Dynamic Voltage Scaling of Supply and Body Bias Exploiting Software Runtime Distribution

Sungpack Hong
EE Department
Stanford University

Sungjoo Yoo, Byeong Bin,
Kyu-Myung Choi, Soo-Kwan Eo
Samsung Electronics

Taehwan Kim
EECS
Seoul National University

ABSTRACT

This paper presents a method of dynamic voltage scaling (DVS) that tackles both switching and leakage power with combined V_{dd}/V_{bs} scaling and gives minimum average energy consumption exploiting the runtime distribution of software execution. We present a mathematical formulation of the DVS problem and an efficient numerical solution. Experimental results show that the presented method shows up to 44% further reduction in energy consumption compared with existing methods. Especially, when the leakage power consumption is significant, i.e. when temperature is high, the presented method is proven to be the most effective.

1. Introduction

Dynamic voltage scaling (DVS) is one of the most effective methods in reducing both switching and leakage power consumption. There have been two classes of DVS methods: inter-task and intra-task DVS. Inter-task DVS methods [1][2] determines the performance level at a task granularity while intra-task DVS methods at finer granularities [3][4][5].

In intra-task DVS, workload estimation plays a central role since the performance level (normalized w.r.t. maximum frequency) in the middle of task execution is dynamically determined, mostly, by *X*/*T*, where *X* is the estimated remaining workload and *T* is the time to deadline. Thus, the accuracy of workload estimation determines the quality of intra-task DVS method.

Several methods of workload estimation have been proposed: worst case execution time [3][4], average case execution path [5], average energy execution path [6], and statistical methods [7]. Among them, the statistical method and average energy execution path-based one are reported to give the best reduction in average switching energy consumption since they provide global minimum solutions based on mathematical formulations. However, the leakage power consumption is not minimized by the methods since they minimize only the switching energy based on the assumption of $P \sim f^3$ ($P \sim CV^2f \sim f^3$ since $V \sim f$).

Leakage power consumption has already become a real design issue. Especially, excessive leakage power consumption at high temperatures often causes significant product parametric yield drop in reality[1]. Thus, DVS methods need to optimize leakage energy as well as switching energy.

In order to reduce leakage energy consumption, we apply combined V_{dd}/V_{bs} scaling [10][11] since body biasing (scaling V_{bs}) is the most effective way to control leakage power consumption. In our work, we extend the statistical DVS method (which originally targets only dynamic energy) to tackle the reduction of both switching and leakage energy by scaling both V_{dd} and V_{bs}. Note that the statistical method covers the method based on average energy execution path [6] as a simplified case.

We give a mathematical formulation of the problem of V_{dd}/V_{bs} scaling based on the statistical information, i.e. the distribution of software runtime. The formulation gives a multi-variable non-linear function of total energy consumption. As a practical solution to obtain the workload estimations for the minimum average energy consumption, we present a numerical solution.

This paper is organized as follows. Section 2 reviews existing DVS methods. Section 3 explains the mathematical formulation of statistical DVS based on combined V_{dd}/V_{bs} scaling. Section 4 gives a total power function for combined V_{dd}/V_{bs} scaling. Section 5 presents a numerical solution to the problem. Section 6 reports experimental results and Section 7 concludes the paper.

2. Related Work

In [3], an intra-task DVS method called runtime voltage hopping is presented to exploit workload variation to reduce the energy consumption. In this work, the workload variation is a slack which is calculated, at the hopping point, as the difference between the expected worst-case execution time and the real program runtime of already executed software code. In [4], the remaining workload is estimated to be the execution cycle of worst-case execution path from a performance setting point in the software program to the end of program. The execution cycle of average-case execution path is estimated to be the remaining workload in [5]. The concept of virtual execution path is presented in [6] to estimate workloads for minimum average energy consumption. The method uses the worst-case execution cycles of remaining basic blocks to predict the remaining workload assuming $P \sim f^3$. In [13] and [14], methods of DVS based on combined V_{dd}/V_{bs} scaling are presented. In these works, the relationship between power and frequency can be an arbitrary one. However, those works do not consider the runtime distribution of software execution, but is based on the worst-case execution cycle.

The above methods have a common assumption in estimating the remaining workload. They all assume the worst-case execution cycle (of the entire remaining program or of each basic block). However, in reality, it is rare to encounter the worst-case execution. Since minimizing energy consumption is mostly an optimization problem for average cases (e.g. the battery lifetime of mobile device is mostly evaluated in an average sense after running an extensive set of benchmarks and use cases), it is required to tackle DVS problems statistically to reduce average energy consumption while meeting the given deadline constraints. In [7], a statistical method based on the runtime distribution of software execution cycle (not

[1] Although the power consumption specification can be met at room temperature, it cannot often be met due to significant leakage power consumption at high temperatures in the product specifications, e.g. 80 or 125℃.

978-3-9810801-3-1/DATE08 © 2008 EDAA

the worst-case execution cycle) is presented. This method enables to obtain the estimation of remaining workload that yields the minimum average energy consumption. However, this method is based on the assumption of $P \sim f^3$. Thus, it does not minimize the entire energy consumption, especially, when the leakage power is not negligible.

3. Mathematical Formulation of Statistical DVS based on Combined V_{dd}/V_{bs} Scaling

Figure 1 illustrates the intra-task DVS based on runtime distribution. We divide the entire software program into chunks of code called program regions (PR's), shown as rectangles in the figure, and set operating frequency at the beginning of each PR (called performance setting point) based on the estimation of remaining workload, X. For instance, in the beginning of program region N_0 in the figure, we set frequency to be X_0/T where X_0 is the estimated remaining workload and T the time to deadline.

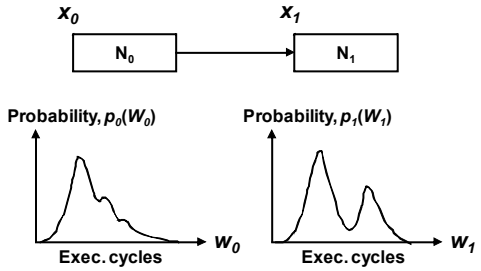


Figure 1 Intra-task DVS with runtime distribution

Each program region has a distribution of its runtime (execution cycles), i.e. a probability distribution function (PDF) as illustrated in Figure 1[2]. The distribution results from both data dependency (e.g., data dependent number of loop iterations) and underlying hardware architecture (e.g., cache misses, variable latency instructions, etc.). In this case, our problem is to calculate the estimated workload x_0 that will give the minimum average energy consumption of all the remaining program regions.

In [7], the authors formulate the problem mathematically with two assumptions. The first assumption is $P \sim f^3$. The second assumption is the independence between X_0 and X_1. Thus, X_0 can be calculated assuming that X_1 has been already obtained (thus, assuming that it is a constant) in the bottom-up traversal of PR's from the leaf PR (i.e. the end of program). At each program region, x_0 is obtained by solving $\delta E(X_0)/\delta X_0 = 0$. However, when applying combined V_{dd}/V_{bs} scaling, the two assumptions do not hold any more. The average energy consumption function $E(X_0,X_1)$ becomes more complicated as follows (the details are omitted for page limit).

$$E(X_0,X_1)=\int\left(\begin{array}{l} P(\frac{X_0}{T})\cdot\frac{w_0}{X_0}\cdot T\cdot p_0(w_0)+ \\ \int P(\frac{X_1X_0}{T(X_0-w_0)})\cdot\frac{w_1(X_0-w_0)}{X_0X_1}\cdot T\cdot p_1(w_1)dw_1 \end{array}\right)dw_0 \quad (1)$$

where $P(f)$ is the total power function (when V_{dd}/V_{bs} scaling is applied) with a normalized frequency f as the input argument, $p_0(w_0)$ and $p_1(w_1)$ are the PDF's of program regions N_0 and N_1 as exemplified in Figure 1. In Eqn. (1), the total power function $P(f)$ is no longer a simple function (such as $P \sim f^3$), but more complicated one (which will be presented in Section 4). In Eqn. (1), in order to obtain the minimum average energy consumption E, X_0 cannot be determined independently from X_1. They need to be determined simultaneously. In the case of n program regions, n workload estimations $\{ X_0, X_1, \ldots, X_n \}$ for the minimum average energy consumption need to be obtained simultaneously. Thus, the original bottom-up traversal method in [7] cannot be applied in this case.

In summary, given n program regions (and their PDF's), we need to find a set of workload estimations $\{ X_0, X_1, \ldots, X_n \}$ giving the global minimum of average energy consumption. Since the total power function is not an analytical function, we need numerical solutions to find the global minimum. In this paper, we present a numerical solution to this problem.

4. Total Power Function in Combined V_{dd}/V_{bs} Scaling

This section presents the total power function and its dependency on temperature when combined V_{dd}/V_{bs} scaling is applied. Given a frequency requirement, we can obtain a pair of V_{dd} and V_{bs} values that give the minimum total (switching and leakage) power consumption. In this section, we present our total power function and associated pair of V_{dd} and V_{bs} values following the method in [3].

Switching power P_{SW} is calculated as follows.

$$P_{SW} = C_{eff} * V_{dd}^2 * f \quad (2)$$

where the effective capacitance C_{eff} is given in Table 1. Leakage power P_{Leak} is obtained as follows.

$$P_{Leak} = (V_{dd} * I_{sub} + |V_{bs}| * I_j) * L_g \quad (3)$$

where the subthreshold current I_{sub} is given as

$$I_{sub} = K_3 * e^{(K4 * Vdd)} * e^{(K5 * Vbs)} \quad (4)$$

The parameters I_j, L_g, K_3, K_4, and K_5 are given in Table 1. The threshold voltage V_{th} and inverter delay t_{inv} is modeled as follows.

$$V_{th} = V_{th1} - K_1 * V_{dd} - K_2 * V_{bs} \quad (5)$$

$$t_{inv} = L_d * K_6 / (V_{dd} - V_{th})^{alpha} \quad (6)$$

The operating frequency f is determined to be $1/(L_d * t_{inv})$, where L_d is the logic depth of critical path. Table 1 summarizes all the parameters used when deriving the total power function for combined V_{dd}/V_{bs} scaling based on the method and parameters in [9][10].

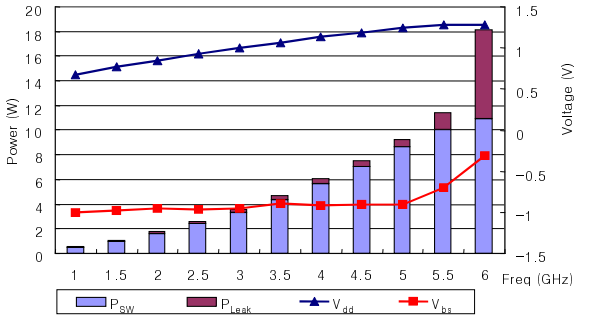
Table 1 Power parameters [9][10]

K_1	0.163	K_6	5.26e-12	V_{th1}	0.244
K_2	0.153	K_7	-0.144	I_j	4.8e-10
K_3	5.38e-7	V_{dd0}	1	C_{eff}	1.11e-9
K_4	1.83	V_{bs0}	0	L_d	35
K_5	4.19	$alpha$	1.5	L_g	4e6
C_r	1e-6	C_s	4e-6		

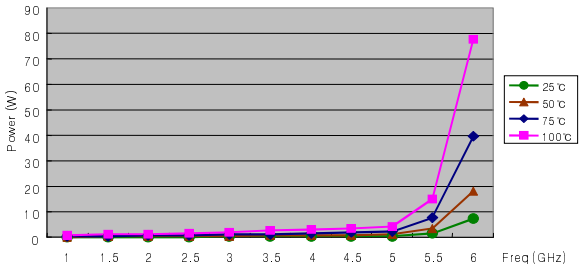
Figure 2 (a) shows the total power function and the corresponding pair of V_{dd} and V_{bs} values. V_{dd} (V_{bs}) ranges between 0.5v and 1.28v (-1.0v and 0v)[3]. As shown in the figure, the total power is not an analytical function of frequency.

[2] In reality, the X-axis of PDF is quantized into multiple bins as explained in Section 5.

[3] Note that the frequency range is between 1GHz and 6GHz. This range is higher than the critical frequency in [3].



(a) Total power function at 50C



(b) Temperature dependency of total power consumption

Figure 2 Total power function in combined V_{dd}/V_{bs} scaling

Figure 2 (b) shows the trend of total power consumption as temperature varies. We model the temperature dependency of leakage power as follows [12].

$$I(Temp) = I_s * \exp(- A / (Temp + 273 - B)) \quad (7)$$

where I_s is the leakage power at room temperature and parameters A and B are 606.53 and 192.02, respectively [12].

In our experiments, we use the total power function presented in this section. Note that the presented method in Section 5 does not necessarily depend on the total power function presented in this section. Thus, any total power functions, e.g., those obtained from slower processors or real measurements, can be used in the presented method.

5. Presented Numerical Method

In this section, first we explain our terminology, a numerical solution based on an iterative improvement, and how to calculate energy consumption with arbitrary total power function under software runtime distribution.

5.1 Terms and Notations

Node: Since we divide the whole program into a set of program regions, we consider the entire program as a graph where nodes correspond to program regions and arcs to the control dependency between nodes. Thus, we use two terms, node and program region, interchangeably. In our notations, N_i is the node with index i. N_0 is the node corresponding to the entry program region.

Runtime Distribution: We regard the runtime distribution of program region N_i a random variable w_i. We obtain the runtime distribution by extensive profiling to be explained in Section 6. In reality, we have the PDF of w_i as a function with M bins (M=32 in our experiments). w_i^k denotes the representing value (execution cycle) of k-th bin, and $p(w_i^k)$ is the probability of the bin.

Remaining Execution Cycles: We profile the maximum (WT_i), average (AT_i) and minimum (BT_i) numbers of execution cycles from the beginning of node N_i to the end of program.

Time to Deadline: T_i is a random variable representing the remaining time to deadline at the beginning of node N_i. By definition, the remaining time to deadline at the entry node T_0 equals to the given deadline D. The distribution of T_i is represented as a PDF with L bins (L=256 in our experiments). T_i^j is the representing value (time to deadline) of j-th bin, and $q(T_i^j)$ is the probability of the bin. Note that the distribution of T_0 is trivial as follows.

$$q(T_0^j) = \begin{cases} 1.0 & j = L \\ 0 & j = 1 \sim (L-1) \end{cases} \quad (8)$$

We explain how to calculate PDF $q(T_i^j)$ in Section 5.3.

Estimated Workload: We denote the estimated workload of node N_i as X_i, which is the variable we want to determine for each node so that the average energy consumption under runtime distribution is minimized for the whole program.

Performance Setting Point (PSP): The PSP is a code location where the performance level of processor is adjusted. Each program region has a PSP at its beginning. In terms of software code, we insert, at the PSP, a function for performance setting, PS() as follows[4].

```
PS(i) { // for node N_i
    T_i = Get_Time_to_Deadline();
    f_min = Get_Min_Required_ Freq(i,T_i);
    f = max(X_i/T_i, f_min); // X_i was calculated at design time.
    Set_Freq_Voltages(f); /* Adjust V_dd and V_bs as in Figure 2 */
}
```

As shown above, in order to meet the given deadline, function PS() first calculates the minimum required frequency f_{min} by calling function Get_Min_Required_Freq(). The details of this routine can be found in [7] and we omit the details here. Note that the presented DVS method satisfies the given deadline. In our work, we insert PSP's manually at the boundaries of compute-intensive loop iterations. Automatic insertion of PSP's will be an interesting topic and we will investigate it in our future work.

5.2 Workload Estimation

Now we present the numerical solution to obtain the set of workload estimation X_i's such that the average total energy consumption (as illustrated in Eqn. (1)) is minimized. Figure 3 gives a pseudo code of the presented solution.

```
1   Find_All_Workload_Estimations() {
2     X_i = WT_i ;   /* for all i = 0 ~ N-1 */   /* Initial Solution */
3     E_best = Calculate_Average_Energy();  /* In Section 5.4 */
4     Do {   /* Main Loop */
5       for i = 0 to N-1
6         { X_i = Find_Single_Workload_Estimation(i, BT_i, WT_i); }
7       E = Calculate_Average_Energy();
8       if (E < E_best) { E_best = E; Save_Current_Estimations(); }
9       else break;  /* No improvement in the while loop */
10    } while ( loop_count++ < MAX_COUNT)
11  }
```

Figure 3 An iterative solution

The basic idea of presented solution is that we tackle one variable (workload estimation of a node) at a time iteratively until there is no further reduction in average total energy consumption. First, all the workload estimations X_i's are set to the worst-case remaining

[4] Note that we take also voltage transition delay into account in our implementation of PS() as in [7].

execution cycles WT_i's as the initial solutions (line 2 in Figure 3). Then, the expected energy consumption for this solution is calculated as will be explained in Section 5.4 (line 3). In the main loop (lines 4–10), we obtain the workload estimation of each node that gives the minimum average energy consumption while assuming the other workload estimations are set to the current set of X_i's (line 6). Function Find_Single_Workload_Estimation(), which gives the workload estimation, will be explained later in this subsection. The function returns a new workload estimation X_i (line 6). Once all the nodes are processed, we calculate total energy consumption with new workload estimations (line 7), and update the best case if there is any improvement (line 8). This main loop is repeated for a given number of iterations (MAX_COUNT, line 10).

Figure 4 shows the pseudo code of function Find_Single_ Workload_Estimation(). The function sweeps candidate values of workload estimation within the given range of possible workload estimation (between minimum and maximum remaining cycles, i.e., BT_i and WT_i). For each candidate value, first we update the PDF's of time to deadline T_i for all the other remaining nodes (line 4 in the figure). It is because T_i's for the remaining nodes (nodes to be executed after node N_i) change depending on how much cycles node N_i spends. Thus, depending on the choice of X_i, the PDF's of time to deadline T_i's for all the remaining nodes need to be updated consequently. We explain how to update the PDF's of time to deadline T_i (in function Update_PDF_Time_to_Deadline_ Recursively()) in Section 5.3.

```
1   Find_Single_ Workload_Estimation(i, MIN, MAX) {
2    E_best = MAX_VAL;
3    for X_i = MIN to MAX step STRIDE {
4      Update_PDF_Time_to_Deadline_Recursively(i, X_i );
5      E = Calucate_Average_Energy();
6      if (E < E_best)  E_best = E;  X_best = X_i;
7    }
8    return X_best ; }
```

Figure 4 Function to find single workload estimation

Figure 5 illustrates an example result of such a sweep (lines 3—7 in Figure 4) to find the workload estimation. As shown in Figure 5, the sweep locates the workload estimation (inside the rectangle in the figure) giving the minimum average energy consumption.

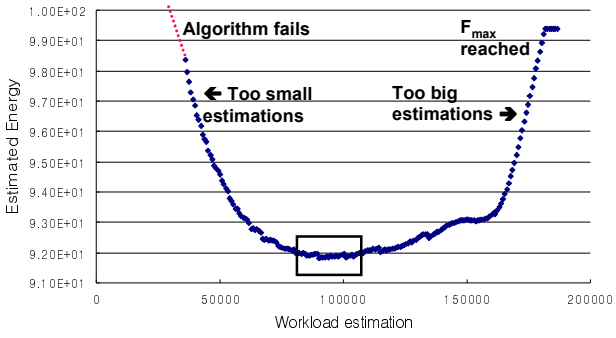


Figure 5 An example of single workload estimation

5.3 Derivation of the PDF's of Time to Deadline

The PDF of time to deadline of node N_0, T_0 is trivial as shown in Eqn. (8). Figure 6 shows the pseudo code of PDF derivation for the other nodes. Figure 7 illustrates the PDF updating. Assuming that the PDF updating is applied to two nodes N_0 and N_1 (N_1 starts to execute after N_0 finishes), the figure shows the runtime distribution (in PDF) of N_0, $p(w_0)$ and its PDF of time to deadline T_0, $q(T_0)$ in the upper part. The figure illustrates how to derive node N_1's PDF of time to deadline $q(T_1)$ from the two PDF's, $p(w_0)$ and $q(T_0)$. Suppose that X_0 = 12 in the loop of Figure 4 (lines 4—6) and that function Update_PDF_Time_to_Deadline_Recursively(0,12) is called (line 4 in Figure 4).

```
1   Update_PDF_Time_to_Deadline (i,X_i) {
2    T_{i+1}^l=0.0;   /*  l = 1 ~ L  */
3    for j = 1 to L {
4     for k = 1 to M {
5       T_used = w_i^k / (X_i/T_i^j ) ;
6       T_remain = T_i^j – T_used;
7       prob = q(T_i^j)*p(w_i^k);
8       l = Get_Bin_Index_T(T_remain);
9       T_{i+1}^l = T_{i+1}^l + prob;
10  } } }
11
12  Update_PDF_Time_to_Deadline_Recursively (i, X_i) {
13    for (m = i to N) {
14       Update_PDF_Time_to_Deadline(m, X_m)
15  } }
```

Figure 6 Functions to update the PDF, $q(T_i)$

Now we calculate the PDF of node N_1's time to deadline, $q(T_1)$. Since T_0^L = 12 as shown in the middle of Figure 7, we set frequency at N_0 to 1 (=X_0/T_0^L). Considering the execution cycle of node N_0, w_0, we can have different probabilities for different w_0^k values. For instance, the probability that node N_0 takes w_0^2 (= 3) clock cycles, $p(w_0^2=3)$ is 0.4 as shown in Figure 7. In this case, since node N_0 takes 3 cycles (=$w_0^2/(X_0/T_0^L)$ as in line 5 of Figure 6), only 9 cycles (=12-3) remains as the time to deadline for node N_1 (line 6 in Figure 6). Thus, the probability that T_1 has 9 cycles of time to deadline, $q(T_1^3=9)$ becomes 0.4 (=$p(w_0^2)$* $q(T_0^L)$ = 0.4*1.0) as the two arrows in Figure 7 illustrates (also, line 7 in Figure 6). All the probabilities of the other bins for T_1 are calculated in the same way.

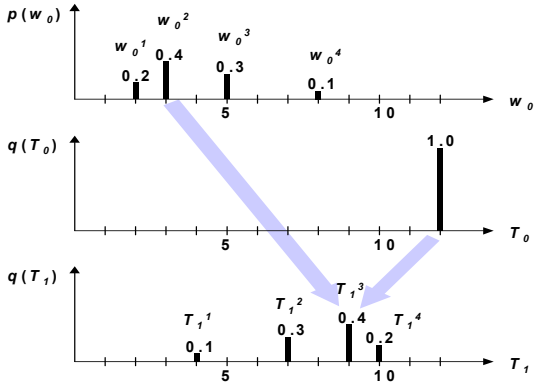


Figure 7 Updating the PDF of time to deadline

Based on function Update_PDF_Time_to_Deadline(), function Update_PDF_Time_to_Deadline_Recursively() in Figure 4 can be easily implemented as shown in Figure 6 (lines 12—15).[5]

5.4 Calculating Average Energy Consumption

When the workload estimations are once determined, function

[5] For simplicity, we assume a sequential chain of nodes in this explanation. However, in the case that there are multiple children or parents for a node (i.e. conditional branches), the above algorithm should be slightly modified to include branch probability and breadth-first iteration as in [7]. However, the extension is trivial and hence omitted for brevity here.

Calculate_Average_Energy() gives the average total energy consumption for the workload estimations. First, the energy consumption of node N_i can be calculated as follows.

$$\begin{aligned} Energy &= Power\ (freq) \cdot (time\ consumed\ by\ N_i) \\ &= P(\frac{X_i}{T_i}) \cdot (\frac{w_i T_i}{X_i}) \end{aligned} \quad (9)$$

where, X_i is a constant in this case while w_i and T_i are random variables with their PDF's, $p(w_i)$ and $q(T_i)$. The average energy consumption of node N_i , e_i can be calculated as follows.

$$e_i = \sum_{j=1}^{L}\sum_{k=1}^{M} P(\frac{x_i}{T_i^j}) \cdot (\frac{w_i^k T_i^j}{x_i}) \cdot p(w_i^k) \cdot q(T_i^j) \quad (10)$$

In consquence, the average total energy consumption of the whole program is calculated by summing the e_i's each multiplied by the probability of executing the corresponding node to account for conditional branches [7].

5.5 Consideration of Temperature Conditions

In order to consider that the power–frequency characteristic changes as temperature, we propose a simple approach to adapt the presented method to varying temperature conditions. First, we select a number of representative temperatures (e.g. 25, 50, 75, 100 ℃), and establish the total power function $P_K(f)$ for each of those cases. We make a set of workload estimations for each representative temperature. Note that these calculations are all done at design time. During the runtime, the DVS algorithm obtains temperature information by consulting the thermal sensor and chooses the appropriate set of workload estimations based on the current temperature. Then, it performs performance setting as explained above.

6. EXPERIMENTS

In our experiments, we assume the processor power model presented in Section 4. We assume discrete frequencies that range between 1GHz and 6GHz with 500MHz step. For each frequency step, a set of V_{dd}/V_{bs} is applied to give an optimal V_{dd}/V_{bs} scaling as explained in Section 4. From the set of discrete frequencies, we select a frequency level, which is the lowest but higher than or equal to the frequency calculated in function PS(), as the required performance level. We also prepare the power models at four different temperature conditions, 25, 50, 75, and 100℃.

Voltage transition time is assumed to be 200μs. The energy consumption of voltage transition, E_s is modeled as follows [4].

$$E_s = |\Delta V_{dd}|^2 * C_r + |\Delta V_{bs}|^2 * C_s \quad (11)$$

The runtime overhead of performance setting function call PS() and that of voltage/frequency transition are assumed to be 1k cycles and 50μs, respectively[6]. The delay overhead of voltage transition is also taken into account in function PS() as in [7] when checking whether the deadline constraint can be met.

When software execution finishes before the deadline, we apply power and/or clock gating. If the remaining time is less than 1ms, we apply only clock gating. Thus, in this case, leakage power is consumed from the end of software execution to the deadline. If the remaining time is longer than 1ms, we apply power gating, after the 1ms period of clock gating, until the deadline. We assume also that processor power gating takes 1ms.

We apply the presented method to four multimedia software applications: H.264 decode, MPEG4 decode/encode, and MP3 decode. We insert PSP's manually at the boundaries of sets of loop iterations in the source code of the applications as in [7][7]. We obtain the distribution of software runtime after running representative benchmarks for each application on the PC (Pentium4, 2.8GHz). For H.264 and MPEG4, we use the same benchmarks that are used in [7]. Table 1 gives the summary of applications. Note that we set practical deadlines on the applications to account for the real multi-task software execution environment. For instance, OS consumes a portion of processor cycles for its housekeeping operation, e.g. timer.

Table 2 Software programs used in the experiments

Application	# PSP's	Deadline
H.264 Decode (H.264)	5	30 ms (33 fps)
MPEG4 Decode (MPEG4-d)	5	30 ms (33 fps)
MPEG4 Encode (MPEG4-e)	10	40 ms (25 fps)
MP3 Decode (MP3)	6	20 ms[8]

Figure 8 shows the runtime distribution (PDF) of five nodes for H.264 and MPEG4-d, respectively. It shows the ratio of maximum to minimum execution cycle for each node (X-axis). It also shows the relative portion (numbers in rectangles) of execution cycles of each node to the total execution cycle. For instance, the fourth program region of H.264 has the ratio of 6.09 (maximum execution cycle is 6.09 times bigger than minimum execution cycle) and consumes 9.3% of total execution cycle. As shown in the figure, H.264 gives more runtime distribution than MPEG4-d. This fact is reflected in the experimental results in Figure 9.

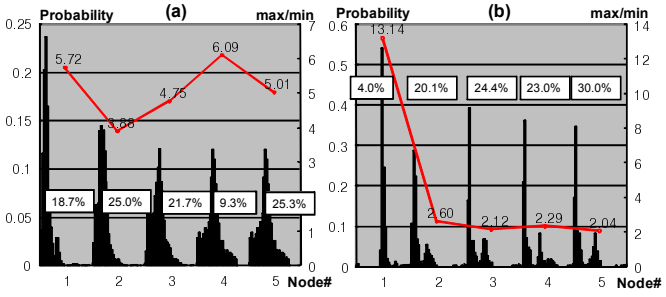


Figure 8 Runtime distributions of H.264 (a) and MPEG4-d (b)

Figure 9 shows the comparison of energy consumption for the four software applications. We apply three methods of workload estimation: worst-case remaining execution cycle-based method

[6] Regarding the voltage transition overhead, 50μs, we take a conservative approach that processor does not perform computation during the transition. The runtime overhead of function PS() is negligible in our examples (in reality also) since the spacing between two consecutive PSP's is in the order of millisecond as shown in Table 2.

[7] To find suitable PSP locations is another interesting problem, but is beyond the scope of this paper. Practically, however, programmers can easily identify a few candidates among major loops and functions in their codes.

[8] In the case of MP3 application, the deadline of 20ms is set assuming a multi-task software execution environment where most of processor cycles are consumed by other compute-intensive user programs, e.g. web browsing, game, multimedia searching, etc.

(WT), e.g. [13], average remaining execution cycle-based method (AT), and the proposed one (Ours). The results are normalized to the WT method. In order to analyze the effectiveness of those methods, four different temperatures are assumed as shown in the figure. The figure also shows the energy reduction (%) of our method compared with the best of WT and AT, i.e. min(WT, AT). The presented method gives up to 44% reduction in energy consumption.

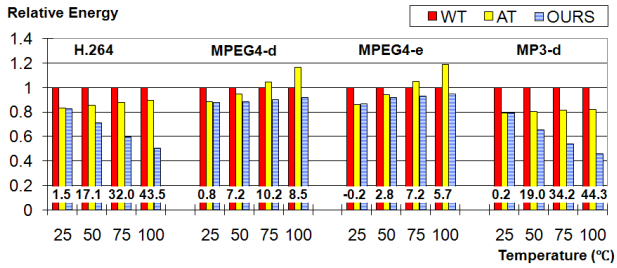


Figure 9 Energy consumption comparison

Figure 10 explains how the presented method gives better energy efficiency than the other two. The figure shows the frequency change of the three methods during the execution of H.264 decode application for three frames. As shown in the figure, WT starts at a high frequency since it assumes the worst-case execution for the remaining execution. However, as the program run advances, performance level drops rapidly and the program finishes earlier than the other two cases. AT shows the opposite behavior. In the beginning, assuming the average execution cycle as the remaining workload, it starts with a very low frequency level. However, due to the too optimistic estimation in the beginning, the performance level needs to be increased at the end of execution to meet the given timing constraint, in this case, 30ms for one frame decoding. We call each of the above frequency settings *early* and *late high frequency setting*, respectively. As shown in Figure 2, at high temperatures, high frequency levels suffer from the penalty of large leakage power. Thus, both WT and AT suffer from this penalty. The presented method takes a balanced approach. As shown in Figure 10, it starts a performance level between those of WT and AT and keeps the balanced position to the end of execution thereby avoiding the penalty of high frequency.

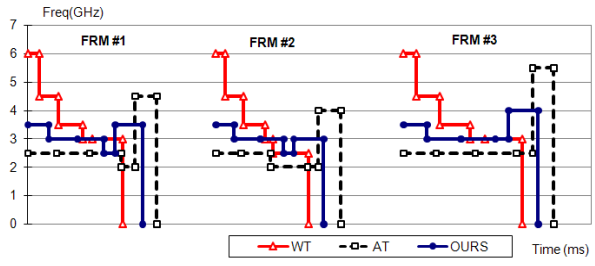


Figure 10 Comparison of frequency settings

The results of Figure 9 are explained by both (1) the difference of early and late high frequency settings between WT and AT and (2) the runtime distribution shown in Figure 8. In H.264 and MP3, WT suffers from large energy consumption at high temperatures. It is because (1) their runtime distribution (H.264's is shown in Figure 8) has high max/min ratio (thus, worst-case estimation can be too pessimistic) and (2) the penalty of its early high frequency becomes dominant at high temperatures.

In contrast, in the cases of MPEG4-d and MPEG4-e, AT gives inferior results to the others as temperature increases. The max/min ratio is small in these cases (MPEG4-d's max/min ratio is shown in Figure 8). Thus, the penalty of early high frequency in WT diminishes since the workload estimation based on worst-case execution cycle gives more accurate estimation than in the case of high max/min ratio. However, AT still suffers from the penalty of late high frequency settings thereby giving inferior results.

7. CONCLUSION

In this paper, we presented a DVS problem based on V_{dd}/V_{bs} scaling and software runtime distribution. We explained the problem mathematically and presented a numerical solution to solve this problem. The experimental results show that the presented method gives significant energy reduction, up to 44%, especially when temperature is high and leakage power dominates. Currently, we are working on applying the presented method to multi-processor DVS and on developing adaptive methods that exploit the dynamically varying software runtime distribution.

8. REFERENCES

[1] D. Kwon and T. Kim, "Optimal Voltage Allocation Techniques for Variable Voltage Processors", DAC, 2003.

[2] K. Choi, W. Lee, R. Soma, and M. Pedram, "Dynamic Voltage and Frequency Scaling under a Precise Energy Model Considering Variable and Fixed Components of the System Power Dissipation", ICCAD, 2004.

[3] S. Lee and T. Sakurai, "Run-time Voltage Hopping for Low-Power Real-Time Systems", DAC, 2000.

[4] A. Azevedo, *et. al.*, "Profile-Based Dynamic Voltage Scheduling Using Program Checkpoints", DATE, 2002.

[5] D. Shin and J. Kim, "Optimizing Intra-Task Voltage Scheduling using Data Flow Analysis", ASPDAC, 2005.

[6] J. Seo, T. Kim, and K. Chung, "Profile-Based Optimal Intra-Task Voltage Scheduling for Hard Real-Time Applications", DAC, 2004.

[7] S. Hong, *et. al.* "Runtime Distribution-Aware Dynamic Voltage Scaling", ICCAD, 2006.

[8] R. Jejurikar and R. Gupta, "Dynamic Slack Reclamation with Procrastination Scheduling in Real-time Embedded Systems", DATE, 2005.

[9] R. Jejurikar, C. Pereria, R. Gupta, "Leakage Aware Dynamic Voltage Scaling of Real-Time Embedded Systems", DAC, 2004.

[10] S. Martin, K. Flautner, T. Mudge, D. Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads", ICCAD, 2002.

[11] L. Yan, J. Luo, N. Jha, "Joint Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-Time Embedded Systems", TCAD, 2005.

[12] W. Liao, F.Li, L.He, "Microarchitecture Level Power and Thermal Simulation Considering Temperature Dependent Leakage Model", ISPLED, 2003.

[13] P. Huang, S. Ghiasi, "Leakage-aware Intraprogram Voltage Scaling for Embedded Processors", DAC, 2006.

[14] P. Huang, S. Ghiasi, "Efficient and Scalable Compiler-Directed Energy Optimization for Realtime Application", DATE, 2007.