Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints

Alexandru Andrei [*], Marcus T. Schmitz [†], Petru Eles [*], Zebo Peng [*], Bashir M. Al Hashimi [†]

[*]Dept. of Computer and Information Science
Linköping University
Linköping ,S-58183, Sweden
{alean,petel,zebpe}@ida.liu.se

[†]Dept. of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, United Kingdom
{ms4,bmah}@ecs.soton.ac.uk

Abstract

Supply voltage scaling and adaptive body-biasing are important techniques that help to reduce the energy dissipation of embedded systems. This is achieved by dynamically adjusting the voltage and performance settings according to the application needs. In order to take full advantage of slack that arises from variations in the execution time, it is important to recalculate the voltage (performance) settings during run-time, i.e., online. However, voltage scaling (VS) is computationally expensive, and thus significantly hampers the possible energy savings. To overcome the online complexity, we propose a quasi-static voltage scaling scheme, with a constant online time complexity $O(1)$. This allows to increase the exploitable slack as well as to avoid the energy dissipated due to online recalculation of the voltage settings. We conduct several experiments that demonstrate the advantages of the proposed technique over the previously published voltage scaling approaches.

1 Introduction and Related Work

Two system-level approaches that allow an energy/performance trade-off during application run-time are dynamic voltage scaling (DVS) [5, 9, 12] and adaptive body-biasing (ABB) [10, 12]. While DVS aims to reduce the dynamic power consumption by scaling down circuit supply voltage V_{dd}, ABB is effective in reducing the leakage power by scaling down frequency and increasing the threshold voltage V_{th} through body-biasing. Voltage scaling (VS) approaches for time constrained multi-task systems can be broadly classified into *offline* (e.g. [3, 5, 9, 11]) and *online* (e.g. [5, 7, 8, 20]) techniques, depending on when the actual voltage settings are calculated. Offline techniques calculate all voltage settings at compile time (before the actual execution), i.e., the voltage settings for each task in the system are not changed at run-time. On the other hand, online techniques recompute the voltage settings during run-time. Both approaches have their advantages and disadvantages. Offline VS approaches avoid the computational overhead in terms of time and energy associated with the calculation of the voltage settings. However, to guarantee the fulfilment of deadline constraints, worst-case execution times (WCET) have to be considered during the voltage calculation. In reality, nevertheless, the actual execution time of the tasks, for most of their activations, is shorter than their WCET, with variations of up to 10 times [15]. Thus, an offline optimization based on the worst case is too pessimistic and hampers the achievable energy savings. In order to take advantage of the dynamic slack that arises from variations in the execution times, it is useful to dynamically recalculate the voltage settings during application run-time, i.e., *online*.

Dynamic approaches, however, suffer from the significant overhead in terms of execution time and power consumption caused by the online voltage calculation. As we will show, this overhead is intolerablly large even if low complexity online heuristics are used instead of higher complexity optimal algorithms. Unfortunatelly, researchers have neglected this overhead when reporting high quality results obtained with dynamic approaches [5, 7, 8, 20].

In [13] an approach is outlined in which the online scheduler is executed at each activation of the application. The decision taken by the scheduler is based on a set of precalculated supply voltage settings. The approach assumes that at each activation it is known in advance which subgraphs of the whole application graph will be executed. For each such subgraph worst case execution times are assumed and, thus, no dynamic slack can be exploited.

In this paper we propose a quasi-static voltage scaling technique for energy minimization of multi-task real-time systems. This technique is able to exploit the dynamic slack and, at the same time, keeps the online overhead (required to readjust the voltage settings at run-time) extremly low. The obtained performance is superior to any of the previously proposed dynamic approaches. Henceforth, we will refer to the proposed voltage scaling technique as quasi-static voltage scaling (QSVS).

The work presented in this paper makes the following contributions: **(a)** A quasi-static voltage scaling scheme for multi-task applications is proposed. In the offline phase, it computes and stores possible voltage settings. The online phase is responsible to adapt the applied voltage settings at run-time, based on the stored information and the actual task execution times. We investigate the "optimality" of QSVS compared to an ideal, but practically infeasible, online voltage scaling approach. The time complexity of the QSVS technique is polynomial in the offline phase and constant in the online part. **(b)** The introduced QSVS scheme addresses the *combined* reduction of dynamic and leakage power consumption using supply and body-bias voltage scaling. **(c)** We perform, for the first time, an evaluation of the impact of the overhead of different dynamic voltage scaling approaches on realistic applications. These evaluations are performed using cycle accurate simulation tools as well as measurements on a real processor.

The paper is organized as follows: Preliminaries and motivations are given in Section 2, as well as the key ideas behind the presented work. An exact problem formulation for quasi-static voltage scaling is given in Section 3. Our algorithms to solve this problem are described in Section 4. Extensive experimental results, including a real-life example, are presented in Section 5. Finally, conclusions are drawn in Section 6.

2 Preliminaries

2.1 Application and Architecture Model

In this work, we consider applications that are modeled as task graphs, i.e., several tasks with possible data dependencies among them, as in Fig. 1(a). Each task is characterized by several parameters (see also section 3), such as a deadline, the effectively switched capacitance, and the number of clock cycles required in the best-case (BNC), expected-case (ENC), and worst-case (WNC). Tasks are running without being preempted until their completion. For such systems is has been proved in [19] that the EDF (earliest deadline first) scheduling is optimal from the energy point of view. Thus, we assume that the order of task execution is fixed offline, according to an EDF policy, as shown in Fig. 1(b). The tasks are executed on an embedded architecture that consists of a voltage-scalable processor (scalable in terms of *supply* and *body-bias*

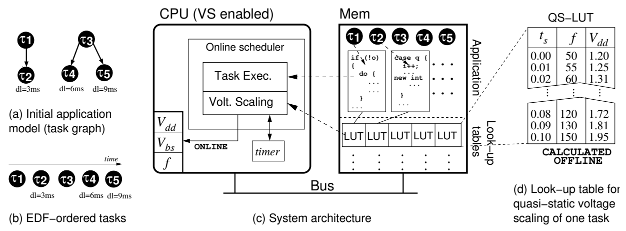


Figure 1. System architecture

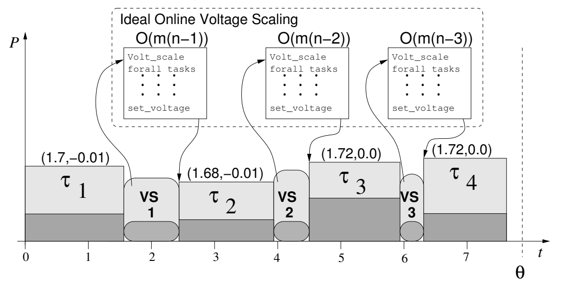


Figure 2. Ideal online voltage scaling approach

voltage). The processor is connected to a memory that stores the application and a set of look-up tables (LUT), one for each task, required for QSVS. This architectural setup is shown in Fig. 1(c). During execution the scheduler has to adjust the processor's performance to the appropriate level via voltage scaling, i.e., the scheduler writes the settings for the operational frequency f, the supply voltage V_{dd}, and the body-bias voltage V_{bs} into special processor registers before the task execution starts. An appropriate performance level allows the tasks to meet their deadlines while maximizing the energy savings. In order to exploit slack that arises from variations in the execution time of tasks, it is unavoidable to dynamically re-calculate the performance levels. Nevertheless, as we shall see later in Section 2.3.1, voltage scaling (the means by which performance levels are calculated) is a computational expensive task, i.e., it requires precious CPU time, which, if avoided would allow to lower the CPU performance and, consequently, the energy.

The approach presented in this paper aims to reduce this online overhead by performing the necessary VS computations offline (at compile time) and storing a limited amount of information as look-up tables (LUTs) within memory. This information is then used during application run-time (i.e., online) to calculate the voltage and performance settings extremely fast (constant time $O(1)$), see Fig. 1(d).

2.2 Power and Delay Models

Digital CMOS circuitry has two major sources of power dissipation: (a) dynamic power P_{dyn}, which is dissipated whenever active computations are carried out (switching of logic states), and (b) leakage power P_{leak} which is consumed whenever the circuit is powered, even if no computations are performed. The dynamic power is expressed by [4, 12],

$$P_{dyn} = C_{eff} \cdot f \cdot V_{dd}^2 \tag{1}$$

where C_{eff}, f, and V_{dd} denote the effective charged capacitance, operational frequency, and circuit supply voltage, respectively. Although, until recently, the dynamic power dissipation had been dominating, the trend to reduce the overall circuit supply voltage and, consequently, threshold voltage is raising concerns about the leakage currents—for near future technology ($< 70nm$) it is expected that leakage will account for more than 50% of the total power. The leakage power is given by [12],

$$P_{leak} = L_g \cdot V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju} \tag{2}$$

where V_{bs} is the body-bias voltage and I_{Ju} represents the body junction leakage current. The fitting parameters K_3, K_4 and K_5 denote circuit technology dependent constants and L_g reflects the number of gates. For clarity reasons we maintain the same indices as used in [12], where also actual values for these constants are provided. Nevertheless, scaling the supply and the body-bias voltage, in order to reduces the power consumption, has a side effect on the circuit delay d, which is inverse proportional to the operational frequency f [4, 12],

$$f = \frac{1}{d} = \frac{((1+K_1) \cdot V_{dd} + K_2 \cdot V_{bs} - V_{th1})^{\alpha}}{K_6 \cdot L_d \cdot V_{dd}} \tag{3}$$

where α denotes the velocity saturation imposed by the given technology (common value: $1.4 \leq \alpha \leq 2$), L_d is the logic depth, and K_1, K_2, K_6, and V_{th1} reflect circuit dependent constants [12]. Equations (1), (2), and (3) provide the energy/performance trade-off of digital circuits.

2.3 Motivation

This section motivates the presented work and outlines the basic idea of the proposed quasi-static voltage scaling technique.

2.3.1 Complexity Evaluation of Voltage Scaling

As we have mentioned in the introduction, to fully take advantage of variations in the execution time of tasks, with the aim to reduce the energy dissipation, it is unavoidable to recompute the voltage settings online according to the actual task execution times. This is demonstrated in Fig. 2, where we consider an application consisting of $n = 4$ tasks. Only after task τ_1 has terminated, we know its actual finishing time and, accordingly, the amount of dynamic slack that can be distributed to the remaining tasks (τ_2,τ_3,τ_4). Ideally, in order to optimally distribute the slack among these tasks (τ_2,τ_3, and τ_4), it is necessary to run a voltage scaling algorithm (in Fig. 2 indicated as VS 1) before starting the execution of task τ_2. A straightforward implementation of an ideal online voltage scaling algorithm is to perform a "complete" recalculation of the voltage settings each time a task finishes, using for example the VS approaches described in [11, 17]. However, such an implementation would be only feasible if the computational overhead associated with the voltage scaling algorithm is very low, which is not the case in practice. The computational complexity of such optimal voltage scaling algorithms for monoprocessor systems is $O(m \cdot n)$ [11, 17] (with m specifying the accuracy-a usual value of 100 and n being the number of tasks). That is, a substantial amount of CPU cycles are spent calculating the voltage/frequency settings each time a task finishes—during these cycles the CPU uses precious energy and reduces the amount of exploitable slack.

To get insight into the computational requirements of voltage scaling algorithms and how this overhead compares to the amount of computations performed by actual applications, we have simulated and profiled several applications and voltage scaling techniques, using two cycle accurate simulators: StrongARM (SA-1100) [14] and PowerPC(MPC750)[18]. We have also performed measurements on actual implementations using an AMD platform (AMD Athlon 2400XP). Tab. 1 shows these results for two applications that can be commonly found in hand-held devices: a GSM voice codec and an MPEG video encoder. Results are shown for AMD, SA-1100 and MPC750 and are

Bench-mark type	AMD Athlon			SA1100			MPC750		
	BNC (k)	WNC (k)	Var. (%)	BNC (k)	WNC (k)	Var. (%)	BNC (k)	WNC (k)	Var. (%)
GSM	140	155	10	367	394	7	159	181	13
MPEG	731	1,700	43	4,458	8,043	45	3,869	6,439	40

Table 1. Simulation results of different applications

given in terms of best-case (BNC) and worst-case number (WNC) of thousands of clock cycles needed for the execution of one period of the considered applications (20 ms for the GSM codec and 40 ms for the MPEG encoder). [1] For instance, on the SA-1100 processor one iteration of the MPEG encoder requires in the best-case 4.458 kcycles and in the worst-case 8.043 kcycles, that is a variation of 45%. Similarly,

[1]Note that the numbers for BNC and WNC are lower and upper bounds observed during the profiling. They have not been analytically derived.

Voltage scaling algorithm	AMD	SA-1100	MPC750
	NC (k)	NC (k)	NC (k)
OptimalVS(Vdd+Vbs, 20 tasks) [11]	8,410	1,232,552	136,950
OptimalVS(Vdd, 20 tasks) [11]	210	32,320	3,513
MTS Heuristic(Vdd, 20 tasks) [6]	8	84	12
MTS Heuristic(Vdd+Vbs, 20 tasks)	40	623	73
Greedy Heuristic(Vdd) [7]	0.9	10	1.0
Greedy Heuristic(Vdd+Vbs)	4.9	34	3.8
Quasi-Static(Vdd+Vbs) (proposed)	0.9	1.0	1.0

Table 2. Simulation results: Voltage scaling algorithms

Tab. 2 presents the simulation outcomes for different voltage scaling algorithms. As an example, performing the optimal online voltage scaling using the algorithm from [11] *once* for 20 remaining tasks (just like VS 1 is once performed for the three remaining tasks τ_2, τ_3, and τ_4 in Fig. 2) requires 8,410 kcycles on the AMD processor, 136,950 kcycles on the MPC750 processor, while the SA-1100 requires even 1,232,552 kcycles. Using the same algorithm for V_{dd}-only scaling (no V_{bs} scaling), needs 210 kcycles on the AMD processor, 32,320 kcycles on the SA-1100 and 3,513 kcycles on the MPC750. The difference in complexity between supply voltage scaling and combined supply and body bias scaling comes from the fact that in the case of V_{dd}-only, for a given frequency there exists one corresponding supply voltage, as opposed to a potentially infinite number of (V_{dd}, V_{bs}) pairs in the other case. Given a certain frequency, an optimization is needed to compute the (V_{dd}, V_{bs}) pair that minimizes the energy. Comparing the results in Tables 1 and 2 indicates that voltage scaling often surpasses the complexity of the applications itself. For instance, performing a "simple" V_{dd}-only scaling requires more CPU time (on AMD 210k cycles) than decoding a single voice frame using the GSM codec (on AMD 155k cycles). Clearly, such overheads seriously affect the possible energy savings, or even outdo the energy consumed by the application.

Several suboptimal heuristics with lower complexities have been proposed for online computation of the supply voltage. Gruian [6] has proposed a linear time heuristic, while the approaches given in [7, 20] use a greedy heuristic of constant time complexity. We report their performance in terms on the required number of cycles in Tab. 2, including also their additional adaptation for combined supply and body bias scaling. While these heuristics have a smaller online overhead than the optimal algorithms, their cost is still high, except for the greedy algorithm for supply voltage scaling [7, 20]. However, even the cost of the greedy increases up to 5.4 times when it is used for supply and body bias scaling. The overhead of our proposed algorithm is given in the last line of Tab. 2.

2.3.2 Basic Idea: Quasi-Static Voltage Scaling

To overcome the VS overhead problem, we propose a quasi-static voltage scaling technique. This approach is divided into two phases. In the first phase, which is performed before the actual execution (i.e., offline), voltage settings for all tasks are pre-computed based on possible task start times. The resulting voltage/frequency settings are stored in look-up tables (LUTs) that are specific to each task. It is important to note that this phase performs the time intensive optimization of the voltage settings.

The second phase is performed online and it is outlined in Fig. 3. Each time new voltage settings for a task need to be calculated, the online scheme looks up the voltage/frequency settings from the LUT based on the actual task start time. If there is no exact entry in the LUT that corresponds to the actual start time, then the voltage settings are estimated using a linear interpolation between the two entries that surround the actual start time. For instance, task τ_3 has an actual start time of 3.58ms. As indicated in Fig. 3, this start time is surrounded by the LUT entries 3.55ms and 3.60ms. In accordance, the frequency and voltage setting for task τ_3 are interpolated based on these entries. The main advantage of the online quasi-static VS algorithm is its constant time complexity $\mathcal{O}(1)$. As shown in the last line of Tab. 2, the LUT look-up

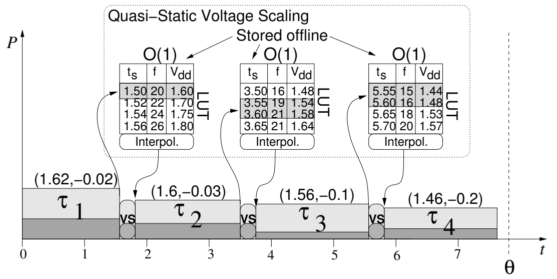


Figure 3. Quasi-static voltage scaling based on pre-stored look-up tables

and voltage interpolation requires only 900 CPU cycles each time new voltage settings have to be calculated. Please note the complexity of the online quasi-static VS is independent of the number of remaining tasks.

3 Problem Formulation

Consider a set of NT tasks, $\mathcal{T} = \{\tau_i\}$ such that the execution order is fixed according to an EDF policy. The processor can vary its supply voltage V_{dd} and body-bias voltage V_{bs} within certain continuous ranges. The dynamic and leakage power dissipation as well as the operational frequency (cycle time) depend on the selected voltage pair (mode). Tasks are executed cycle by cycle and each cycle can be potentially executed at different voltage settings, i.e., a different energy/performance trade-off. Each task τ_i is characterized by a six-tuple,

$$\tau_i = < BNC_i, ENC_i, WNC_i, Ceff_i, D_i >$$

where BNC_i, ENC_i, and WNC_i denote the best-case, the expected-case, and the worst-case number of clock cycles, respectively, that task τ_i requires for its execution. BNC (WNC) is defined as the lowest (highest) number of cycles task τ_i needs for its execution, while ENC is the arithmetic mean value of the probability density function $p(WNC)$ of the task execution cycles WNC, i.e., $ENC = \sum_{j=1}^{WNC} j \cdot p_j(j)$. Further, $Ceff_i$ and D_i represent the effectively charged capacitance and the deadline. The aim is to reduce the energy consumption by exploiting *dynamic slack* as well as *static slack*. Dynamic slack results from tasks that require less execution cycles than in their worst case. Static slack is the result of idleness due to system over-performance, observable even when tasks execute with the worst-case number of cycles.

Our goal is to store a look-up table LUT$_i$ for each task τ_i, such that the energy consumption during runtime is minimized. The size of the memory available for storing the look-up tables (and, implicitly the total number NL of table entries) is given as a constraint.

4 Quasi-Static Algorithm

Quasi-static voltage scaling aims to reduce the online overhead required to compute voltage settings by splitting the voltage scaling process into two phases. That is, the voltage settings are prepared offline, and the stored voltage settings are used online to adjust the voltage/frequency in accordance to the actual task execution times.

4.1 Offline Algorithm

The pseudo-code corresponding to the calculations performed offline is given in Fig. 4. The algorithm requires the following input information. The scheduled task set $\mathcal{T}$, defined in section 3. For the tasks $\tau_i \in \mathcal{T}$, the expected (ENC$_i$), the worst-case (WNC$_i$) and the best-case (BNC$_i$) number of cycles, the effectively switched capacitance (Ceff$_i$) and the deadline D$_i$. Furthermore, total number of look-up table entries NL is given. The algorithm returns the quasi-static scaling table `LUT`$_i$, for each task $\tau_i \in \mathcal{T}$. This table includes n_i ($\sum_{i=1}^{n} n_i = NL$) possible start times $t_{s_{i,j}}, j = 1..n_i$ for each task τ_i, and the corresponding optimal settings for the supply voltage `Vdd` and the operational frequency `f`.

```
Algorithm:  QUASI_STATIC_VS_OFF-LINE
Input:  - execution order of tasks $\tau \in \mathcal{T}$
        - for all tasks $\tau_i \in \mathcal{T}$:
          $BNC_i, ENC_i, WNC_i, Ceff_i, D_i$
        - $NL$
Output: - Lookup tables $LUT_i$
01:  for $i=1$ to $NT$ {
02:    EST$_i$ ← calc_earliest_starttime
03:  }//end for
04:  for $i=NT$ downto 1 {
05:    LST$_i$ ← calc_latest_starttime
06:  }//end for
07:  $\mathcal{T}_r \leftarrow \mathcal{T}$
08:  for all $\tau_i \in \mathcal{T}_r$ { //ordered i=1..NT
09:    $I_i$ ← LIST$_i$ − EIST$_i$
10:    j ← 0
11:    $n_i$ ← comp_interpolation_points($\tau_i$,LST$_i$,EST$_i$)
12:    for ($t_s \leftarrow$EST$_i$; $t_s \leq$LST$_i$; $t_s \leftarrow t_s + I_i$/n) {
13:      $t_{s_i} \leftarrow t_s$
14:      (Vdd$_i$,Vbs$_i$,f$_i$) ← volt_scaling($\mathcal{T}_r$,$t_{s_i}$) //ENC based
15:      LUT$_i$[j] ← store_QS_lookup($t_{s_i}$,Vdd$_i$,f)
16:      j ← j + 1
17:    }//end for
18:    $\mathcal{T}_r \leftarrow \mathcal{T}_r - \tau_i$
19:  }//end for all
20:  for all $\tau_i \in \mathcal{T}$ return LUT$_i$
```

Figure 4. Pseudocode: Offline Algorithm

Upon initialization, the algorithm computes the earliest and latest start times for each task (lines 01–06). The earliest start time $\mathtt{EST}_i$ is based on the situation in which all tasks would execute with their best-case number of clock cycles at the highest voltage settings, i.e., the shortest possible execution (lines 01–03). The latest start time $\mathtt{LST}_i$ is calculated as the latest start time of task τ_i that allows to satisfy the deadlines for all the tasks τ_j, $j \geq i$, executed with the worst-case number of clock cycles at the highest voltages (lines 04–06). The algorithm proceeds by initializing the set of remaining tasks $\mathcal{T}_r$ with the set of all tasks $\mathcal{T}$ (line 07). In the following (lines 08–18), the voltage and frequency settings for the start time intervals of each task are calculated. More detailed, in line 09 and 10 the size of the interval $[\mathtt{EST}_i, \mathtt{LST}_i]$ of possible start times is computed and the interval counter j is initialized. The number of entry points $\mathtt{n}_i$ that are stored for each task (i.e., the number of possible start times considered) is calculated in line 11. This will be further discussed in Section 4.3. For all $\mathtt{n}_i$ possible start times t_s in the start time interval of task τ_i (line 12), the task start time t_{s_i} is set to the possible start time (line 13) and the corresponding optimal voltage and frequency settings of τ_i are computed (line 14). For this computation, we use the algorithm presented in [3], modified to incorporate the optimization for the expected case. We will explain this algorithm in Section 4.1.1. At this point, the voltages and frequency of task τ_i for a starting at time t_{s_i} are stored in the quasi-static scaling table $\mathtt{LUT}_i$ (line 15). The for-loop (line 11–23) is repeated for all $\mathtt{n}_i$ possible start times of task τ_i. Please note that we do not store the body-bias voltages $\mathtt{Vbs}_i$, which will be recomputed online. The reason behind this decision will be outlined in the quasi-static online algorithm (Section 4.2). The algorithm returns the quasi-static scaling table $\mathtt{LUT}_i$ for all tasks $\tau_i \in \mathcal{T}$.

4.1.1 Voltage Scaling Algorithm

We will briefly present the voltage scaling algorithm used in line 14 (Fig.4). The problem can be formulated as a convex nonlinear optimization as follows: Minimize

$$\sum_{k=i}^{|\mathcal{T}_r|}\Big(\underbrace{ENC_k \cdot C_{eff_k} \cdot V_{dd_k}^2}_{E_{dyn_k}} + \underbrace{L_g(K_3 \cdot V_{dd_k} \cdot e^{K_4 \cdot V_{dd_k}} \cdot e^{K_5 \cdot V_{bs_k}} + I_{Ju} \cdot |V_{bs_k}|) \cdot t_k}_{E_{leak_k}}\Big) \tag{4}$$

subject to

$$t_k = ENC_k \cdot \frac{(K_6 \cdot L_d \cdot V_{dd_k})}{((1+K_1) \cdot V_{dd_k} + K_2 \cdot V_{bs_k} - V_{th_1})^{\alpha}} \tag{5}$$

```
Algorithm:  QUASI_STATIC_VS_ONLINE
Input:  - start time $t_{sn}$ of next task $\tau_n$
        - Quasi-Static Scaling Table LUT$_n$
        - number of start time interval steps n
Output: - frequency and voltage settings for task $\tau_n$
01:  (x,y) ← calc_st_interval(LUT$_n$,$t_{sn}$)
02:  f$_n$ ← inter_freq(LUT$_n$,x,y,$t_{sn}$)
03:  Vdd$_n$ ← inter_Vdd(LUT$_n$,x,y,$t_{sn}$)
04:  Vbs$_n$ ← calc_Vbs(f$_n$,Vdd$_n$)
05:  return (f$_n$,Vdd$_n$,Vbs$_n$)
```

Figure 5. Pseudocode: Online Algorithm

$$s_k + t_k \leq s_{k+1} \tag{6}$$

$$s_k + t_k \leq D_k \quad \forall \tau_k \text{ that have a deadline} \tag{7}$$

$$s_k + t_k \leq LST_{k+1} \tag{8}$$

$$s_k \geq 0 \tag{9}$$

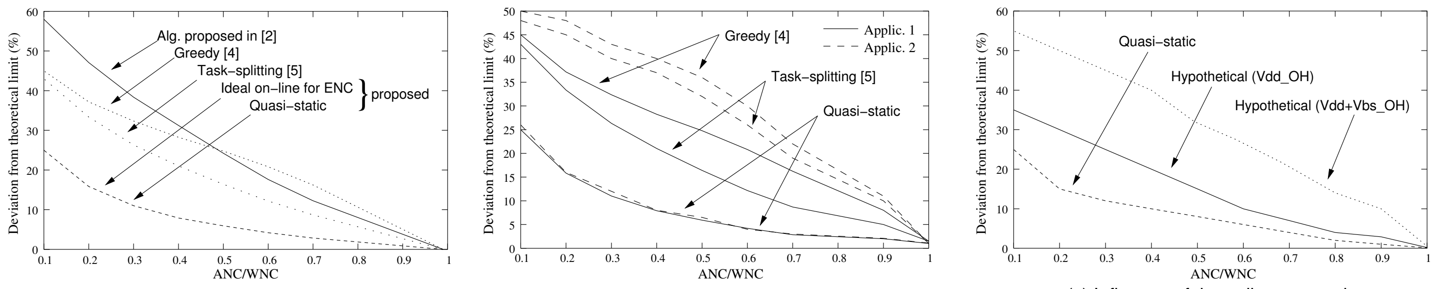
$$V_{dd_{min}} \leq V_{dd_k} \leq V_{dd_{max}} \text{ and } V_{bs_{min}} \leq V_{dd_k} \leq V_{bs_{max}} \tag{10}$$

The variables that need to be optimized in this formulation are the task execution times t_k, the task start times s_k as well as the voltages V_{dd_k} and V_{bs_k}. The whole formulation can be explained as follows. The total energy consumption, which is the combination of dynamic and leakage energy, has to be minimized. As we aim the energy optimization in the most likely case, the expected number of clock cycles ENC_k is used. The minimization has to comply to the following relations and constraints. The task execution time has to be equivalent to the number of clock cycles of the task multiplied by the circuit delay for a particular V_{dd_k} and V_{bs_k} setting, as expressed by Eq. (5). Eq. 6 expresses the task execution order. Deadlines are inforced in Eq. 7. Up to this point, we have used the expected number of clock cycles ENC_k as this is the situation for which we want to optimize the energy consumption. However, in order to insure that tasks will not miss their deadlines in the worst case, we need Eq. 8. Please remember from the computation of the latest start time (LST), that if task τ_k finishes the execution before the latest start time of task τ_{k+1}, then the rest of the tasks are guaranteed to meet their deadlines even in the worst case.

4.2 Online Algorithm

Fig. 5 gives the pseudocode of the online algorithm. This algorithm is called each time after a task finishes its execution, in order to calculate the voltage settings for the next task τ_n. The input consists of the task start time t_{s_n}, the quasi-static scaling table $\mathtt{LUT}_n$, and the number of interval steps $\mathtt{n}_n$. As output, the algorithm returns the frequency $\mathtt{f}_n$ and voltage settings $\mathtt{Vdd}_n$ and $\mathtt{Vbs}_n$ for the next task, τ_n. In the first step, the algorithm calculates the two entries x and y from the quasi-static scaling table $\mathtt{LUT}_n$ that contain the start times which surround the actual time t_{s_n} (line 01). According to the identified entries, the frequency setting $\mathtt{f}_n$ for the execution of task τ_n is linearly interpolated using the two frequency settings from the quasi-static scaling table $\mathtt{LUT}_n\mathtt{[x]}$ and $\mathtt{LUT}_n\mathtt{[y]}$ (line 02). Similarly, in step 03 the supply voltage $\mathtt{Vdd}_n$ is linearly interpolated from the two surround voltage entries in $\mathtt{LUT}_n$. As mentioned in Section 4.1, we do not directly interpolate the setting for the body-bias voltage $\mathtt{Vbs}_n$, due to the nonlinear relation between frequency, supply voltage, and body-bias voltage. That is, interpolating V_{dd} and V_{bs} at the same time, can result in operational frequency that do not match the actually needed frequency—resulting in possible deadline violations. Therefore, we calculate the body-bias voltage directly for the interpolated frequency and supply voltage values, using Eq. 3 (line 04). The algorithm terminates and returns the settings for the frequency, supply and body-bias voltage (line 05). It is worthwhile to mention that all four steps that are necessary to perform the online calculation are computed in constant time, i.e., the time complexity of the quasi-static online algorithm is $O(1)$.

At this point it is also interesting to note that the linear approximation performed by the online algorithm is safe (i.e. frequencies are calculated such that they guarantee that tasks will not violate their dead-



(a) Scaling for the expected−case execution time (assuming zero overhead)

(b) Influence of the online overhead on different online VS approaches

(c) Influence of the online overead on an ideal linear time heuristic

Figure 6. Experimental results: online voltage scaling

lines). It can be shown that running the voltage scaling algorithm for all possible start times t_s of a task results in a frequency function $f(t_s)$ that is convex [2]. Thus any linear approximation of the frequency will result in frequency values higher or equal to the optimal one, and consequently no deadline will be violated. The quality of the the linear approximation depends directly on the number of intermediate start times used. The next section will discuss this important aspect.

4.3 Calculation of the Look-Up Table Sizes

In this section we address the problem of how many entries to assign to each LUT under a given memory constraint, such that the resulting entries yield high energy savings. A simple approach to distribute the memory among the LUTs is to allocate the same number of entries for each LUT. However, due to the fact that different tasks have different start time interval sizes and nominal energy consumptions, the memory should be distributed using a more effective scheme (i.e. reserving more memory for critical tasks). In the following we will introduce a heuristic approach to solve the LUT size problem. Clearly, the two main parameters that determine the criticality (in the sense that it should be allocated more entries in the LUT) of a task are the size of the interval of possible task τ_i start times ($LST_i - EST_i$) and the nominal expected energy consumption of a task (E_i). The expected energy consumption of a task E_i is the energy consumed by that task when executing the expected number of clock cycles (ENC_i) at the nominal voltages. Consequently, in order to allocate the n_i look-up table entries for each tasks, we use the following formula:

$$n_i = NL \cdot \frac{E_i \cdot (LST_i - EST_i)}{\sum_{i=1}^{NT} E_i \cdot (LST_i - EST_i)} \tag{11}$$

5 Experimental Results

We have conducted several experiments using numerous generated benchmarks as well as a real-life application, in order to demonstrate the applicability of the proposed approach. The processor parameters have been adopted from [12].

The first set of experiments was conducted in order to investigate the quality of the results provided by different online VS techniques in the case when their actual run-time overhead is ignored. In Fig. 6(a) we show the results obtained with the following five different VS approaches:

1) the ideal online VS approach (the scheduler that calculates the optimal VS with no overhead).

2) the quasi-static VS technique proposed in this paper (Section 4).

3) the greedy heuristic proposed in [7].

4) the task splitting heuristic proposed in [20].

5) the algorithm proposed in [5].

Originally, approaches given in [5, 7, 20] perform DVS only. However, for comparison fairness, we have extended these algorithms towards combined supply and body-bias scaling. The results of all five techniques are given as the percentage deviation from the results produced by a hypothetical voltage scaling algorithm that would know in advance the exact number of clock cycles executed by each task. Of course such an approach is practically impossible. Nevertheless, we use this theoretical lower limit as baseline for the comparison. During the experiments, we varied the ratio of actual number of clock cycles (ANC) and worst case number of clock cycles (WNC) from 0.1 to 1 with a step width of 0.1. For each step, 1000 randomly generated task graphs were evaluated, resulting in a total of 10000 evaluations for each plot. As mentioned earlier, for this first experiment we ignored the computational overheads of all the investigated approaches, i.e., we assumed that the voltage scaling requires zero time. Furthermore, the actual number of clock cycles (ANC) are set based on a normal distribution using the expected number of cycles (ENC) as the mean value. Observing Fig. 6(a) leads to the following interesting conclusions. Firstly, if the actual number of cycles (ANC) corresponds to the worst-case number (WNC), all VS techniques approach the theoretical limit. In other words, if the application has been scaled for the WNC and all task execute with WNC, then all online VS techniques perform equally well. This, however, changes if the ANC differs from the WNC, which is always the case in practice. For instance, in the case that the ratio between ANC and WNC is 0.1, we can observe that ideal online VS is 25% off the theoretical limit. On the other hand, the technique described in [5] is 60% worse than the theoretical limit. The approaches based on the methods proposed in [7, 20] yield results that are 42% and 45% below the theoretical optimum. Another interesting observation is the fact that the ideal online scaling and our proposed quasi-static technique produce results of the same high quality. Of course, the quality of the quasi-static VS depends on the number of entries that are stored in the look-up tables (LUTs). Due to the importance of this influence, we have devoted a supplementary experiment to demonstrate how the number of entries affects the VS quality. In the experiment illustrated in Fig. 6(a) and (b) the total number of entries was set to 4000, which was sufficient to achieve results that differed with less then 0.5% from the ideal online scaling for task graphs with up to 100 nodes. In summary, Fig. 6(a) demonstrates the high quality of the voltage settings produced by the quasi-static approach, which are very close of those produced by the ideal algorithm and substantially better then the values produced by any other proposed approach.

In order to evaluate the global quality of the different VS approaches (taking into consideration the online overheads), we conducted two sets of experiments (Fig. 6(b) and Fig. 6(c)). In Fig. 6(b) we have compared our quasi-static algorithm with the approaches proposed in [7, 20]. The influence of the overheads is tightly linked with the size of the applications. Therefore, we use two sets of applications (Applic. 1 and Applic. 2) of different sizes. Applic. 1 has the size comparable to that of the MPEG encoder and Applic. 2 has a size similar to the GSM codec. As we can observe, the proposed quasi-static voltage scaling achieves considerably higher savings than the other two approaches. Although all three approaches illustrated in Fig. 6(b) have constant online complexity ($O(1)$), the overhead of the quasi-static approach is considerably lower. At the same time, as shown in Fig. 6(a), the quality of settings produced by QSVS is much higher.

In Fig. 6(c) we have compared our quasi-static approach with a hy-

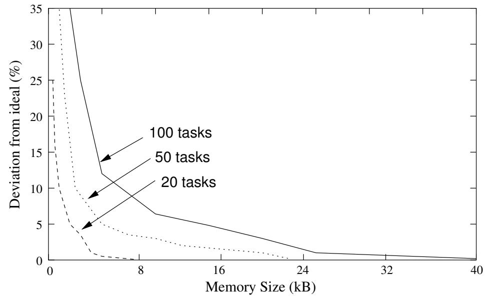


Figure 7. Experimental results: deviation from ideal

pothetical "best possible" dynamic voltage scaling algorithm. Such a hypothetical algorithm would produce the optimal voltage settings with a linear overhead similar to that of the heuristic proposed in [6] (see Tab. 2). Please note that such an algorithm has not been proposed since all known optimal solutions incur a higher complexity then the one in [6]. We evaluated 10000 randomly generated task graphs. In this particular experiment we set the size of the task graphs similar to the MPEG encoder. We considered two cases: the hypothetical online algorithm is executed with the overhead from [6] for V_{dd}-only and with the overhead that would result if the algorithm is rewritten for the combined (V_{dd}, V_{bs}) scaling. Please note that in both of the above cases we consider that the hypothetical algorithm performs V_{dd} as well as V_{bs} scaling. As we can see, the quasi-static algorithm is superior by up to 10% even to the hypothetical algorithm with the lower V_{dd}-only overhead, while in case which is still optimistic but closer to reality of the higher (V_{dd}, V_{bs}) overhead the superiority of the quasi-static approach is up to 30%. Overall these experiments demonstrate that the quasi-static solution is superior to any proposed and foreseeable dynamic voltage scaling approach.

The next set of experiments was conducted in order to demonstrate the influence of the memory size used of the look-up tables on the possible energy savings with the quasi-static voltage scaling. For this experiment we have used three sets of tasks graphs with 20, 50, and 100 tasks, respectively. Fig. 7 shows the percentage deviation of energy savings with respect to an ideal online VS as a function of the memory size. For example, in order to obtain a deviation below 0.5%, a memory of 40kB is needed for systems consisting of 100 tasks. For the same quality, 20 and 8kB are needed for 50 and 20 tasks, respectively. It is interesting to observe that with a small penalty in the energy savings, the required memory decreases almost by half. For instance, for 100 tasks, the quasi-static algorithm achieves 2% deviation relative to the ideal algorithm with a memory of only 24kB. It is important to note, that in all the performed experiments we have taken into consideration the energy overhead due to the memories. This overheads have been calculated based on the energy values raported in [1, 16] in the case of SRAM memories.

In addition to the above given benchmark results, we have conducted experiments on a real-life MPEG encoder. The MPEG encoder consists

Approach	E(μJ)	Reduc. (%)
Nominal	1.63	–
Static VS	1.39	15
Greedy [7]	0.55	67
Task Splitting [20]	0.52	69
Quasi-static	0.36	78

Table 3. Optimization results for the MPEG algorithm

of 25 tasks and is considered to run on a MPC750 processor. Tab. 3 shows the resulting energy consumption obtained with different scaling approaches. The first line gives the energy consumption of the MPEG encoder running at the nominal voltages. Line two shows the result obtained with an optimal static voltage scaling approach. The energy improvement in this case is approximatelly 15%. Lines 3-4 show the improvements produced using the greedy online techniques proposed in [7, 20] which achieve reductions of 67% and 69%, respectively. The last row presents the results obtained by our quasi-static algorithm that improves over the nominal consumption by 78%. The results confirm the high solution quality of the proposed quasi-static scaling technique.

6 Conclusions

In this paper, we have introduced a novel quasi-static voltage scaling technique for time-constraint applications. The method avoids an unnecessarily high overhead by precomputing possible voltage scaling scenarios and storing the outcome in look-up tables. The avoided overheads can be turned into additional energy savings. Furthermore, we have addressed both dynamic and leakage power through supply and body-bias voltage scaling. We have shown that the proposed approach is superior to both static and dynamic approaches proposed so far in literature. Experiments conducted on numerous automatically generated examples and real-life benchmarks demostrate the quality of the proposed technique. We have introduced our approach considering that continous voltages are available. However, the proposed technique can be easily adapted to the situation when only discrete voltage levels are available. In this case the calculation of the discrete voltage levels which replace the calculated continuous ones, as well as the consideration of the transition overheads, can be performed using the techniques proposed in [3, 9].

Acknowledgements

Alexandru Andrei was supported by CUGS swedish graduate school in computer science. Petru Eles and Zebo Peng were supported by SSF through the STRINGENT excellence center. Marcus Schmitz and Bashir Al Hashimi were supported by EPSRC under grant GR/S95770.

References

[1] E. Macii A. Macii and M. Poncino. Improving the Efficiency of Memory Partitioning by Address Clustering. In *Proc. Design, Automation and Test in Europe Conf. (DATE03)*, pages 18–22, March 2003.

[2] A. Andrei, M.T. Schmitz, P. Eles, and Z. Peng. Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints. Technical report, Linkoping University, Department of Computer and Information Science, Sweden, September 2004.

[3] Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, and Bashir Al-Hashimi. Overhead-Conscious Voltage Selection for Dynamic and Leakage Power Reduction of Time-Constraint Systems. In *Proc. Design, Automation and Test in Europe Conf. (DATE04)*, pages 518–523, Feb 2004.

[4] Anantha P. Chandrakasan and Robert W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publisher, 1995.

[5] A. Demers F. Yao and S. Shenker. A Scheduling Model for Reduced CPU Energy. *Proc. IEEE FOCS*, pages 374–382, 1995.

[6] F. Gruian. Energy-Centric Scheduling for Real-Time Systems. In *Phd Thesis*, 2002.

[7] D. Mosse H. Aydin, R. Melhem and P. Mejia-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proc. RTSS'01*, pages 95–105, 2001.

[8] M. Potkonjak I. Hong and M. B. Srivastava. On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processors. In *Proc. ICCAD'98*, pages 653–656, 1998.

[9] Tohru Ishihara and Hiroto Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. Int. Symp. Low Power Electronics and Design (ISLPED'98)*, pages 197–202, 1998.

[10] C. Kim and K. Roy. Dynamic Vth Scaling Scheme for Active Leakage Power Reduction. In *Proc. Design, Automation and Test in Europe Conf. (DATE02)*, pages 163–167, March 2002.

[11] J. Luo L. Yan and N. Jha. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems. In *Proc. ICCAD'03*, pages 30–37, 2003.

[12] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads. In *Proc. ICCAD-02*, pages 721–725, 2002.

[13] F. Catthoor P. Yang. Pareto-Optimization-Based Run-Time Task Scheduling for Embedded Systems. In *Proc. CODES+ISSS '03*, pages 120–125, 2003.

[14] W. Qin and S. Malik. Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation. In *Proc. Design, Automation and Test in Europe Conf. (DATE03)*, pages 556–561, March 2003.

[15] W. Ye R. Ernst. Embedded program timing alalysis based on path clustering and architecture classification. In *Proc. ICCAD'97*, pages 598–604, 1997.

[16] A. Alvandpour S. Hsu, S. Mathew, S.-L. Lu, R. K. Krishnamurthy, and S. Borkar. A 4.5-ghz 130-nm 32-kb l0 cache with a leakage-tolerant self reverse-bias bitline scheme. *Journal of Solid State Circuits*, 38(5):755–761, May 2003.

[17] Marcus T. Schmitz and Bashir M. Al-Hashimi. Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In *Int. Symp. System Synthesis (ISSS'01)*, pages 250–255, October 2001.

[18] www.microlib.org.

[19] X. Hu Y. Zhang and D. Chen. Task Scheduling and Voltage Selection for Energy Minimization. In *Proc. IEEE DAC'02*, pages 183–188, June 2002.

[20] Y. Zhu and F. Mueller. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. In *Proc. RTAS'04*, pages 84–93, 2004.