Memory-aware dynamic voltage scaling for multimedia applications

J. Choi and H. Cha

Abstract: As the computing environments are continuously moving towards battery-operated mobile and handheld systems, the development of energy-saving mechanisms for such devices has recently become a technical challenge. Dynamic voltage scaling (DVS) has historically been considered an effective method to reduce the processor power consumption. Conventional DVS techniques typically consider only processor utilisation issues in a policy-making process. However, as memory-bound multimedia applications are becoming popular in handheld devices, the DVS policies should consider the so-called 'memory wall' problem to maximise energy gain. Recent DVS techniques suffer from the inefficiency of their policies caused by the memory-wall problem while executing multimedia applications, and no previous research on DVS considers the problem explicitly. The existence of the memory wall problem in a real system is revealed and a metric that can be used to detect the problem in advance is found. A memory-aware DVS (M-DVS) technique that takes the memory wall problem fully into consideration is proposed. The experimental results on a PDA show that M-DVS can reduce ~8% of additional power consumption, compared with conventional DVS, without any QoS degradation for handling multimedia clips.

1 Introduction

Dynamic voltage scaling (DVS) is a technique to prolong battery life in handheld devices, such as personal digital assistants (PDAs) and Smartphone, by scaling the frequency and voltage of the processor dynamically depending on the execution complexity of running applications. Weiser *et al.* [1] showed that dynamic adjustment of a processor frequency, based on the processor utilisation, reduces power consumption. Govil *et al.* [2] further proposed various kinds of prediction techniques for processor utilisation and showed that the power reduction depends not only on the accuracy of prediction but also on the smoothness in the performance level changes. Although DVS techniques based on processor utilisation effectively reduce the power consumption, the techniques are not optimal to support multimedia applications where external memory is frequently accessed. This inadequacy is because current prediction techniques do not fully consider the memory-bound nature of sequential multimedia access in their DVS policies.

Recent work on DVS does not explicitly consider the interaction between the processor and the memory, although they adopt multimedia applications as key workloads [3–5]. Grunwald *et al.* [6] pointed out that the non-linear performance characteristics between the processor and the memory are related to memory access latency. Martin *et al.* [7, 8] studied the problem from the system's point of view and showed that low-performance memory causes a performance bottleneck in executing multimedia applications. Martin emphasised that this memory bottleneck problem should be considered in the development of DVS techniques, but no concrete methodology was discussed. Fan *et al.* [9] showed that an optimal power reduction can be achieved by considering the power characteristics of the processor and the memory. However, the suggested mechanism on dynamic power state transition requires long latency in accessing the main memory, and it is not practically feasible to change the power state of a particular memory bank during operation. Weissel and Bellosa [10] proposed a DVS technique that utilises event counters to set the operating frequency of the processor in consideration of the memory wall effect. A metric called energy performance ratio is used for their DVS policy, but the policy does not consider the voltage scaling and hence their approach has practical limitations in its reduction of power. Choi *et al.* [11] proposed a technique to reduce power consumption by lowering the processor performance when decoding multimedia clips residing in external memory. However, the technique is based on the rather unrealistic assumption that the external memory operates at a constant speed regardless of the processor frequency. Also, the processor studied in their experiments is not adequate for handheld devices, hence the result may not be of practical use. Cho and Chang [12] proposed a memory-aware DVS (M-DVS) policy that assigns optimal frequencies of both memory and processor, considering the system-level power consumption characteristics. The policy, however, assumes a linear relationship between the processor frequency and the memory frequency, which is not adequate in fine granularity DVS processors.

This paper focuses on the development of a DVS policy that considers the non-linear performance characteristics, the so-called 'memory wall' problem, that are typically found in handheld devices when running memory-bound multimedia applications. We experimentally reveal the existence of the memory wall problem in a real system

© IEE, 2006
IEE Proceedings online no. 20050031
doi:10.1049/ip-cdt:20050031
Paper first received 29th January and in revised form 9th November 2005

The authors are with the Department of Computer Science, Yonsei University, Seodaemun-gu, Shinchon-dong 134, Seoul 120-749, Korea
E-mail: hjcha@cs.yonsei.ac.kr

and propose a metric that can be used to detect the problem in advance. An M-DVS that takes the memory wall problem fully into consideration is then proposed.

2 Memory wall problem

The memory wall problem is a well-known issue in computer architecture research [13, 14], where overall system performance is degraded because of lower-speed memory device compared with processor speed. The problem is aggravated in handheld devices because high performance Rambus DRAM or DDR DRAM cannot be adopted because of their high power consumptions. Additionally, multimedia applications that are commonly used in handheld devices require frequent and sequential memory accesses. To show the memory wall problem experimentally, a set of video clips, 300, 500, and 700 kbps MPEG clips, are played using MPlayer [15] on a PXA270-based embedded platform [16] that runs the Linux operating system. The PXA270 is a general-purpose processor based on the Xscale architecture and has a DVS capability of changing its frequency from 117 to 416 MHz.

Fig. 1 shows the changes in processor utilisation, measured in the kernel context, as the processor frequency increases. Processor utilisation does not decrease linearly, although the processor's operating frequency increases linearly. That is, the processor utilisation starts to increase in the range between 208 and 286 MHz, then decreases again after that. This demonstrates that the system performance is indeed affected by a factor other than the frequency of processor. It is possibly because of the memory component because low memory performance cannot keep up with the performance of the processor, especially when running memory-bound multimedia applications. The phenomenon is observed consistently with other clips with different bit rates. The higher the bit rate, in effect as the access frequency increases, the more obvious the effect.

The observed memory wall problem is caused by the structural characteristic of the SoC-based processor, which usually has a uni-clock source for distributing clocks to processor and memory, and also uses low-to-mid-speed memory for low power consumption. Implementing the uni-clock leads to design simplification and cost reduction. Fig. 2 shows that the operating frequencies of the processor and the memory have different gradients, but the trends are the same in each region. P_{mw} is defined as the range of processor frequency causing the memory wall problem. In this range, the memory performance is lower than that of the processor and this induces a processor blockage in executing multimedia applications. Consequently, it results in excessive processor utilisation.

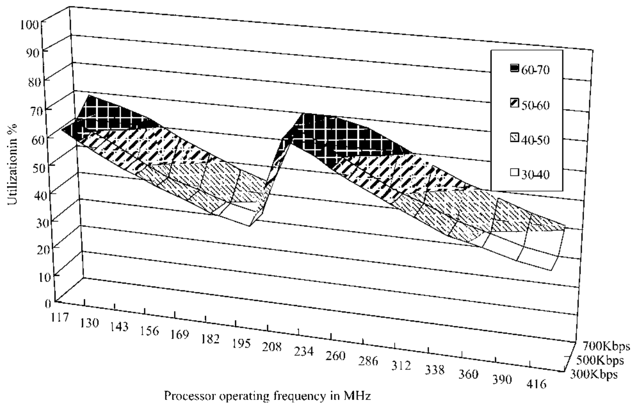


Fig. 1 *Memory wall problem*

Conventional DVS policies simply use overestimated processor utilisation, caused by the memory wall problem, as a metric for the performance setting of the processor. Hence, the policies do not accurately reflect the actual utilisation of the processor. To illustrate this, we define the system utilisation and the jobs to be executed as follows. Note that in our work a multimedia application, which typically shows periodic execution patterns, is considered for the domain of study. The system utilisation $U_{sys}(i)$ at ith period refers to the overall utilisation of the system. $U_{sys}(i)$ is measured by the kernel and includes both the processor and the memory utilisation and ranges between U_{sys}^{lb} and U_{sys}^{ub}, which are, respectively, the minimum and the maximum available utilisation in the system. $J(i)$ refers to the actual job at the ith period of a process that has a period Q. $J_{proc}(i)$ and $J_{mem}(i)$ are, respectively, the portion of the processor-bound job and the memory-bound job within $J(i)$.

Fig. 3 shows an example of a periodic process. Here, $J(i)$ consists of $J_{proc}(i)$ and $J_{mem}(i)$ and $U_{sys}(i)$ is defined as $(T_{mem}(i) + T_{proc}(i))/Q$. In other words, $U_{sys}(i)$ is the ratio of the execution time of both the processor-bound job and the memory-bound job in the ith period, over the period Q. During the period of $J_{mem}(i)$, the processor blocks and does nothing but waits for the media data to be available to the processor. The $U_{sys}(i)$ metric that has typically been used in conventional DVS polices does not, however, differentiate $J_{mem}(i)$ and $J_{proc}(i)$. Therefore the overall system utilisation is overestimated because of the existence of the memory wall problem. Accordingly, the memory wall problem that may be confronted in executing multimedia applications should be carefully considered in developing a DVS technique. In the next section, we propose an M-DVS technique that takes the memory wall problem into consideration.

3 Memory-aware dynamic voltage scaling

This section describes the M-DVS policy that specifically considers the memory wall problem. Fig. 4 illustrates the operational principles of M-DVS and conventional DVS. In the figure, f_{proc}^{ub} and f_{sd}^{ub} denote the maximum operating frequencies of the processor and the memory, respectively. In conventional DVS, the operating point starts at $(U_{sys}(i), f_{proc}(i)) = (a, a')$ and, as time advances, it reaches (c, c') via (b, b'). The conventional technique acknowledges that the system utilisation is saturated at (c, c'). In reality, however, the processor utilisation has been over-estimated in this region P_m where the memory performance is restricted because of the memory wall problem. Here, recognising the system utilisation in (c, c') is just a reflection of the memory wall phenomenon (via M-DVS), an additional power reduction can be achieved by moving the processor frequency further down to (d, d').

Before discussing the M-DVS policy, we first illustrate how M-DVS can detect the memory wall problem in practice. Knowing that the application is memory-bound, the key idea is to monitor the number of data cache misses in the processor, for a unit time interval, and uses it for the metric. Recent processors used for handheld devices are usually equipped with a capability of monitoring internal component behaviour in software to analyse the performance of the processor. This capability is normally named the performance monitoring unit (PMU) and it enables the real-time monitoring of the data cache, instruction cache, translation lookaside buffer (TLB) and so on. In our work,

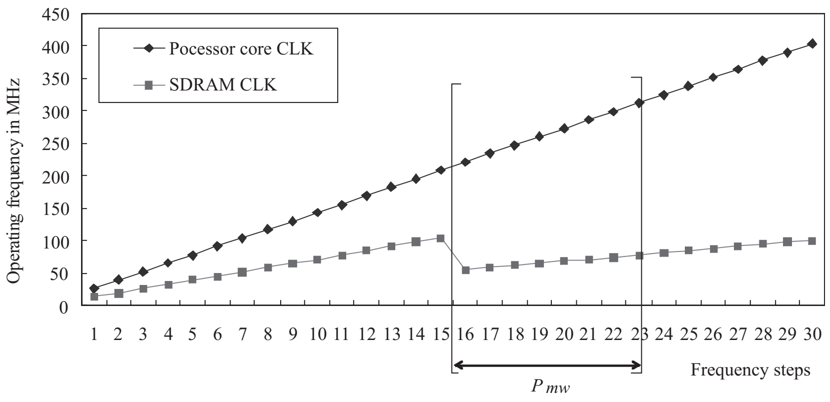


Fig. 2 *Operating frequencies of processor and memory*

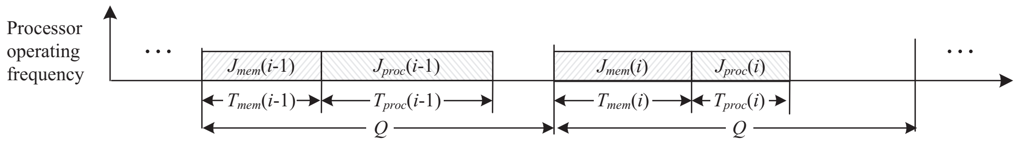


Fig. 3 *Processor-bound and memory-bound jobs in a periodic process*

the number of data cache misses $D^{\text{miss}}_{\text{cache}}(i)$ is obtained at run-time via the PMU of the PXA270 processor, and the statistics are used to detect the memory wall problem in real applications. $D^{\text{miss}}_{\text{cache}}(i)$ denotes the number of data cache misses occurring during the ith period. $D^{\text{miss}}_{\text{cache}}(i)$ ranges between $D^{\text{lb}}_{\text{cache}}$ and $D^{\text{ub}}_{\text{cache}}$, which are the experimental upper and lower bounds, respectively.

To show the suitability of the metric, $D^{\text{miss}}_{\text{cache}}(i)$ is experimentally monitored in the same environment as described in Fig. 1 and the results are shown in Fig. 5. Here, $D^{\text{miss}}_{\text{cache}}(i)$ changes sharply in the region between 208 and 286 MHz, where the memory wall problem occurred in the previous experiment in Fig. 1. The 'hit-under-miss' functionality of the PXA270 processor generates different $D^{\text{miss}}_{\text{cache}}(i)$ values for different combinations of the processor and memory frequencies, in decoding the same clip. The hit-under-miss functionality enables cache access even in the case of cache miss. With this functionality, $D^{\text{miss}}_{\text{cache}}(i)$ increases sharply in the memory wall section because the cache access module is synchronised with the processor clock, whereas the cache content is filled by external memory that is slower than the processor. On the basis of this observation, we adopt $D^{\text{miss}}_{\text{cache}}(i)$ as a metric that detects the memory wall problem.

Using $D^{\text{miss}}_{\text{cache}}(i)$, the memory wall problem is now detected and consequently an effective DVS policy can be devised. In normal situations, M-DVS works similar to a conventional DVS, but upon detecting the memory wall problem, the policy further reduces the operating frequency of the processor and achieves an additional power reduction. This is possible because the system utilisation has wrongly been overestimated. Implementing M-DVS in a real system requires an appropriate mapping of the system utilisation to the valid operating frequencies of the processor. In our scheme, n different operating frequencies are chosen and each frequency is associated to the system utilisation $U_{\text{sys}}(i)$. Fig. 6 illustrates the concept. Here, K_{seg} is a constant (in percentage) that determines the segment size of each $U_{\text{sys}}(i)$.

The size of the $U_{\text{sys}}(i)$ segment determined by K_{seg} has a significant impact on the degree of power consumption in the DVS algorithm and, therefore finding an optimal K_{seg} is important in achieving a maximum power reduction. It is, however, non-trivial to find an optimal value analytically, because the processing requirement and the memory access are subject to change dynamically because of the non-linearity of the interactions among the cache memory, TLB, the pre-fetch module and the pipeline at run-time. For practical reasons, we have used a heuristic mechanism to determine K_{seg} by adjusting the value through a series of DVS experiments. This generates a minimum energy index ε. The energy index ε is defined as the normalised value

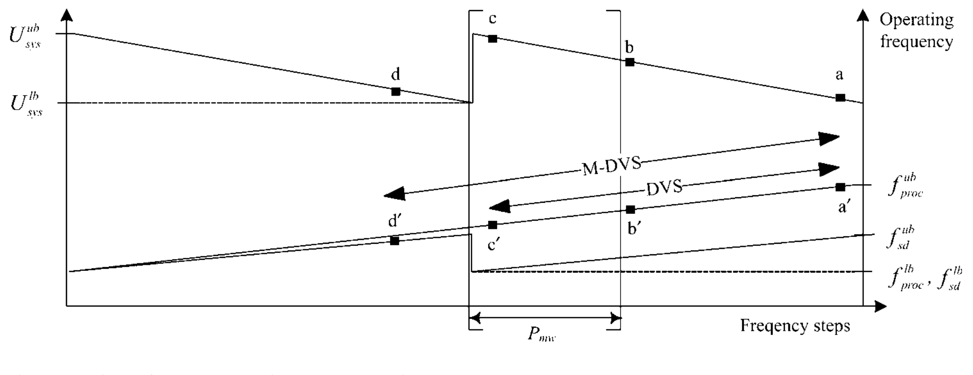


Fig. 4 *Operational principles of M-DVS and conventional DVS*

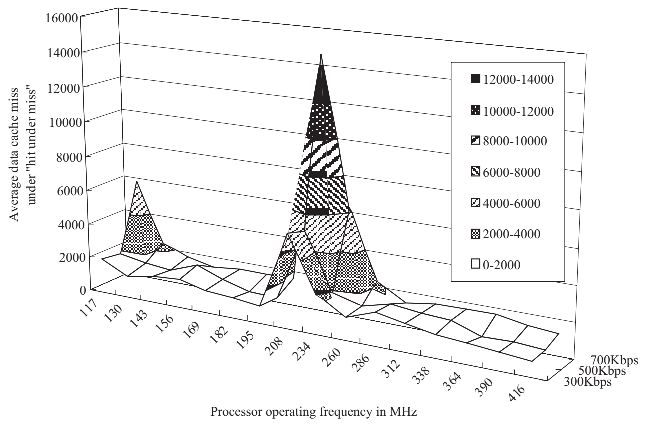


Fig. 5 *Average number of data cache miss*

of voltage settings upon performing the DVS operation, given the number of voltage setting segments, that is, $\varepsilon = \sum_{i=1}^{s} N(i) * C_{fv} * f_{\text{proc}}(i) / \sum_{i=1}^{s} N(i)$, where $N(i)$ is the number of processor setting occurrences for $f_{\text{proc}}(i)$ and C_{fv} is a constant that converts $f_{\text{proc}}(i)$ to the associated voltage. To prevent quality of service (QoS) degradation with DVS, we have conducted the experiments by increasing the portion of high performance frequency by varying K_{seg} from 25 to 75%. As shown in Fig. 6, the heuristic method finds an appropriate value without QoS degradation because the portion of $f_{\text{proc}}^{\text{ub}}$, which is the highest frequency of the processor, increases with the higher value of K_{seg}. This approach is based on the fact that the bursty workloads in MPEG require more portion of the highest operating frequency to have minimal energy consumption without QoS degradation [6]. Next, we devise a heuristic to determine, by comparing with $D_{\text{cache}}^{\text{miss}}(i)$ at run-time, the threshold $D_{\text{cache}}^{\text{thr}}$ that is used to detect the memory wall property of a task. This is similar to the one used for finding K_{seg} and finds $D_{\text{cache}}^{\text{thr}}$ that has a minimum ε without QoS degradation.

$$f_{\text{proc}}(i) = \frac{j^* f_{\text{proc}}^{\text{ub}}}{n}, \; U_{\text{sys}}(i-1) \in \left[\frac{(j-1)^*(100 - K_{\text{seg}})}{n-1}, \frac{j^*(100 - K_{\text{seg}})}{n-1} \right), \quad \text{for } j = 1, \ldots, n \tag{1}$$

$$f_{\text{proc}}(i) = f_{\text{proc}}(i) - K_g, \quad \text{if } U_{\text{sys}}(i-1) = U_{\text{sys}}^{\text{ub}} \text{ and } D_{\text{cache}}^{\text{miss}}(i-1) > D_{\text{cache}}^{\text{thr}} \tag{2}$$

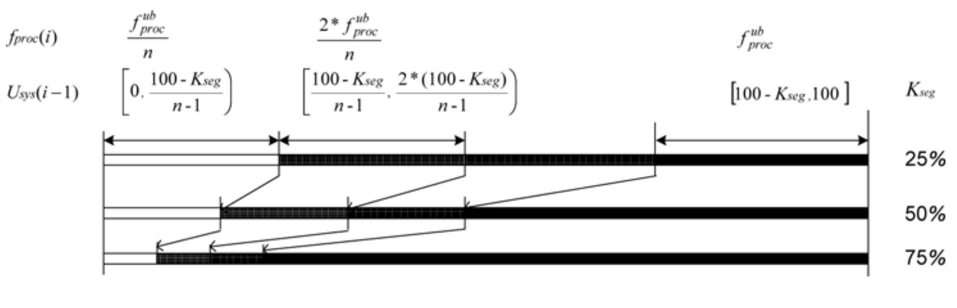
On the basis of K_{seg} and $D_{\text{cache}}^{\text{thr}}$, the M-DVS policy is expressed in (1) and (2). In (1), the next frequency $f_{\text{proc}}(i)$ of the processor is determined by the current system utilisation $U_{\text{sys}}(i-1)$. As a processor can have only discrete operating points, M-DVS selects an appropriate operating point in the range between $f_{\text{proc}}^{\text{ub}}/n$ and $f_{\text{proc}}^{\text{ub}}$, according to the current system utilisation $U_{\text{sys}}(i-1)$ and K_{seg}. After the initial setting of $f_{\text{proc}}(i)$, (2) adjusts $f_{\text{proc}}(i)$ by checking the occurrence of the memory wall problem. That is, if $D_{\text{cache}}^{\text{miss}}(i-1)$ is larger than $D_{\text{cache}}^{\text{thr}}$ and $U_{\text{sys}}(i-1)$ reaches its maximum, the system is alerted to experience the memory wall problem and the clock frequency is further adjusted to achieve an additional power reduction. In this case, M-DVS re-adjusts the processor frequency to $f_{\text{proc}}(i) - K_g$, where K_g is the variable that moves $f_{\text{proc}}(i)$ to the position d in Fig. 4.

4 Experiments

This section evaluates the performance characteristics of M-DVS on the basis of real experiments. The experimental platform is the Intel PXA270 processor development kit running the Linux kernel 2.4.19. Fig. 7 illustrates the M-DVS implementation that consists of multiple functional modules based on the layering concept. Both the policy module and the monitor module reside in the kernel. The monitor module consists of two parts: the utilisation monitor that measures and calculates the system utilisation and the memory behaviour monitor that monitors the data cache. The M-DVS policy module determines the next frequency of the processor on the basis of the data gathered from those two units, and the next processor setting is passed to the hardware through the API layer. We minimise the operating overhead by implementing the M-DVS policy module in the context of the process scheduler in the kernel. Also, system portability and modularity are achieved by accessing the low-level hardware through a well-defined set of API functions. For the actual power measurement, the Agilent data logger 34970A/34901A [17] is used. Simultaneous monitoring of both core voltage and current consumption is conducted to increase the measurement accuracy of the actual power consumption.

Now, we experimentally demonstrate that the proposed M-DVS system reduces more power consumption than the conventional DVS, without QoS degradation. The process of finding thresholds for both conventional DVS and M-DVS is discussed first. The functional behaviour of M-DVS is then validated by showing the frequency distributions generated by conventional DVS and M-DVS. The last experiments prove that M-DVS achieves an additional power reduction, compared with conventional DVS, by showing the actual power consumption.

Table 1 shows the process of finding the optimal value of K_{seg} for the cases of decoding MPEG clips of 300, 500 and 700 kbps with Mplayer. Given four different kinds of operating frequency, M-DVS selects the K_{seg} value that induces the lowest ε without missing the deadline, in effect the actual decoding time (TS) does not surpass the deadline. For example, in the case of 300 kbps, $K_{\text{seg}} = 55\%$ is selected instead of 25%, because TS for $K_{\text{seg}} = 25\%$ surpasses the deadline although it has the lowest ε. Table 2



$f_{proc}(i)$
$\frac{f_{proc}^{ub}}{n}$
$\frac{2*f_{proc}^{ub}}{n}$
f_{proc}^{ub}
$U_{sys}(i-1)$
$\left[0, \frac{100-K_{seg}}{n-1}\right)$
$\left[\frac{100-K_{seg}}{n-1}, \frac{2*(100-K_{seg})}{n-1}\right)$
$[100-K_{seg}, 100]$
K_{seg}
25%
50%
75%

Fig. 6 *Utilisation based on the K_{seg} threshold*

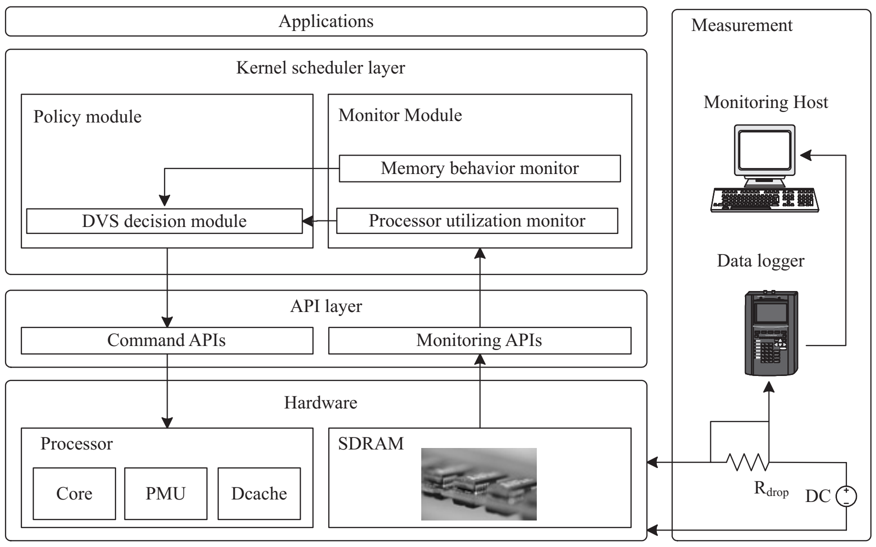


Fig. 7 *M-DVS system and its power measurement*

Table 1: Determining K_{seg}

Bit rate, kbps	K_{seg}(%)	ε	TS, s	Deadline
300	25	1.311	33.6	Miss
	40	1.321	32.4	–
	55	1.319	32.8	–
	75	1.325	32.6	–
500	25	1.373	32.7	–
	40	1.451	30.6	–
	55	1.465	31.1	–
	75	1.459	31.3	–
700	25	1.401	32.5	–
	40	1.405	30.9	–
	55	1.404	31.2	–
	75	1.421	31.4	–

Table 2: Determining D^{thr}_{cache}

Bit rate, kbps	D^{thr}_{cache}	ε	TS, s	Deadline
300	5000	1.196	31.1	–
	10 000	1.188	31.6	–
	15 000	1.202	31.1	–
	20 000	1.196	31.0	–
500	5000	1.238	32.0	–
	10 000	1.309	30.6	–
	15 000	1.297	30.5	–
	20 000	1.264	30.8	–
700	5000	1.148	35.7	Miss
	10 000	1.146	30.7	–
	15 000	1.176	31.1	–
	20 000	1.117	30.9	–

shows the process for finding D^{thr}_{cache} for the same clips. Similar to the case of K_{seg}, D^{thr}_{cache} is selected upon the condition that the power consumption is minimised without missing the deadline. The large variation of optimal values is because of the fact that the processor used in our experiment is based on the 'out-of-order processor' architecture, which induces unpredictable recursive cache access [18].

On the basis of the thresholds obtained by the heuristic search, Fig. 8 shows the run-time frequency distributions of the processor with three different policies, No-DVS, DVS and M-DVS, for decoding a 300 kbps video clip. Note that No-DVS does not use any DVS policy and DVS is a conventional policy based on the processor utilisation, without considering the memory wall effect. For a fair comparison, both DVS and M-DVS use the same K_{seg}. The experimental results show that because of the low compression rate of the clip, both DVS and M-DVS select operating frequencies below 104 MHz for the most part. It is apparent in the figure that M-DVS shifts the frequency settings of [104, 156] MHz of DVS to 78 MHz (57%) and also [312, 390] MHz to lower frequencies. It is also observed that a small portion of the M-DVS frequency is set to the maximum frequency of the processor, 416 MHz, in order to compensate for possible QoS degradation by transition to a lower frequency. In summary, M-DVS is shown to lower the processor frequency when detecting the memory wall problem, and sets the maximum operating frequency to prevent a possible QoS degradation caused by the reduced frequency.

Fig. 9 shows the traces of power consumption that correspond to the processor frequencies determined by the three policies. The top graph shows the overall performance, and the bottom graph shows the magnified view on [100, 153] segment to observe whether M-DVS properly detects the memory wall problem. Here, the moving averages of the two latest samples are presented both for DVS and M-DVS. The top graph clearly shows that both DVS and M-DVS reduce the power consumption compared with No-DVS whose trace of the power consumption is dispersed higher than 150 mW. Also shown is that M-DVS consumes less power than DVS because most power consumption measurements of M-DVS ranges from 50 mW to 60 mW, compared with that of DVS which is in the range from 50 to 100 mW. In the bottom figure, two segments ([108, 125] and [128, 139]) validate that M-DVS works as expected and achieves an additional power reduction by shifting the processor frequency appropriately with the memory wall compensation. The figure also shows that M-DVS sets the highest processor performance to cope

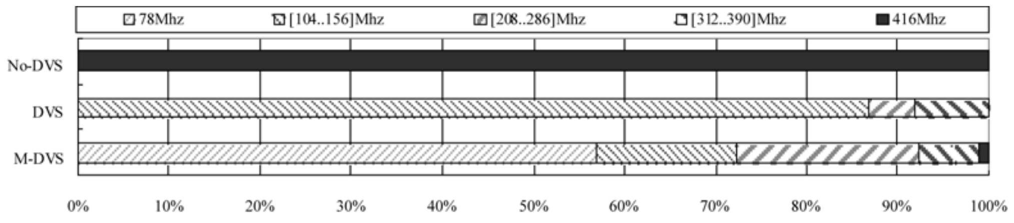


Fig. 8 *Frequency distributions of different DVS policies with 300 kbps clip*

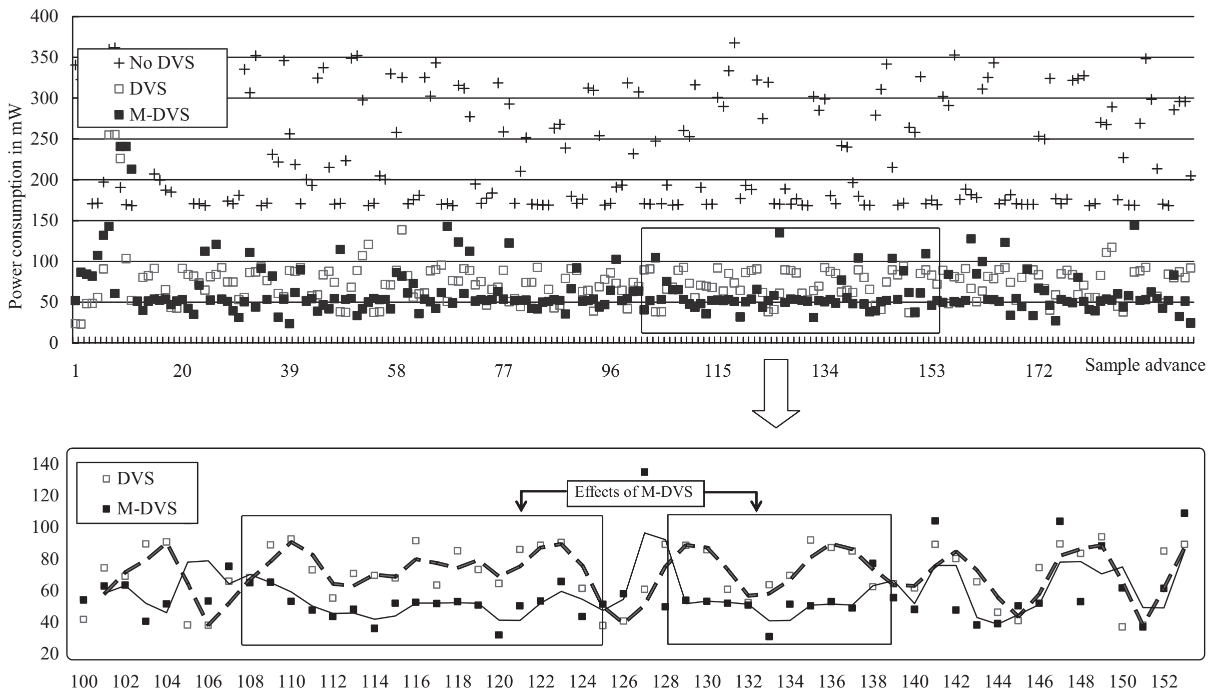


Fig. 9 *Power consumption traces with 300 kbps clip*

Table 3: Power consumption summary

MPEG workloads, kbps	Policies No-DVS	DVS	M-DVS
300	241	75	59
500	267	103	78
700	250	99	82

Average power consumption in milliwatts.

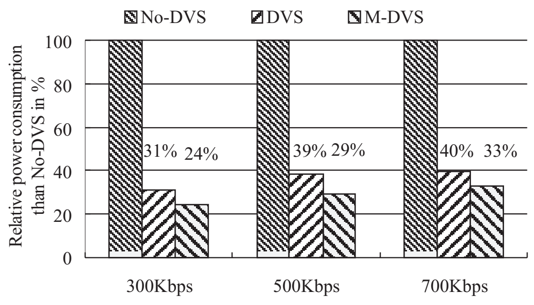


Fig. 10 *Relative power consumption*

with possible performance loss in compensating for the memory wall problem. This is observed in the sample instances of 127, 141, 147 and 153.

The average power consumptions of the processor for three different videos under the same experimental environment are summarised in Table 3. Fig. 10 shows the relative power consumption of DVS and M-DVS compared with No-DVS. DVS and M-DVS save, respectively, ~63 and 71% power consumption on average compared with No-DVS. Also, M-DVS saves ~8% additional power consumption compared with DVS. To avoid QoS degradation, the experiment considers the QoS factors, such as dropped frames, decoding time and deadline, in selecting thresholds. The experimental results show that there are neither dropped frames nor deadline misses monitored with our experimental setup and hence the proposed M-DVS satisfies the application's QoS requirements.

5 Conclusions

Although there are many DVS-related studies on the efficient utilisation of limited battery resources, DVS methods considering only the processor encounter the memory wall problem when executing multimedia applications. This paper experimentally demonstrates that the memory wall problem exists in a real system and proposes M-DVS, a DVS technique, that effectively detects the memory wall problem at run-time and further reduces power consumption using an M-DVS policy. The net result is that M-DVS saves ~8% power consumption when compared with a conventional DVS.

The main contribution of our work is as follows. First, we experimentally showed that the memory wall problem exists in a real handheld device when executing memory-bound applications, such as multimedia playback, and that conventional DVS techniques fail to optimise power reduction as a result. We then redefined the 'processor utilisation' used in it conventional DVS as the system utilisation and constructed a supplementary algorithm, on the top of conventional DVS, which enables an additional power reduction. Our work shows that the behaviour of main memory should be considered in developing an efficient DVS policy within a certain category of applications, and our experimental results should give insights to any practitioner in the related field. In addition, the design factors observed and analysed from the viewpoint of the operating system and power consumption can be reflected in SoC design process. Recently, multimedia applications such as streaming services have become dominant applications for handheld devices such as portable media player or cellular phones. Our research results can be easily applied to those devices, as the platform used in our work adopts a similar processor and memory structure.

Our mechanism needs to be extended to cover the possibility of multiple memory wall points because the speed of processors used in the handheld platforms tends to increase sharply compared to DRAM. In addition, considering the fact that the DMA architecture of embedded processors affects the main memory performance, further study is required to understand the power consumption characteristics of the DMA peripherals in the context of processor DVS. This will be part of our future work.

6 Acknowledgments

This work was supported by the Korea Research Foundation Grant (KRF-2003-041-D00475).

7 References

1 Weiser, M., Welch, B., Demers, A., and Shenker, S.: 'Scheduling for reduced CPU energy'. Proc. First Symp. on Operating Systems Design and Implementation, Monterey, CA, November 1994, pp. 13–23

2 Govil, K., Chan, E., and Wasserman, H.: 'Comparing algorithms for dynamic speed-setting of a low-power CPU'. Proc. Int. Conf. on Mobile Computing and Networking, Berkeley, CA, November 1995, pp. 13–25

3 Nurvitadhi, E., Lee, B., Yu, C., and Kim, M.: 'A comparative study of dynamic voltage scaling techniques for low-power video decoding'. Proc. Int. Conf. on Embedded Systems and Applications, Las Vegas, NV, June 2003, pp. 292–298

4 Pouwelse, J., Langendoen, K., and Sips, H.: 'Application-directed voltage scaling', *IEEE Trans. VLSI Syst.*, 2003, **11**, (5), pp. 812–826

5 Mohapatra, S., Cornea, R., Dutt, N., Nicolau, A., and Venkatasubramanian, N.: 'Integrated power management for video streaming to mobile handheld devices'. Proc. 11th ACM Int. Conf. on Multimedia, Berkeley, CA, November 2003, pp. 582–591

6 Grunwald, D., Levis, P., Morrey, C.B. III, Neufeld, M., and Farkas, K.I.: 'Policies for dynamic clock scheduling'. Proc. 4th Symp. on Operating Systems Design and Implementation, San Diego, CA, October 2000

7 Martin, T.L., and Siewiorek, D.P.: 'Non-ideal battery and main memory effects on CPU speed-setting for low power', *IEEE Trans. VLSI Syst.*, 2001, **9**, (1), pp. 29–34

8 Martin, T.L., Siewiorek, D.P., Smailagic, A., Bosworth, M., Ettus, M., and Warren, J.: 'A case study of a system-level approach to power-aware computing', *ACM Trans. Embedded Comput. Syst.*, 2003, **2**, (3), pp. 255–276

9 Fan, X., Ellis, C.S., and Lebeck, A.R.: 'The synergy between power-aware memory systems and processor voltage scaling'. Proc. Power Aware Computer Systems, San Diego, CA, December 2003, pp. 164–179

10 Weissel, A., and Bellosa, F.: 'Process cruise control-event-driven clock scaling for dynamic power management'. Proc. Compilers, Architectures and Synthesis for Embedded Systems, Grenoble, France, October 2002, pp. 238–246

11 Choi, K., Soma, R., and Pedram, M.: 'Off-chip latency-driven dynamic voltage and frequency scaling for an MPEG decoding'. Proc. 41st Design Automation Conf., San Diego, CA, June 2004, pp. 544–549

12 Cho, Y., and Chang, N.: 'Memory-aware energy-optimal frequency assignment for dynamic supply voltage scaling'. Proc. Int. Symp. on Low Power Electronics and Designs (ISLPED 2004), Newport Beach, CA, August 2004, pp. 387–392

13 Hennessy, J., and Patterson, D.A.: 'Computer architecture - A quantitative approach' (Morgan Kaufmann Publishers, 2003, 3rd edn.)

14 Wulf, W., and McKee, S.: 'Hitting the memory wall: implications of the obvious', *Comput. Archit. News*, 1995, **23**, (1), pp. 20–24

15 Available online at http://www.mplayerhq.hu

16 Intel: Intel PXA27x processor family, available at http://developer.intel.com/design/pca/prodbref/253820.htm, accessed 21 February 2005

17 Agilent: 34970A data acquisition switch unit, available at www.agilent.com, accessed 21 February 2005

18 Srinivasan, S.K., and Velev, M.N.: 'Formal verification of an Intel XScale processor model with scoreboarding, specialized execution pipelines, and impress data-memory exceptions'. Proc. First ACM and IEEE Int. Conf. on Formal Methods and Models for Co-design, Mont Saint-Michel, France, June 2003, pp. 65–74