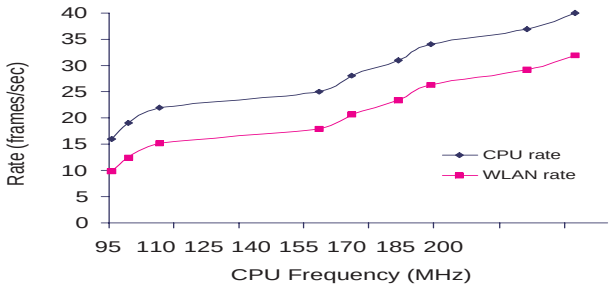

Figure 9: MPEG Frame Rates vs. CPU Frequency

We compare our rate change detection algorithm to ideal detection and to exponential moving average algorithm. Exponential moving average can be defined as follows:

$$Rate_{ave}^{new} = (1-g)Rate_{ave}^{old} + gRate^{cur} \quad (6)$$

where $Rate_{ave}^{new}$ is the new average rate, $Rate_{ave}^{old}$ is the old average, $Rate^{cur}$ is the current measured rate and g is the gain. Figure 10 shows the comparison results for detecting a change from 10 fr/sec to 60 fr/sec. Our algorithm detects the correct rate within 10 frames and is more stable than the exponential moving average algorithm.

Table 2: MP3 audio streams

MP3 Audio Clip Label	Bit rate (Kb/s)	Sample Rate (KHz)	Decoding Rate (frames/s)
A	16	16	51.35
B	16	32	27.30
C	32	16	49.80
D	32	32	26.05
E	64	16	47.95
F	64	32	25.25

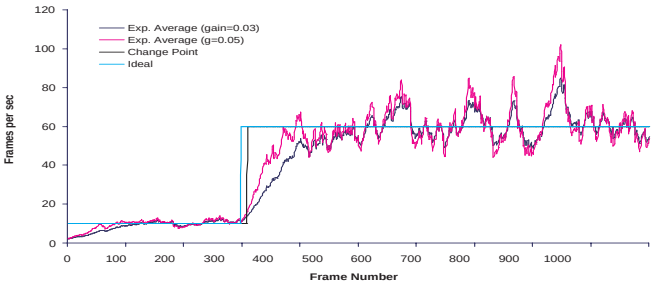

Figure 10: Rate Change Detection Algorithms

In the following set of results we compare (i) the ideal detection algorithm, (ii) the exponential average approximation used in previous work and (iii) the maximum processor performance to (iv) the change point algorithm presented in this paper. For this purpose we use six audio clips totaling 653 seconds of audio, each running at a different set of bit and sample rates as shown in Table 2. We have found that there was very little variation on frame-by-frame basis in decoding rate within a given audio clip, but the variation in decoding rate between clips can be large as shown in Table 2 (the decoding rates are for 202.4MHz processor frequency).

Table 3: MP3 audio DVS

MP3 Audio Sequence	Result	Ideal	Change Point	Exp. Ave.	Max
ACEFBD	Energy	196	217	225	316
	Fr.Delay	0.1	0.09	0.1	0
BADECF	Energy	189	199	231	316
	Fr.Delay	0.1	0.09	0.1	0
CEDAFB	Energy	190	214	232	316
	Fr.Delay	0.1	0.04	0.1	0

During decoding, the DVS algorithm detects changes in both arrival and decoding rates for the MP3 audio sequences. The resulting energy (kJ) and average total frame delay (s) are displayed in Table 3. Each sequence consists of a combination of six audio clips. For all sequences, the frame arrival rate varied between 16 and 44 frames/sec. Our change point algorithm performs well, its results are very close to the ideal, with no performance loss as compared to the ideal detection algorithm that allows an average $0.1s$ total frame delay (corresponding to 6 extra frames of audio in the buffer).

Table 4: MPEG video DVS

MPEG Video Clip	Result	Ideal	Change Point	Exp. Ave.	Max
Football	Energy	214	218	300	426
(875s)	Fr.Delay	0.1	0.11	0.16	0
Terminator2	Energy	280	294	385	570
(1200s)	Fr.Delay	0.1	0.11	0.16	0

The next set of results are for decoding two different video clips. In contrast to MP3 audio, for MPEG video there is a large variation in decoding rates on frame-by-frame basis (this has been shown in [15, 16] as well). We again report results for ideal detection, exponential average, maximum processor performance and our change point algorithm. The

ideal detection algorithm allows for 0.1s average total frame delay equivalent to 2 extra frames of video in the buffer. The arrival rate varies between 9 and 32 frames/second. Energy (kJ) and average total frame delay (s) are shown in Table 4. The results are similar to MP3 audio. The exponential average shows poor performance and higher energy consumption due to its instability (see Figure 10). Our change point algorithm performs well, with significant savings in energy and a very small performance penalty (0.11s frame delay instead of allowed 0.1s).

Table 5: DPM and DVS

Algorithm	Energy (kJ)	Factor
None	4260	1.0
DVS	3142	1.4
DPM	2460	1.7
Both	1342	3.1

Finally, we combine the dynamic voltage scaling detection with power management algorithms presented in [2, 3]. We use a sequence of audio and video clips, separated by idle time. During longer idle times, the power manager has the opportunity to place the SmartBadge in the standby state. The optimal power management policy can be obtained by either of the two approaches presented in [2, 3] as the only decision point is upon the entrance into the idle state. Table 5 shows the energy savings if we implemented only dynamic voltage scaling (and thus did not transition into standby state during longer idle times), or if only power management is implemented (and thus processor runs at maximum frequency and voltage in the active state) and finally also for the combination of the two approaches. We obtain savings of a factor of three when expanding the power manager to include dynamic voltage scaling with our change point detection algorithm.

5. CONCLUSIONS

We presented a new approach for dynamic voltage scaling that can be used as a part of a power managed system, such as systems presented in [2, 3]. Our dynamic voltage scaling algorithm is based on two critical portions: (i) change point detection algorithm that detects the change in arrival or decoding rates, and (ii) the frequency setting policy that sets the processor frequency and voltage based on the current arrival and decoding rates in order to keep constant performance. We tested our approach on MPEG video and MP3 audio algorithms running on the SmartBadge portable device [1]. Our change point detection algorithm is very stable as compared to the exponential moving average algorithm presented previously. As a result, it gives large energy savings at a small performance penalty for both MPEG video and MP3 audio applications. Finally, we implemented our DVS algorithm together with power management algorithms and show **factor of three savings in energy** due to the combined approach.

6. REFERENCES

[1] G. Q. Maguire, M. Smith and H. W. Peter Beadle "SmartBadges: a wearable computer and communication system", *6th International Workshop on Hardware/Software Codesign*, 1998.

[2] T. Simunic, L. Benini and G. De Micheli, "Energy Efficient Design of Portable Wireless Devices", *International Symposium on Low Power Electronics and Design*, pp. 49–54, 2000.

[3] T. Simunic, L. Benini and G. De Micheli, "Dynamic Power Management for Portable Systems", *The 6th International Conference on Mobile Computing and Networking*, pp. 22–32, 2000.

[4] L. Geppert, T. Perry, "Transmeta's magic show," *IEEE Spectrum*, vol. 37, pp.26–33, May 2000.

[5] M. Weiser, B. Welch, A. Demers, S. Shenker, "Scheduling for reduced CPU energy," *Proceedings of Symposium on Operating Systems Design and Implementation*, pp.13–23, Nov. 1994.

[6] K. Govil, E. chan, H. Wasserman, "Comparing algorithms for Dynamic speed-setting of a low-power CPU," *Proceedings of Internactional Conferenc on Mobile Computing and Networking*, Nov. 1995.

[7] F. Yao, A. Demers, S. Shenker, "A scheduling model for reduced CPU energy," *IEEE Annual foundations of computer sciend*, pp.374–382, 1995.

[8] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M. Srivastava, "Power optimization of variable voltage-core based systems," *Proceedings of Design Automation Conference*, pp.176–181, 1998.

[9] I. Hong, M. Potkonjak, M. Srivastava, "On-line Scheduling of Hard Real-time Tasks on Variable Voltage Processor," *Proceedings of International Conference on Computer-Aided Design*, Nov. 1998.

[10] T. Ishihara, H. Yasuura, "Voltage Scheduling Problem for dynamically variable voltage processors," *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, pp.197–202, 1998.

[11] Y. Shin, K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proceedings of Design Automation Conference*, pp.134–139, 1999.

[12] S. Lee, T. Sakurai, "Run-time voltage hopping for low-power real-time systems," *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, pp.806–809, 2000.

[13] T. Pering, T. Burd, R. Brodersen, "The simulation and evaluation of Dynamic Voltage Scaling Algorithms" *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, 1998.

[14] T. Pering, T. Burd, R. Brodersen, "Voltage scheduling in the IpARM microprocessor system" *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, pp.96–101, 2000.

[15] A. Bavier, A. Montz, L. Peterson, "Predicting MPEG Execution Times," *Proceedings of SIGMETRICS*, pp.131–140, 1998.

[16] A. Chandrakasan, V. Gutnik, T. Xanthopoulos, "Data Driven Signal Processing: An Approach for Energy Efficient Computing," *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, pp.347–352, 1996.

[17] R. Vesilo, "Cumulative Sum Techniques in ATM Traffic Management," *Proceedings of IEEE GLOBECOM*, pp.2970–2976, 1998.