Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints

Alexandru Andrei, Petru Ion Eles, Olivera Jovanovic, Marcus Schmitz,
Jens Ogniewski and Zebo Peng

Alexandru Andrei, Petru Ion Eles, Olivera Jovanovic, Marcus Schmitz, Jens Ogniewski and Zebo Peng, Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints, 2010, IEEE Transactions on Very Large Scale Integration (vlsi) Systems, (19), 1, 10-23.
http://dx.doi.org/10.1109/TVLSI.2009.2030199

Quasi-Static Voltage Scaling for Energy Minimization With Time Constraints

Alexandru Andrei, Petru Eles, *Member, IEEE*, Olivera Jovanovic, Marcus Schmitz, Jens Ogniewski, and Zebo Peng, *Senior Member, IEEE*

***Abstract*—Supply voltage scaling and adaptive body biasing (ABB) 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 runtime, i.e., online. However, optimal voltage scaling algorithms are computationally expensive, and thus, if used online, significantly hamper the possible energy savings. To overcome the online complexity, we propose a quasi-static voltage scaling (QSVS) scheme, with a constant online time complexity $\mathcal{O}(1)$. This allows to increase the exploitable slack as well as to avoid the energy dissipated due to online recalculation of the voltage settings.**

***Index Terms*—Energy minimization, online voltage scaling, quasi-static voltage scaling (QSVS), real-time systems, voltage scaling.**

I. Introduction and Related Work

TWO system-level approaches that allow an energy/performance tradeoff during application runtime are dynamic voltage scaling (DVS) [1]–[3] and adaptive body biasing (ABB) [2], [4]. 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 multitask systems can be broadly classified into *offline* (e.g., [1], [3], and [5][6]) and *online* (e.g., [3] and [7]–[10]) 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 runtime. In particular, Andrei *et al.* [6] present optimal algorithms as well as a heuristic for overhead-aware offline voltage selection, for real-time tasks with precedence constraints running on multiprocessor hardware architectures. On the other hand, online techniques recompute the voltage settings during runtime. Both approaches have their advantages and disadvantages. Offline voltage selection approaches avoid the computational overhead in terms of time and energy associated with the calculation of the voltage settings. However, to guarantee the fulfillment of deadline constraints, worst case execution times (WCETs) 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 ten times [11]. 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 runtime, 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 intolerably large even if low-complexity $(O(n))$ online heuristics are used instead of higher complexity optimal algorithms. Unfortunately, researchers have neglected this overhead when reporting high-quality results obtained with dynamic approaches [3], [7]–[9].

Hong and Srivastava [7] developed an online preemptive scheduling algorithm for sporadic and periodic tasks. The authors propose a linear complexity voltage scaling heuristic which uniformly distributes the available slack. An acceptance test is performed online, whenever a new sporadic task arrives. If the task can be executed without deadline violations, a new set of voltages for the ready tasks is computed.

In [8], a power-aware hard real-time scheduling algorithm that considers the possibility of early completion of tasks is proposed. The proposed solution consists of three parts: 1) an offline part where optimal voltages are computed based on the WCET, 2) an online part where slack from earlier finished tasks is redistributed to the remaining tasks, and 3) an online speculative speed adjustment to anticipate early completions of future executions. Assuming that tasks can possibly finish before their WCET, an aggressive scaling policy is proposed. Tasks are run at a lower speed than the one computed assuming the worst case, as long as deadlines can still be met by speeding up the next tasks in case the effective execution time was higher than expected. As the authors do not assume any knowledge of the expected execution time, they experiment several levels of aggressiveness.

Zhu and Mueller [9], [10] introduced a feedback earliest deadline first (EDF) scheduling algorithm with DVS for hard real-time systems with dynamic workloads. Each task is divided in two parts, representing: 1) the expected execution time, and 2) the difference between the worst case and the expected

Manuscript received December 04, 2008; revised April 16, 2009 and June 29, 2009. First published October 23, 2009; current version published December 27, 2010.

A. Andrei is with Ericsson AB, Linköping 58112, Sweden.

P. Eles and Z. Peng are with the Department of Computer and Information Science Linköping University, Linköping 58183, Sweden.

O. Jovanovic is with the Department of Computer Science XII, University of Dortmund, Dortmund 44221, Germany.

M. Schmitz is with Diesel Systems for Commercial Vehicles Robert BOSCH GmbH, Stuttgart 70469, Germany.

J. Ogniewski is with the Department of Electrical Engineering, Linköping University, Linköping 58183, Sweden.

Digital Object Identifier 10.1109/TVLSI.2009.2030199

execution time. A proportional–integral–derivative (PID) feedback controller selects the voltage for the first portion and guarantees hard deadline satisfaction for the overall task. The second part is always executed with the highest speed, while for the first part DVS is used. Online, each time a task finishes, the feedback controller adapts the expected execution time for the future instances of that task. A linear complexity voltage scaling heuristic is employed for the computation of the new voltages. On a system with dynamic workloads, their approach yields higher energy savings then an offline DVS schedule.

The techniques presented in [12]–[15] use a stochastic approach to minimize the average-case energy consumption in hard real-time systems. The execution pattern is given as a probability distribution, reflecting the chance that a task execution can finish after a certain number of clock cycles. In [12], [14], and [15], solutions were proposed that can be applied to single-task systems. In [13], the problem formulation was extended to multiple tasks, but it was assumed that continuous voltages were available on the processors.

All above mentioned online approaches greatly neglect the computational overhead required for the voltage scaling.

In [16], 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, WCETs are assumed and, thus, no dynamic slack can be exploited.

Noticeable exceptions from this broad offline/online classification are the intratask voltage selection approaches presented in [17]–[20]. The basic idea of these approaches is to perform an offline execution path analysis, and to calculate for each of the possible paths the voltage settings in advance. The resulting voltage settings are stored within the application program. During runtime the voltage settings along the activated path are selected. The execution time variation among different execution paths can be exploited, but worst case is assumed for each such path, not being possible to capture dynamic slack resulted, for example, due to cache hits. Despite their energy efficiency these approaches are most suitable for single-task systems, since the number of execution paths $p = z^n$ in multitask applications grows exponentially with the number of tasks n and depends also on the number of execution paths in a single task z.

Most of the existing work addresses the issue of energy optimization with the consideration of the dynamic slack only in the context of single-processor systems. An exception is [21], where an online approach for task scheduling and speed selection for tasks with identical power profiles and running on homogenuous processors is presented.

In this paper, we propose a quasi-static voltage scaling (QSVS) technique for energy minimization of multitask real-time embedded 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 runtime) extremely low. The obtained performance is superior to any of the previously proposed dynamic approaches. We have presented preliminary results regarding the quasi-static algorithms based on continuous voltage selection in [22].

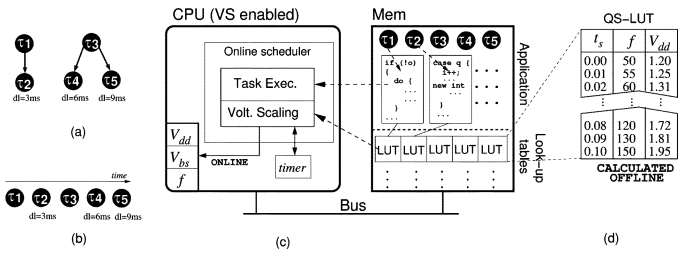

Fig. 1. System architecture. (a) Initial application model (task graph). (b) EDF-ordered tasks. (c) System architecture. (d) LUT for QSVS of one task.

II. Preliminaries

A. 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 III), 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). Once activated, tasks are running without being preempted until their completion. The tasks are executed on an embedded architecture that consists of a voltage-scalable processor (scalable in terms of *supply* and *body-bias* voltage). The power and delay model of the processor is described in Section II-B. The processor is connected to a memory that stores the application and a set of lookup tables (LUTs), 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 recalculate the performance levels. Nevertheless, calculating appropriate voltage levels (and, implicitly, the performance levels) is a computationally expensive task, i.e., it requires precious central processing unit (CPU) time, which, if avoided, would allow to lower the CPU performance and, consequently, the energy consumption.

The approach presented in this paper aims to reduce this online overhead by performing the necessary voltage selection computations offline (at compile time) and storing a limited amount of information as LUTs within memory. This information is then used during application runtime (i.e., online) to calculate the voltage and performance settings extremely fast [constant time $\mathcal{O}(1)$]; see Fig. 1(d).

In Section VIII, we will present a generalization of the approach to multiprocessor systems.

B. Power and Delay Models

Digital complementary metal–oxide–semiconductor (CMOS) circuitry has two major sources of power dissipation: 1) dynamic power P_{dyn}, which is dissipated whenever active computations are carried out (switching of logic states),

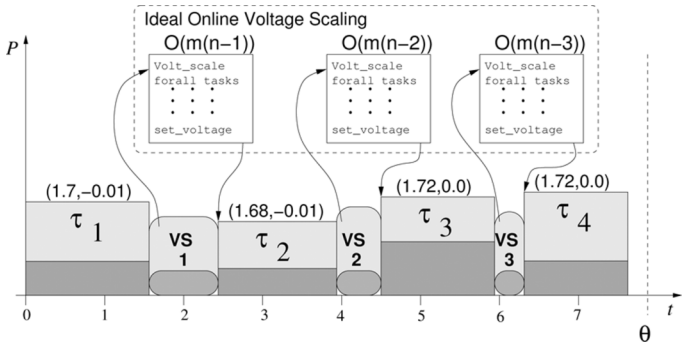

Fig. 2. Ideal online voltage scaling approach.

and 2) leakage power P_{leak} which is consumed whenever the circuit is powered, even if no computations are performed. The dynamic power is expressed by [2], [23]

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

where C_{eff}, f, and V_{dd} denote the effective charged capacitance, operational frequency, and circuit supply voltage, respectively.

The leakage power is given by [2]

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

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 [2], where also actual values for these constants are provided. Nevertheless, scaling the supply and the body-bias voltage, in order to reduce the power consumption, has a side effect on the circuit delay d, which is inverse proportional to the operational frequency f [2], [23]

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

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 [2]. Equations (1)–(3) provide the energy/performance tradeoff for digital circuits.

C. Motivation

This section motivates the proposed QSVS technique and outlines its basic idea.

1) Online Overhead Evaluation: As we have mentioned earlier, 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 illustrated in Fig. 2, where we consider an application consisting of $n = 4$ tasks. The voltage level pairs (V_{dd}, V_{bs}) used for each task are also included in the figure. 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 VS1) 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 approaches described in [5] and [24]. However, such an implementation would be only feasible if the computational overhead associated with the voltage scaling algorithm was very low, which is not the case in practice. The computational complexity of such optimal voltage scaling algorithms for monoprocessor systems is $\mathcal{O}(m \cdot n)$ [5], [24] (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) [25] and PowerPC(MPC750) , [26]. We have also performed measurements on actual implementations using an AMD platform (AMD Athlon 2400XP). Table I shows these results for two applications that can be commonly found in handheld devices: a GSM voice codec and an MPEG video encoder. Results are shown for AMD, SA-1100, and MPC750 and are given in terms of BNC and WNC numbers 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] The period corresponds to the encoding of a GSM, and respectively, of an MPEG frame. Within the period, the applications are divided in tasks (for example, the MPEG encoder consists of 25 tasks; a voltage scaling algorithm would run upon the completion of each task). For instance, on the SA-1100 processor, one iteration of the MPEG encoder requires in the BNC 4.458 kcycles and in the WNC 8.043 kcycles, which is a variation of 45%. Similarly, Table II presents the simulation outcomes for different voltage scaling algorithms. As an example, performing one single time the optimal online voltage scaling using the algorithm from [5] for 20 remaining tasks (just like VS1 is performed for the three remaining tasks τ_2, τ_3, and τ_4 in Fig. 2) requires 8410 kcycles on the AMD processor, 136 950 kcycles on the MPC750 processor, while on SA-1100, it 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 3513 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 I and II 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 210 kcycles) than decoding a single voice frame using the

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

TABLE I
SIMULATION RESULTS (CLOCK CYCLES) OF DIFFERENT APPLICATIONS

Benchmark type	AMD Athlon			SA1100			MPC750		
	BNC (kcyc)	WNC (kcyc)	Var. (%)	BNC (kcyc)	WNC (kcyc)	Var. (%)	BNC (kcyc)	WNC (kcyc)	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 II
SIMULATION RESULTS (CLOCK CYCLES) OF VOLTAGE SCALING ALGORITHMS

Voltage scaling algorithm	AMD NC (kcyc)	SA-1100 NC (kcyc)	MPC750 NC (kcyc)
OptimalVS(Vdd+Vbs, 20 tasks) [5]	8,410	1,232,552	136,950
OptimalVS(Vdd, 20 tasks) [5]	210	32,320	3,513
MTS Heuristic(Vdd, 20 tasks) [29]	8	84	12
MTS Heuristic(Vdd+Vbs, 20 tasks)	40	623	73
Greedy Heuristic(Vdd) [9]	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

GSM codec (on AMD 155 kcycles). 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 [27] has proposed a linear time heuristic, while the approaches given in [8] and [9] use a greedy heuristic of constant time complexity. We report their performance in terms of the required number of cycles in Table II, 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 [8], [9]. 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 Table II.

2) Basic Idea: QSVS: To overcome the voltage selection overhead problem, we propose a QSVS 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 precomputed based on possible task start times. The resulting voltage/frequency settings are stored in 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 3.58 ms. As indicated in Fig. 3, this start time is surrounded by the LUT entries 3.55 and 3.60 ms. In accordance, the frequency and voltage settings for task τ_3 are interpolated based on these entries. The main advantage of the online quasi-static voltage selection algorithm is its constant time complexity $\mathcal{O}(1)$. As shown in the last line of Table II, the LUT and voltage interpolation requires only 900 CPU cycles each time new voltage settings have to be calculated. Note that the complexity of the online quasi-static voltage selection is independent of the number of remaining tasks.

III. PROBLEM FORMULATION

Consider a set of NT tasks $\Pi = \{\tau_i\}$. Their execution order is fixed according to a nonpreemptive scheduling policy. Conceptually, any static scheduling algorithm can be used. We assume as given the order in which the tasks are executed. In particular, we have used an EDF ordering, in which the tasks are sorted and executed in the increasing order of their deadlines. It was demonstrated in [28] that this provides the best energy savings for single-processor systems. According to this order, task τ_i has to be executed after τ_{i-1} and before τ_{i+1}. The processor can vary its supply voltage V_{dd} and body-bias voltage V_{bs}, and consequently, its frequency f within certain continuous ranges (for the continuous optimization) or within a set of discrete modes $m_z = \{V_{dd_z}, V_{bs_z}, f_z\}$ (for the discrete optimization). The dynamic and leakage power dissipation as well as the operational frequency (clock cycle time) depend on the selected voltage pair (mode). Tasks are executed clock-cycle-by-clock-cycle and each clock cycle can be potentially executed at different voltage settings, i.e., a different energy/performance tradeoff. Each task τ_i is characterized by a six-tuple

$$\tau_i =< \mathrm{BNC}_i, \mathrm{ENC}_i, \mathrm{WNC}_i, \mathrm{Ceff}_i, dl_i >$$

where $\mathrm{BNC}_i, \mathrm{ENC}_i$, and WNC_i denote the BNC, ENC, and WNC numbers of clock cycles, respectively, which task τ_i requires for its execution. BNC (WNC) is defined as the lowest (highest) number of clock cycles task τ_i needs for its execution, while ENC is the arithmetic mean value of the probability density function $p(\mathrm{WNC})$ of the task execution cycles WNC, i.e., $\mathrm{ENC} = \sum_{j=1}^{\mathrm{WNC}} j \cdot p_j(j)$. We assume that the probability density functions of tasks' execution cycles are independent. Further, Ceff_i and dl_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 WNC. Static slack is the result of idleness due to system overperformance, observable even when tasks execute with the WNC number of cycles.

Our goal is to store a 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 LUTs (and, implicitly the total number NL of table entries) is given as a constraint.

IV. OFFLINE ALGORITHM: OVERALL APPROACH

QSVS 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.

The pseudocode corresponding to the calculations performed offline is given in Fig. 4. The algorithm requires the following input information: the scheduled task set Π, defined in Section III; for the tasks $\tau_i \in \Pi$, the expected (ENC_i), the worst case (WNC_i), and the best case (BNC_i) number of cycles,

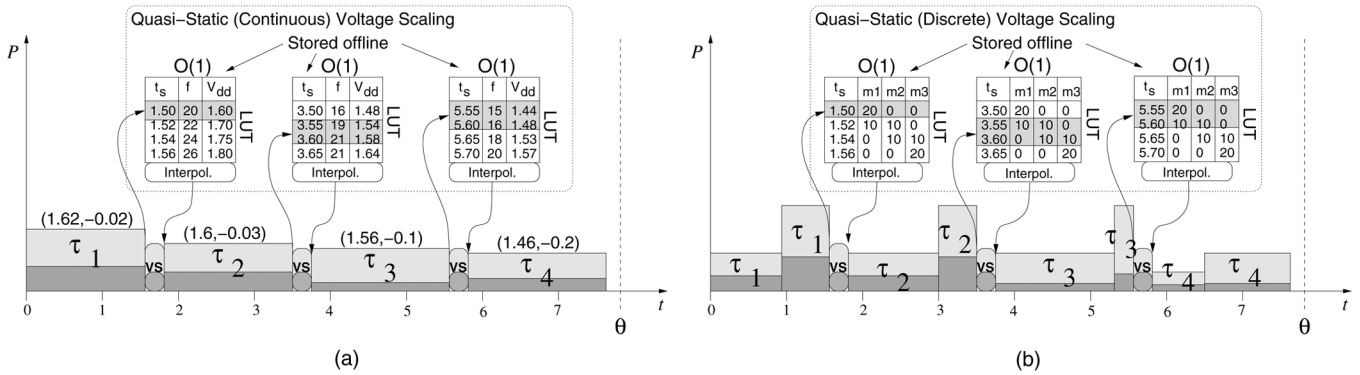

Fig. 3. QSVS based on prestored LUTs. (a) Optimization based on continuous voltage scaling. (b) Optimization based on discrete voltage.

```
Algorithm: QUASI_STATIC_VS_OFF-LINE
Input:  - execution order of tasks τ∈Π
        - for all tasks τi∈Π:
          BNCi, ENCi, WNCi, Ceffi, dli
        - NL
Output: - Look up tables LUTi
01: for i=1 to NT {
02:   ESTi ← calc_earliest_start_time
03: }//end for
04: for i=NT downto 1 {
05:   LSTi ← calc_latest_start_time
06: }//end for
07: for i=NT downto 1 {
08:   LFTi ← calc_latest_finishing_time
09: }//end for
10: Πr ← Π
11: for all τi∈Π { //ordered i=1..NT
12:   Ii ← LSTi − ESTi
13:   j ← 0
14:   ni ← comp_interpolation_points(τi, LSTi, ESTi)
15:   for (ts←ESTi; ts≤LSTi; ts←ts+Ii/n) {
16:     tsi ← ts
17: #if CONT_VS
18:     (Vddi, Vbsi, fi) ← cont_volt_scaling(Πr, tsi) //ENC based
19:     LUTi[j] ← store_QS_lookup(tsi, Vddi, f)
20: #endif
21: #if DISC_VS
22:     (tendi, hi) ← disc_volt_scaling(Πr, tsi) //ENC based
23:     LUTi[j] ← store_QS_lookup(tsi, tendi, h)
24:     compute_compat_mode_pairs();
25: #endif
26:     j ← j + 1
27:   }//end for
28:   Πr ← Πr−τi
29: }//end for all
30: for all τi∈Π return LUTi
```

Fig. 4. Pseudocode: quasi-static offline algorithm.

the effectively switched capacitance (Ceff_i), and the deadline D_i. Furthermore, the total number of LUT entries NL is given. The algorithm returns the quasi-static scaling table LUT_i, for each task $\tau_i \in \Pi$. This table includes n_i ($\sum_{i=1}^{n} n_i = \mathrm{NL}$) possible start times $t_{s_{i,j}}, j = 1, \ldots, n_i$ for each task τ_i, and the corresponding optimal settings for the supply voltage `Vdd` and the operational frequency `f`.

Upon initialization, the algorithm computes the earliest and latest possible start times as well as the latest finishing time for each task (lines 01–09). The earliest start time $\mathtt{EST}_i$ is based on the situation in which all tasks would execute with their BNC 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 $\tau_j, j \geq i$, executed with the WNC number of clock cycles at the highest voltages (lines 04–06). Similarly, we compute the latest finishing time of each task (lines 07–09).

The algorithm proceeds by initializing the set of remaining tasks Π_r with the set of all tasks Π (line 10). In the following (lines 11–29), the voltage and frequency settings for the start time intervals of each task are calculated. More detailed, in lines 12 and 13, 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 14. This will be further discussed in Section VII. For all $\mathtt{n}_i$ possible start times t_s in the start time interval of task τ_i (line 15), the task start time t_{s_i} is set to the possible start time (line 16) and the corresponding optimal voltage and frequency settings of τ_i are computed and stored in the LUT (lines 15–27). For this computation, we use the algorithms presented in [6], modified to incorporate the optimization for the expected case. Instead of optimizing the energy consumption for the WNC number of clock cycles, we calculate the voltage levels such that the energy consumption is optimal in the case the tasks execute their expected case (which, in reality, happens with a higher probability). However, since our approach targets hard real-time systems, we have to guarantee the satisfaction of all deadlines even if tasks execute their WNC number of clock cycles. In accordance with the problem formulation from Section III, the quasi-static algorithm performs the energy optimization and calculates the LUT using continuous (lines 17–20) or discrete voltage scaling (lines 21–25). We will explain both approaches in the following sections, together with their particular online algorithms. The results of the (continuous or discrete) voltage scaling for the current task τ_i, given the start time t_{s_i}, are stored in the LUT. The for-loop (line 15–27) is repeated for all n_i possible start times of task τ_i. The algorithm returns the quasi-static scaling table $\mathtt{LUT}_i$ for all tasks $\tau_i \in \Pi$.

V. Voltage Scaling With Continuous Voltage Levels

A. Offline Algorithm

In this section, we will present the continuous voltage scaling algorithm used in line 18 of Fig. 4. The problem can be formulated as a convex nonlinear optimization as follows: Minimize

$$\sum_{k=i}^{|\Pi_r|}\left(\underbrace{\mathrm{ENC}_k\cdot C_{\mathrm{eff}_k}\cdot V_{dd_k}^2}_{E_{\mathrm{dyn}_k}} + \underbrace{L_g(K_3\cdot V_{dd_k}\cdot e^{K_4 V_{dd_k}}\cdot e^{K_5\cdot V_{bs_k}}+I_{Ju}\cdot|V_{bs_k}|)\cdot t_k}_{E_{\mathrm{leak}_k}}\right) \quad (4)$$

subject to

$$s_i \geq t_{s_i} \quad (5)$$

$$t_k = \begin{cases} \mathrm{WNC}_k\cdot\dfrac{(K_6\cdot L_d\cdot V_{dd_k})}{((1+K_1)\cdot V_{dd_k}+K_2\cdot V_{bs_k}-V_{th_1})^\alpha}, \\ \quad \text{if } k=i \\ \mathrm{ENC}_k\cdot\dfrac{(K_6\cdot L_d\cdot V_{dd_k})}{((1+K_1)\cdot V_{dd_k}+K_2\cdot V_{bs_k}-V_{th_1})^\alpha} \\ \quad \forall\tau_k\in\Pi_r k\neq i \end{cases} \quad (6)$$

$$s_k+t_k\leq s_{k+1} \qquad \forall\tau_k,\qquad k=1\ldots(|\Pi_r|-1) \quad (7)$$

$$s_k+t_k\leq dl_k \qquad \forall\tau_k\in\Pi_r \text{ with deadline} \quad (8)$$

$$s_i+t_i\leq \mathrm{LFT}_i,\qquad \tau_i \text{ is the first task in } \Pi_r \quad (9)$$

$$s_k\geq 0 \qquad \forall\tau_k\in\Pi_r \quad (10)$$

$$V_{dd_{\min}}\leq V_{dd_k}\leq V_{dd_{\max}}$$
$$V_{bs_{\min}}\leq V_{dd_k}\leq V_{bs_{\max}} \qquad \forall\tau_k\in\Pi_r. \quad (11)$$

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 start time of the current task has to match the start time assumed for the currently calculated LUT entry (5). 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 in the objective. 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 (6). In order to guarantee that the current task τ_i end before the deadline, its execution time t_i is calculated using the WNC number of cycles WNC_i. Remember from the computation of the latest finishing time (LFT) that if task τ_i finishes its execution before LFT_i, then the rest of the tasks are guaranteed to meet their deadlines even in the WNC. This condition is enforced by (9).

As opposed to the current task τ_i, for the remaining $\tau_k\in\Pi_r, k\neq i$, the expected number of clock cycles ENC_k is used when calculating their execution time in (6). This is important for a distribution of the slack that minimizes the energy consumption in the expected case. Note that this is possible because after performing the voltage selection algorithm, only the results for the current task are stored in the LUT. The settings calculated for the rest of the tasks are discarded.

```
Algorithm: QUASI_STATIC_VS_ONLINE_CONTINUOUS

Input:  - start time t_sn of next task τ_n
        - Quasi-Static Scaling Table LUT_n
        - number of start time interval steps LUT_SIZE_n
        - latest finishing time LFT_n of task τ_n
Output: - frequency and voltage settings for task τ_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: if t_sn + WNC_n/f_n > LFT_n {
05:     f_n ← f_y
06:     Vdd_n ← Vdd_y
07: }
08: Vbs_n ← calc_Vbs(f_n,Vdd_n)
09: return (f_n,Vdd_n,Vbs_n)
```

Fig. 5. Pseudocode: continuous online algorithm.

The rest of the nonlinear formulation is similar to the one presented in [6], for solving, in polynomial time, the continuous voltage selection without overheads. The consideration of switching overheads is discussed in Section VI-C. Equation (7) expresses the task execution order, while deadlines are enforced in (8). The above formulation can handle arrival times for each task by replacing the value 0 with the value of the given arrival times in (10).

B. Online Algorithm

Having prepared, for all tasks of the system, a set of possible voltage and frequency settings depending on the task start time, we outline next how this information is used online to compute the voltage and frequency settings for the effective (i.e., actual) start time of a task. 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 $\mathtt{LUT\_SIZE}_n$. As an 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 that 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 $\mathtt{LUT}_n$`[x]` and $\mathtt{LUT}_n$`[y]` (line 02). Similarly, in step 03, the supply voltage $\mathtt{Vdd}_n$ is linearly interpolated from the two surrounding voltage entries in $\mathtt{LUT}_n$.

As shown in [29], however, task frequency considered as a function of start time is not convex on its whole domain, but only piecewise convex. Therefore, if the frequencies from $\mathtt{LUT}_n$`[x]` and $\mathtt{LUT}_n$`[y]` are not on a convex region, no guarantees regarding the resulting real-time behavior can be made. The online algorithm handles this issue in line 04. If the task uses the interpolated frequency and, assuming it would execute the WNC number of clock cycles, it would exceed its latest finishing time; the frequency and supply voltage are set to the ones from

$\texttt{LUT}_n\texttt{[y]}$ (line 05–06). This guarantees the correct real-time execution, since the frequency from $\texttt{LUT}_n\texttt{[y]}$ was calculated assuming a start time later than the actual one.

We do not directly interpolate the setting for the body-bias voltage $\texttt{Vbs}_n$, due to the nonlinear relation between frequency, supply voltage, and body-bias voltage. We calculate the body-bias voltage directly from the interpolated frequency and supply voltage values, using (3) (line 08). The algorithm returns the settings for the frequency, supply and body-bias voltage (line 09). The time complexity of the quasi-static online algorithm is $\mathcal{O}(1)$.

VI. Voltage Scaling Algorithm With Discrete Voltage Levels

We consider that processors can run in different modes $m \in \mathcal{M}$. Each mode m is characterized by a voltage pair (V_{dd_m}, V_{bs_m}) that determines the operational frequency f_m, the normalized dynamic power P_{dnom_m}, and the leakage power dissipation P_{leak_m}. The frequency and the leakage power are given by (3) and (2), respectively. The normalized dynamic power is given by $P_{\text{dnom}_m} = f_m \cdot V_{dd_m}^2$.

A. Offline Algorithm

We present the voltage scaling algorithm used in line 22 of Fig. 4. The problem is formulated using mixed integer linear programming (MILP) as follows.

Minimize

$$\sum_{k=i}^{|\Pi_r|} \sum_{m \in \mathcal{M}} \left(C_{\text{eff}_k} \cdot P_{\text{dnom}_m} \cdot t_{k,m} + P_{\text{leak}_m} \cdot t_{k,m}\right) \tag{12}$$

subject to

$$s_i \geq t_{s_i} \tag{13}$$

$$c_{k,m} = t_{k,m} \cdot f_m \qquad \forall \tau_k \in \Pi_r, \qquad m \in \mathcal{M} \tag{14}$$

$$\sum_{m \in \mathcal{M}} c_{k,m} = \begin{cases} \text{WNC}_k, & \text{if} k = i \\ \text{ENC}_k, & \text{if} k \neq i \end{cases} \tag{15}$$

$$s_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq dl_k \qquad \forall \tau_k \in \Pi_r \text{ with deadline} \tag{16}$$

$$s_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq s_{k+1} \qquad \forall \tau_k, \qquad k = 1, \ldots, (|\Pi_r| - 1) \tag{17}$$

$$s_i + \sum_{m \in \mathcal{M}} t_{i,m} \leq \text{LFT}_i \quad \tau_i \text{ is the current task} \tag{18}$$

$$s_k \geq 0 \qquad \forall \tau_k \in \Pi_r \tag{19}$$

$$t_{k,m} \geq 0 \text{ and } c_{k,m} \text{ is integer} \qquad \forall \tau_k \in \Pi_r. \tag{20}$$

The task execution time $t_{k,m}$ and the number of clock cycles $c_{k,m}$ spent within a mode are the variables in the MILP formulation. The number of clock cycles has to be an integer and hence $c_{k,m}$ is restricted to the integer domain (20). The total energy consumption to be minimized, expressed by the objective in (12), is given by two sums. The inner sum indicates the energy dissipated by an individual task τ_k, depending on the time $t_{k,m}$ spent in each mode m, while the outer sum adds up the energy of all tasks. Similar to the continuous algorithm from Section V-A, the expected number of clock cycles is used for each task in the objective function.

The start time of the current task has to match the start time assumed for the currently calculated LUT entry (13). The relation between execution time and number of clock cycles is expressed in (14). For similar reasons as in Section V-A, the WNC number of clock cycles WNC_i is used for the current task τ_i. For the remaining tasks, the execution time is calculated based on the expected number of clock cycles ENC_k. In order to guarantee that the deadlines are met in the worst case, (18) forces task τ_i to complete in the worst case before its latest finishing time LFT_i.

Similar to the continuous formulation from Section V-A, (16) and (17) are needed for distributing the slack according to the expected case. Furthermore, arrival times can also be taken into consideration by replacing the value 0 in (19) with a particular arrival time.

As shown in [6], the discrete voltage scaling problem is NP hard. Thus, performing the exact calculation inside an optimization loop as in Fig. 4 is not feasible in practice. If the restriction of the number the clock cycles to the integer domain is relaxed, the problem can be solved efficiently in polynomial time using linear programming. The difference in energy between the optimal solution and the relaxed problem is below 1%. This is due to the fact that the number of clock cycles is large and thus the energy differences caused by rounding a clock cycle for each task are very small.

Using this linear programming formulation, we compute offline for each task τ_i the number of clock cycles to be executed in each mode and the resulting end time, given several possible start times. At this point it is interesting to make the following observations.

For each task, if the variables $c_{k,i}$ are not restricted to the integer domain, after performing the optimal voltage selection computation, the resulting number of clock cycles assigned to a task is different from zero for at most two of the modes. The demonstration is given in [29].[2] This property will be used by the online algorithm outlined in the next section.

Moreover, for each task, a table of the so-called compatible_modes can be derived offline. Given a mode m_h, there exists one single mode m_l with $f_l \leq f_h$ such that the energy obtained using the pair (m_h, m_l) is lower than the energy achievable using any other mode m_j $(j \neq l, f_j < f_h)$ paired with m_h. We will refer to two such modes m_l and m_h as *compatible*. The compatible mode pairs are specific for each task. In the example illustrated by Fig. 6(b), from all possible mode combinations, the pair that provides the best energy

[2]Ishihara and Yasuura [1] present a similar result, that is given a certain execution time, using two frequencies will always provide the minimal energy consumption. However, what is not mentioned there is that this statement is only true if the numbers of clock cycles to be executed using the two frequencies are not integers. The flaw in their proof is located in (8), where they assume that the execution time achievable by using two frequencies is identical to the execution time achievable using three frequencies. When the numbers of clock cycles are integers, this is mathematically not true. In our experience, from a practical perspective, the usage of only two frequencies provides energy savings that are very close to the optimal. However, the selection of the two frequencies must be performed carefully. Ishihara and Yasuura [1] propose the determination of the discrete frequencies as the ones surrounding the continuous frequency calculated by dividing the available execution time by the WNC number of clock cycles. However, this could lead to a suboptimal selection of two incompatible frequencies.

t_s	m1	m2	m3
1.50	20	0	0
1.52	10	10	0
1.54	0	10	10
1.56	0	0	20

LUT

(a)

t_s	t_e	h
1.50	1.70	1
1.52	1.71	2
1.54	1.72	3
1.56	1.72	3

LUT

h	l
1	1
2	1
3	2
4	3

Compatible mode pairs

(b)

Fig. 6. LUTs with discrete modes.

savings is among the ones stored in the compatible_modes table.

In the following, we will present the equation used by the offline algorithm for the determination of the compatible mode pairs. Let us denote with $e_{k,i}$ the energy consumed per clock cycle by task τ_k running in mode m_i. If the modes m_h and m_l are compatible, with $f_l \leq f_h$, we have shown in [29] that the following holds:

$$e_{k,l} \cdot \left(\frac{1}{f_j} - \frac{1}{f_h}\right) - e_{k,j} \cdot \left(\frac{1}{f_l} - \frac{1}{f_h}\right) < e_{k,h}\left(\frac{1}{f_j} - \frac{1}{f_l}\right), \quad \forall j = 1 \ldots |\mathcal{M}|. \quad (21)$$

It is interesting to note that (21) and consequently the choice of the mode with a lower frequency, given the one with a higher frequency, depend only on the task power profile and the on frequencies that are available on the processor. For a certain available execution time, there is at least one "high" mode that can be used together with its compatible "low" mode such that the timing constraints are met. If several such pairs can potentially be used, the one that provides the best energy consumption has to be selected. More details regarding this issue are given in the next section. To conclude the description of the discrete offline algorithm, in addition to the LUT calculation, the table compatible_modes is also computed offline (line 24 of Fig. 4), for each task. The computation is based on (21). The pseudocode for this algorithm is given in [29].

B. Online Algorithm

We present in this section the algorithm that is used online to select the discrete modes and their associated number of clock cycles for the next task τ_n, based on the actual start time and precomputed values from $\mathtt{LUT}_n$.

In Section V-B, for the continuous voltages case, every LUT entry contains the frequency calculated by the voltage scaling algorithm. At runtime, a linear interpolation of the two consecutive LUT entries with start times surrounding the actual start time of the next task is used to calculate the new frequency. As opposed to the continuous calculation, in the discrete case, a task can be executed using several frequencies. This makes the interpolation difficult.

1) Straightforward Approach: Let us assume, for example, a LUT like the one illustrated in Fig. 6(a), and a start time of 1.53 for task τ_2. The LUT stores, for several possible start times, the number of clock cycles associated to each execution mode, as calculated by the offline algorithm. Following the same approach as in the continuous case, based on the actual start time, the number of clock cycles for each performance mode should be interpolated using the entries with start times at 1.52 and 1.54.

```
Algorithm: QUASI_STATIC_VS_ONLINE_DISC
Input:  - start time t_sn of current task τ_n
        - Quasi-Static Scaling Table LUT_n
        - number of start time interval steps LUT_SIZE_n
Output: - active modes l and h the corresponding
          number of clock cycles c_l and c_h for τ_n
01: (x,y) ← calc_st_interval(LUT_n, t_sn)
02: t_endn ← maximum(t_endx, t_endy)
03: min=∞
04: t_exe = t_endn − t_sn
05: for j = h_y downto h_x {
06:   if WNC_n/f_j ≥ t_exe break;
07:   c=compatible_mode[j];
08:   (nc_c, nc_j)=compute_number_of_cycles(c,j);
09:   e=compute_task_energy(nc_c, nc_j, c, j);
10:   if (e<min) {
11:     min=e; l=c; h=j; nc_l = nc_c; nc_h = nc_j;
12:   }
13: }
14: return (l, h, nc_l, nc_h);
```

Fig. 7. Pseudocode: discrete online algorithm.

However, such a linear interpolation cannot guarantee the correct hard real-time execution. In order to guarantee the correct timing, among the two surrounding entries, the one with a higher start time has to be used. For our example, if the actual start time is 1.53, the LUT entry with start time 1.54 should be used. The drawback of this approach, as will be shown by the experimental results in Section IX, is the fact that a slack of $1.54-1.53 = 0.01$ time units cannot be exploited by the next task.

2) Efficient Online Calculation: Let us consider a LUT like in Fig. 6(a). It contains for each start time the number of clock cycles spent in each mode, as calculated by the offline algorithm. As discussed in Section VI-A, maximum two modes have the number of cycles different from zero. Instead, however, of storing this "extended" table, we store a LUT like in Fig. 6(b), which, for each start time contains the corresponding end time as well as the mode with the highest frequency. Moreover, for each task, the table of compatible modes is also calculated offline. The online algorithm is outlined in Fig. 7. The input consists of the task actual start time t_{s_n}, the quasi-static scaling table $\mathtt{LUT}_n$, the table with the compatible modes, and the number of interval steps LUT_SIZE_n. As an output, the algorithm returns the number of clock cycles to be executed in each mode. In line 01, similar to the continuous online algorithm presented in Section V-B, we must find the LUT entries x and y surrounding the actual start time. In the next step, using the end time values from the lines x and y, together with the actual start time, we must calculate the end time of the task (line 02). The algorithm selects as the end time for the next task the maximum between the end times from the LUT entries x and y. In this way, the hard real-time behavior is guaranteed.

At this point, given the actual start and the end time, we must determine the two active modes and the number of clock cycles to be executed in each. This is done in lines 05–13. From the LUT, the upper and lower bounds h_y and h_x of the higher execution mode are extracted. Using the table of compatible modes calculated offline, for each possible pair having the higher mode in the interval $[h_x, h_y]$, a number of cycles in each mode are calculated (line 07–08). compute_number_of_cycles is equivalent

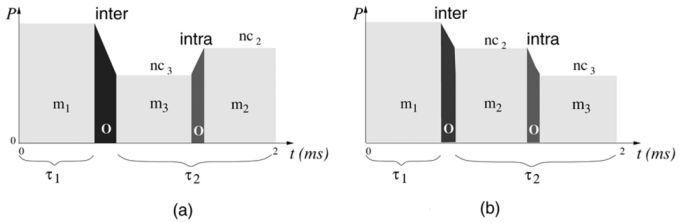

Fig. 8. Mode transition overheads. (a) Executing before in mode m_3. (b) Executing before in mode m_2.

to the following system of equations, used for calculating nc_c and nc_j:

$$nc_c + nc_j = \text{WNC}_n, \quad \frac{nc_c}{f_c} + \frac{nc_j}{f_j} = t_{\text{exe}}. \tag{22}$$

The resulting energy consumption is evaluated and the pair that provides the lowest energy is selected (lines 09–12). The algorithm finishes either when all the modes in the interval $[h_x, h_y]$ have been inspected, or, when, during the for loop, a mode m_i that cannot satisfy the timing requirements is encountered (line 06).

The complexity of the online algorithm increases linearly with the number of available performance modes $|\mathcal{M}|$. It is important to note that real processors have only a reduced set of modes. Furthermore, due to the fact that we use consecutive entries from the LUT, the difference $h_y - h_x$ will be small (typically 0, 1), leading to a very low online overhead.

C. Consideration of the Mode Transition Overheads

As shown in [6], it is important to carefully consider the overheads resulted due to the transitions between different execution modes. We have presented in [6] the optimal algorithms as well as a heuristic that addresses this issue, assuming an offline optimization for the WNC. The consideration of the mode switching overheads is particularly interesting in the context of the expected case optimization presented in this paper. Furthermore, since the actual modes that are used by the tasks are only known at runtime, the online algorithm has to be aware and handle the switching overheads. We have shown in Section VI-A that at most two modes are used during the execution of a task. The overhead-aware optimization has to decide, at runtime, in which order to use these modes. Intuitively, starting a task in a mode with a lower frequency can potentially lead to a better energy consumption than starting at a higher frequency. But this is not always the case. We addresses this issue in the remainder of this section.

Let us first consider the example shown in Fig. 8. Task τ_1 is running in mode m_1 and has finished. The online algorithm (Fig. 7) has decided the two modes m_2 and m_3 in which to run task τ_2. It has also calculated the number of clock cycles nc_3 and nc_2 to be executed in modes m_3 and m_2, respectively, in the case that τ_2 executes its WNC number of cycles WNC_2. Now, the online algorithm has to decide which mode m_3 or m_2 to apply first. The expected number of cycles for τ_2 is ENC_2, and the energy consumed per clock cycle in mode i is e_i.

Let us assume, for example, that $nc_2 = 100, nc_3 = 100, e_2 = 2, e_3 = 1$ and that τ_2 executes its expected number of cycles $\text{ENC}_2 = 175$. The alternative illustrated in Fig. 8(a), with τ_2 executing 100 cycles in the lower mode m_3, and then finishing early after executing only 75 more clock cycles in the higher mode m_2, leads to an energy consumption of $100 \cdot 1 + 75 \cdot 2 = 250$. On the other hand, if τ_2 executes first 100 clock cycles in mode m_2, and then finishes early after executing 75 cycles in mode m_3, the energy consumed by the task is $100 \cdot 2 + 75 \cdot 1 = 275$.

However, the calculation from the previous paragraph did not consider the overheads associated to mode changes. As shown in [2], the transition overheads depend on the voltages characteristic for the two execution modes. In this section, we assume they are given. Let us denote the energy overhead implied by a transition between mode i and j with $\epsilon_{i,j}$. If we examine the schedules presented in Fig. 8(a) and (b), we notice that in the first case, the energy overhead is $\epsilon_{1,3} + \epsilon_{3,2}$ versus $\epsilon_{1,2} + \epsilon_{2,3}$ for the second schedule. We denote the energy resulted from the schedule in Fig. 8(a) and (b) by E_a and E_b, respectively. For example, if $\epsilon_{1,3} = 30, \epsilon_{3,2} = 25, \epsilon_{1,2} = 25$, and $\epsilon_{2,3} = 25$, then $E_a = 250 + 30 + 25 = 305$ and $E_b = 275 + 25 + 25 = 325$. However, if $\epsilon_{1,3} = 55$ (and the rest of the overheads remain unchanged), then $E_a = 250 + 55 + 25 = 330$ and $E_b = 275 + 25 + 25 = 325$. This example demonstrates that the mode transition overheads must be considered during the online calculation when deciding at what frequency to start the next task.

We will present in the following the online algorithm that addresses this issue. The input parameters are: the last execution mode that was used by the previous task m_k, the expected number of clock cycles of the next task τ_n, ENC_n, and the two modes (m_l, m_h) with the corresponding numbers of clock cycles (nc_l, nc_h) calculated by the algorithm in Fig. 7 for the next task. The result of this algorithm is the order of the execution modes: the execution of the task will start in mode m_l or m_h. Let us assume that the frequency associated to m_l is lower than the one associated to m_h. In essence, the algorithm chooses to start the execution in the mode with the lowest frequency m_l, as long as the transition overhead in terms of energy between the mode m_k and m_l can be compensated by the achievable savings assuming the task will execute its expected number of clock cycles ENC_n.

The online algorithm examines four possible scenarios, depending on the relation between ENC_n, nc_l and nc_h.

1) $nc_l \geq \text{ENC}_n, nc_h \geq \text{ENC}_n$. In this case, it is expected that only one mode will be used at runtime, since in both execution modes m_l and m_h we have a number of clock cycles higher than the expected one. Let us calculate the energies E_a consumed when starting the execution of τ_n in m_l and E_b consumed when starting the execution in mode m_h

$$E_a = \epsilon_{k,l} + \text{ENC}_n \cdot e_l \tag{23}$$

$$E_b = \epsilon_{k,h} + \text{ENC}_n \cdot e_h. \tag{24}$$

In this case, it is more energy efficient to begin the execution of τ_n in mode m_l ($E_a \leq E_b$), if $\text{ENC}_n \geq (\epsilon_{k,l} - \epsilon_{k,h})/(e_h - e_l)$.

2) $nc_l \leq \text{ENC}_n, nc_h \leq \text{ENC}_n$. In opposition to the previous case when it is likely that only one mode is used online, in this case, it is expected that both modes will be used at runtime. The energy consumption in each alternative is

$$E_a = \epsilon_{k,l} + x \cdot e_l + \epsilon_{l,h} + (\text{ENC}_n - nc_l) \cdot e_h \tag{25}$$

$$E_b = \epsilon_{k,h} + y \cdot e_h + \epsilon_{h,l} + (\text{ENC}_n - nc_h) \cdot e_l. \tag{26}$$

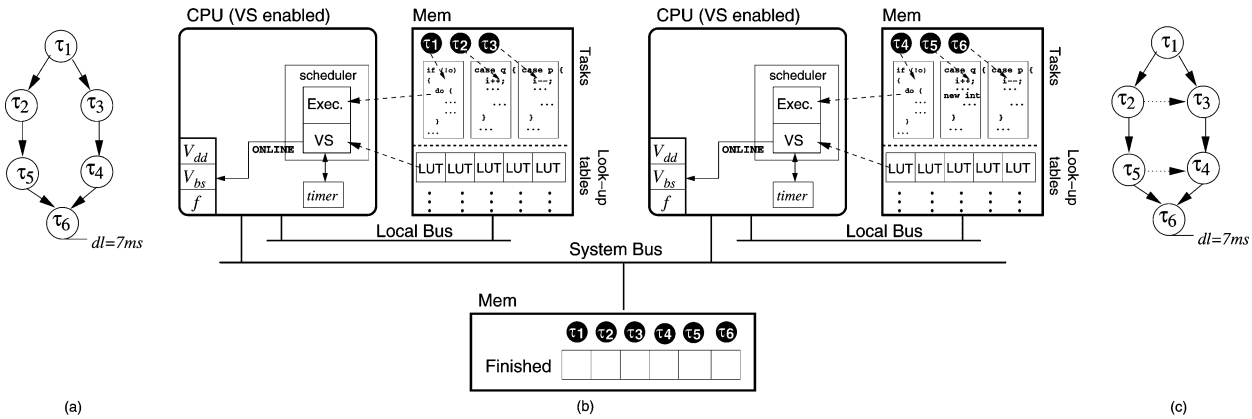

Fig. 9. Multiprocessor system architecture. (a) Task graph. (b) System model. (c) Mapped and scheduled task graph.

Thus, it is more energy efficient to begin the execution of τ_n in mode m_l if $nc_l + nc_h - \text{ENC}_n \geq (\epsilon_{k,l} + \epsilon_{l,h} - \epsilon_{k,h} - \epsilon_{h,l})/(e_h - e_l)$.

3) $nc_l \geq \text{ENC}_n, nc_h \leq \text{ENC}_n$. In this case, assuming the execution starts in m_l, it is expected that the task will finish before switching to m_h. Alternatively, if the execution starts in m_h, after executing nc_h clock cycles, the processor must be taken from m_h to m_l where additional $(\text{ENC}_n - nc_h)$ will be executed

$$E_a = \epsilon_{k,l} + \text{ENC}_n \cdot e_l \tag{27}$$

$$E_b = \epsilon_{k,h} + y \cdot e_h + \epsilon_{h,l} + (\text{ENC}_n - nc_h) \cdot e_l. \tag{28}$$

It is more energy efficient to begin the execution of τ_n in mode m_l if $nc_h \geq (\epsilon_{k,l} - \epsilon_{k,h} - \epsilon_{h,l})/(e_h - e_l)$.

4) $nc_l \leq \text{ENC}_n, nc_h \geq \text{ENC}_n$. Similarly to the previous case, if $nc_l \geq (\epsilon_{k,l} - \epsilon_{k,h} + \epsilon_{l,h})/(e_h - e_l)$, it is better to start τ_n in mode m_l.

As previously mentioned, these four possible scenarios are investigated at the end of the online algorithm in Fig. 7 in order to establish the order in which the execution modes are activated for the next task τ_n.

VII. Calculation of the LUT 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. The number of entries in the LUT of each task has an influence on the solution quality, i.e., the energy consumption. This is because the approximations in the online algorithm become more accurate as the number of points increases.

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. The two main parameters that determine the criticality (in the sense that it should be allocated more entries in the LUT) of a task τ_i are the size of the interval of possible start times $(\text{LST}_i - \text{EST}_i)$ and the nominal expected energy consumption (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 LUT entries for each tasks, we use the following formula:

$$n_i = \text{NL} \cdot \frac{E_i \cdot (\text{LST}_i - \text{EST}_i)}{\sum_{i=1}^{NT} E_i \cdot (\text{LST}_i - \text{EST}_i)}. \tag{29}$$

VIII. QSVS for Multiprocessor Systems

In this section, we address the online voltage scaling problem for multiprocessor systems. We consider that the applications are modeled as task graphs, similarly to Section II-A with the same parameters associated to the tasks. The mapping of the tasks on the processors and the schedule are given, and are captured by the mapped and scheduled task graph. Let us consider the example task graph from Fig. 9(a) that is mapped on the multiprocessor hardware architecture illustrated in Fig. 9(b). In this example, tasks τ_1, τ_2, and τ_3 are mapped on processor 1, while tasks τ_4, τ_5, and τ_6 are mapped on processor 2. The scheduled task graph from Fig. 9(c) captures along with the data dependencies [Fig. 9(a)] the scheduling dependencies marked with dotted arrows (between τ_2 and τ_3 and between τ_5 and τ_4).

Similarly to the single-processor problem, the aim is to reduce the energy consumption by exploiting dynamic slack resulted from tasks that require less execution cycles than in their WNC. For efficiency reasons, the same quasi-static approach, based on storing a LUT_i for each task τ_i, is used.

The hardware architecture is depicted in Fig. 9, assuming, for example, a system with two processors. Note that each processor has a dedicated memory that stores the instructions and data for the tasks mapped on it, and their LUTs. The dedicated memories are connected to the corresponding processor via a local bus. The shared memory, connected to the system bus, is used for synchronization, recording for each task whether it has completed the execution. When a task ends, it marks the corresponding entry in the shared memory. This information is used by the scheduler, invoked when a task finishes, on the processor where the finished task is mapped. The scheduler has to decide when to start and which performance modes to assign to the next task on that processor. The next task, determined by an offline schedule, can start only when its all predecessors have finished. The performance modes are calculated using the LUTs.

The quasi-static algorithms presented in Sections V and VI were designed for systems with a single processor. Nevertheless, they can also be used in the case of multiprocessor systems, with a few modifications.

1) Continuous Approach: In the offline algorithm from Section V-A, (7) that captures the precedence constraints between tasks has to be replaced by

$$s_k + t_k \leq s_l \qquad \forall (k,l) \in \mathcal{E}. \tag{30}$$

$\mathcal{E}$ is the set of all edges in the extended task graph (precedence constrains and scheduling dependencies).

2) Discrete Approach: In the offline algorithm from Section VI-A, (17) that captures the precedence constraints has to be replaced by

$$s_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq s_l \qquad \forall (k,l) \in \mathcal{E}. \tag{31}$$

Both online algorithms described in Sections V-B and VI-B can be used without modifications.

At this point, it is interesting to note that the correct real-time behavior is guaranteed also in the case of a multiprocessor system. The key is the fact that all the tasks, no matter when they are started, will complete before or at their latest finishing time.

IX. Experimental Results

We have conducted several experiments using numerous generated benchmarks as well as two real-life applications, in order to demonstrate the applicability of the proposed approach. The processor parameters have been adopted from [2]. The transition overhead corresponding to a change in the processor settings was assumed to be 100 clock cycles.

The first set of experiments was conducted in order to investigate the quality of the results provided by different online voltage selection techniques in the case when their actual runtime overhead is ignored. In Fig. 10(a), we show the results obtained with the following five different approaches:

1) the ideal online voltage selection approach (the scheduler that calculates the optimal voltage selection with no overhead);
2) the quasi-static voltage selection technique proposed in Section V;
3) the greedy heuristic proposed in [8];
4) the task splitting heuristic proposed in [9];
5) the ideal online voltage scaling algorithm for WNC proposed in [3].

Originally, approaches given in [3], [8], and [9] 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 lower limit as baseline for the comparison. During the experiments, we varied the ratio of actual number of clock cycles (ANC) and 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 10 000 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 and energy. Furthermore, the ANCs are set based on a normal distribution using the ENC as the mean value. Observing Fig. 10(a) leads to the following interesting conclusions. First, if the ANC corresponds to the WNC number, all voltage selection techniques approach the theoretical limit. In other words, if the application has been scaled for the WNC and all tasks execute with WNC, then all online 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 voltage selection is 25% off the hypothetical limit. On the other hand, the technique described in [3] is 60% worse than the hypothetical limit. The approaches based on the methods proposed in [8] and [9] 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 voltage selection depends on the number of entries that are stored in the LUTs. Due to the importance of this influence, we have devoted a supplementary experiment to demonstrate how a number of entries affect the voltage selection quality (this issue is further detailed by the experiments from Fig. 11, explained below). In the experiment illustrated in Fig. 10(a)-(c), 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. 10(a) demonstrates the high quality of the voltage settings produced by the quasi-static approach, which are very close to those produced by the ideal algorithm and substantially better than the values produced by any other proposed approach.

In order to evaluate the global quality of the different voltage selection approaches (taking into consideration the online overheads), we conducted two sets of experiments [Fig. 10(b) and (c)]. In Fig. 10(b), we have compared our quasi-static algorithm with the approaches proposed in [8] and [9]. The influence of the overheads is tightly linked with the size of the applications. Therefore, we use two sets of tasks graphs (set 1 and set 2) of different sizes. The task graphs from set 1 have the size (total number of clock cycles) comparable to that of the MPEG encoder. The ones used in set 2 have a size similar to the GSM codec. As we can observe, the proposed QSVS achieves considerably higher savings than the other two approaches. Although all three approaches illustrated in Fig. 10(b) have constant online complexity $(O(1))$, the overhead of the quasi-static approach is lower, and at the same time, as shown in Fig. 10(a), the quality of settings produced by QSVS is higher.

In Fig. 10(c), we have compared our quasi-static approach with a supposed "best possible" DVS algorithm. Such a supposed algorithm would produce the optimal voltage settings with a linear overhead similar to that of the heuristic proposed in [27] (see Table II). Note that such an algorithm has not been proposed since all known optimal solutions incur a higher complexity than the one of the heuristic in [27]. We evaluated

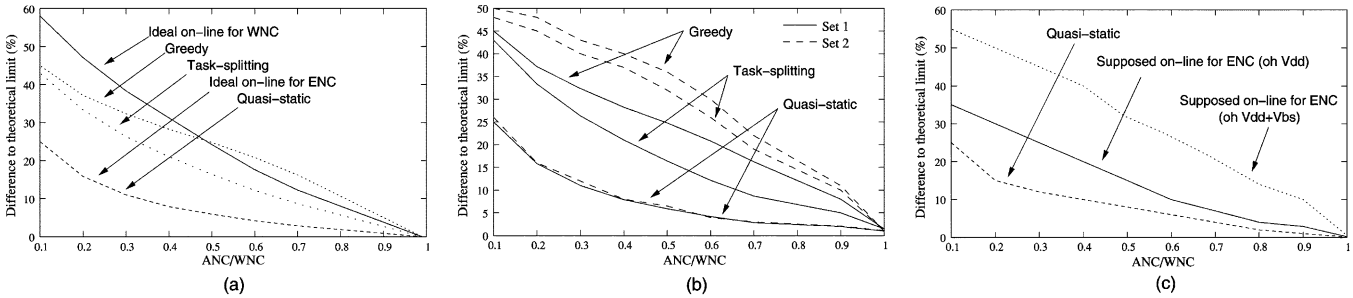

Fig. 10. Experimental results: online voltage scaling. (a) Scaling for the expected-case execution time (assuming zero overhead). (b) Influence of the online overhead on different online VS approaches. (c) Influence of the online overhead on a supposed linear time heuristic.

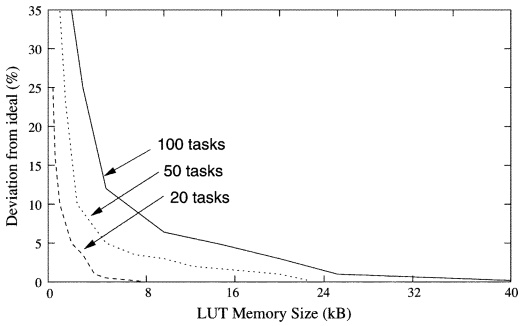

Fig. 11. Experimental results: influence of LUT sizes.

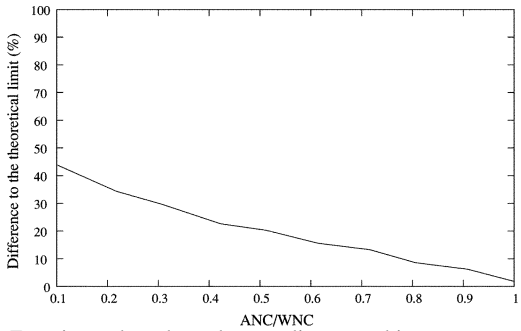

Fig. 13. Experimental results: voltage scaling on multiprocessor systems.

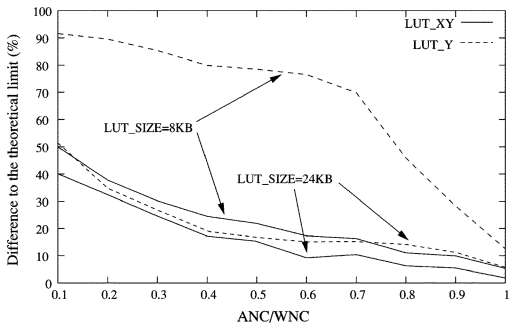

Fig. 12. Experimental results: discrete voltage scaling.

10 000 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 [27] for V_{dd}-only and with the overhead that would result if the algorithm was rewritten for the combined (V_{dd}, V_{bs}) scaling. Please note that in both of the above cases we consider that the supposed 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 supposed algorithm with the lower V_{dd}-only overhead, while in the 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 DVS approach.

The next set of experiments was conducted in order to demonstrate the influence of the memory size used for the LUTs on the possible energy savings with the QSVS. For this experiment, we have used three sets of tasks graphs with 20, 50, and 100 tasks, respectively. Fig. 11 shows the percentage deviation of energy savings with respect to an ideal online voltage selection as a function of the memory size. For example, in order to obtain a deviation below 0.5%, a memory of 40 kB is needed for systems consisting of 100 tasks. For the same quality, 20 and 8 kB are needed for 50 and 20 tasks, respectively. When using very small memories (corresponding to only two LUT entries per task), for 100 tasks, the deviation from the ideal is 35.2%. Increasing only slightly, the LUT memory to 4 kB (corresponding to in average eight LUT entries per task), the deviation is reduced to 12% for 100 tasks. 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 24 kB. It is important to note that in all the performed experiments we have taken into consideration the energy overhead due to the memories. These overheads have been calculated based on the energy values reported in [30] and [31] in the case of SRAM memories.

In the experiments presented until now, we have used the quasi-static algorithm based on continuous voltage selection. As mentioned in the opening of this section, we have used processor parameters from [2]. These, together with (1)– (3), fulfill the mathematical conditions required by the convex optimization used in the offline algorithm presented in Section V-A. While we do not expect that future technologies will break the convexity of the power and frequency functions, this cannot be completely excluded. The algorithm presented in Section VI-A makes no such assumptions.

Another set of experiments was performed in order to evaluate the approach based on discrete voltages, presented in Section VI. We have used task graphs with 50 tasks and a processor with four discrete modes. The four voltage pairs

(V_{dd}, V_{bs}) are: (1.8 V, 0 V), (1.4 V, −0.3 V), (1.0 V, −0.6 V), and (0.6 V, −1.0 V). The processor parameters have been adopted from [2]. An overhead of 100 clock cycles was assumed for a mode transition overhead. The results are shown in Fig. 12. During this set of experiments, we have compared the energy savings achievable by two discrete quasi-static approaches. In the first approach, the LUT stores for each possible start time the number of clock cycles associated to each mode. Online, the first LUT entry that has the start time higher than the actual one is selected. This approach, corresponding to the straightforward pessimistic alternative proposed in Section VI-B1 is denoted in Fig. 12 with LUT_Y. The second approach uses the efficient online algorithm proposed in Section VI-B2, and is denoted in Fig. 12 with LUT_XY. During the experiments, we varied the ratio of ANC to WNC from 0.1 to 1, with a step width of 0.1. For each step, 1000 randomly generated task graphs were evaluated. As indicated in the figure, we present the deviation from the hypothetical limit for the two approaches, assuming two different LUT sizes: 8 and 24 kB. Clearly, the savings achieved by both approaches depend on the size of the LUTs. We notice in Fig. 12 that the size of LUT has a much bigger influence in case of LUT_Y than in case of LUT_XY. This is no surprise, since when LUT_Y is used a certain amount of slack proportional with the distance between the LUT entries is not exploited. For a LUT size of 8 kB, LUT_XY produces energy savings which can be 40% better than those produced with LUT_Y.

The efficiency of the multiprocessor quasi-static algorithm is investigated during the next set of experiments. We assumed an architecture composed of three processors, 50 tasks, and a total LUT size of 24 kB. The results are summarized in Fig. 13 and show the deviation from the hypothetical limit, considering several ratios from ANC to WNC. We can see that the trend does not change, compared to the single-processor case. For example, for a ratio of 0.5 (the tasks execute half the worst case), the quasi-static is 22% away from the hypothetical. At the same ratio, for the single-processor case, the quasi-static approach was at 15% from the hypothetical algorithm. In contrast to a single-processor system, in the multiprocessor case, there are tasks that are executed in parallel, potentially resulting in certain amount of slack that cannot be used by the quasi-static algorithm.

In addition to the above benchmark results, we have conducted experiments on two real-life applications: an MPEG2 encoder and an MPEG2 decoder. The MPEG encoder consists of 25 tasks and is considered to run on a MPC750 processor. Table III 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 number of clock cycles to be executed in each mode is calculated using the offline algorithm from [6]. The algorithm assumes that each task executes its WNC. Applying the results of this offline calculation online leads to an energy reduction of 15%. Lines 3 and 4 show the improvements produced using the greedy online techniques proposed in [8] and [9], which achieve reductions of 67% and 69%, respectively. The next row presents the results obtained by the continuous quasi-static algorithm that improves over the nominal consumption by 78%.

TABLE III
OPTIMIZATION RESULTS FOR THE MPEG ENCODER ALGORITHM

Approach	E(μJ)	Reduc. (%)
Nominal	1.63	–
Static VS [7]	1.39	15
Greedy [9]	0.55	67
Task Splitting [10]	0.52	69
Quasi-static (cont.)	0.36	78

TABLE IV
OPTIMIZATION RESULTS FOR THE MPEG DECODER ALGORITHM

Approach	E(μJ)	Reduc. (%)
Nominal	0.601	–
Static VS [7]	0.561	7
Quasi-static LUT_Y	0.230	61
Quasi-static LUT_XY	0.208	66

The second real-life application, an MPEG2 decoder, consists of 34 tasks and is considered to run on a platform with 3 ARM7 processors. Each processor has four execution modes. Details regarding the hardware platform can be found in [32] and [33], while the application is described in [34]. Table IV shows the energy savings obtained by several approaches. The first line in the table shows the energy consumed when all the tasks are running at the highest frequency. The second line gives the energy that is obtained when static voltage scaling is performed.

The following lines present the results obtained with the approach proposed in Section VI, applied to a multiprocessor system, as shown in Section VIII. The third line gives the energy savings achieved by the straightforward discrete voltage scaling alternative proposed in Section VI-B1. The fourth line gives the energy obtained by efficient alternative proposed in Section VI-B2. A 16-kB memory was considered for storing the task LUTs. We notice that both alternatives provide much better energy savings than the static approach. Furthermore, LUT_XY is able to produces the best energy savings.

X. CONCLUSION

In this paper, we have introduced a novel QSVS technique for time-constraint applications. The method avoids an unnecessarily high overhead by precomputing possible voltage scaling scenarios and storing the outcome in LUTs. 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.

REFERENCES

[1] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. Int. Symp. Low Power Electron. Design*, 1998, pp. 197–202.

[2] 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. IEEE/ACM Int. Conf. Computer-Aided Design*, 2002, pp. 721–725.

[3] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *IEEE Foundations Comput. Sci.*, pp. 374–380, 1995.

[4] C. Kim and K. Roy, "Dynamic VTH scaling scheme for active leakage power reduction," in *Proc. Design Autom. Test Eur. Conf.*, Mar. 2002, pp. 163–167.
[5] L. Yan, J. Luo, and N. Jha, "Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2003, pp. 30–38.
[6] A. Andrei, P. Eles, Z. Peng, M. Schmitz, and B. M. Al-Hashimi, "Energy optimization of multiprocessor systems on chip by voltage selection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 3, pp. 262–275, Mar. 2007.
[7] M. P. I. Hong and M. B. Srivastava, "On-line scheduling of hard real-time tasks on variable voltage processors," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1998, pp. 653–656.
[8] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. Real Time Syst. Symp.*, 2001, pp. 95–105.
[9] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, Oct. 2004, pp. 84–93.
[10] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," *Real-Time Syst.*, vol. 31, no. 1-3, pp. 33–63, Dec. 2005.
[11] W. Y. R. Ernst, "Embedded program timing analysis based on path clustering and architecture classification," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 598–604.
[12] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2001, pp. 46–51.
[13] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M. R. Stan, "Optimal procrastinating voltage scheduling for hard real-time systems," in *Proc. Design Autom. Conf.*, 2005, pp. 905–908.
[14] J. R. Lorch and A. J. Smith, "Pace: A new approach to dynamic voltage scaling," *IEEE Trans. Comput.*, vol. 53, no. 7, pp. 856–869, Jul. 2004.
[15] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron, "Procrastinating voltage scheduling with discrete frequency sets," in *Proc. Design Autom. Test Eur. Conf.*, 2006, pp. 456–461.
[16] P. Yang and F. Catthoor, "Pareto-optimization-based run-time task scheduling for embedded systems," in *Proc. Int. Conf. Hardware-Software Codesign Syst. Synthesis*, 2003, pp. 120–125.
[17] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design Test Comput.*, vol. 18, no. 2, pp. 20–30, Mar.–Apr. 2001.
[18] J. Seo, T. Kim, and K.-S. Chung, "Profile-based optimal intra-task voltage scheduling for real-time applications," in *Proc. IEEE Design Autom. Conf.*, Jun. 2004, pp. 87–92.
[19] J. Seo, T. Kim, and N. D. Dutt, "Optimal integration of inter-task and intra-task dynamic voltage scaling techniques for hard real-time applications," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2005, pp. 450–455.
[20] J. Seo, T. Kim, and J. Lee, "Optimal intratask dynamic voltage-scaling technique and its practical extensions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 1, pp. 47–57, Jan. 2006.
[21] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2001, pp. 84–94.
[22] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi, "Quasi-static voltage scaling for energy minimization with time constraints," in *Proc. Design Autom. Test Eur. Conf.*, Nov. 2005, pp. 514–519.
[23] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Norwell, MA: Kluwer, 1995.
[24] M. Schmitz and B. M. Al-Hashimi, "Considering power variations of DVS processing elements for energy minimisation in distributed systems," in *Proc. Int. Symp. Syst. Synthesis*, Oct. 2001, pp. 250–255.
[25] W. Qin and S. Malik, "Flexible and formal modeling of microprocessors with application to retargetable simulation," in *Proc. Design Autom. Test Eur. Conf.*, Mar. 2003, pp. 556–561.
[26] D. G. Perez, G. Mouchard, and O. Temam, "Microlib: A case for the quantitative comparison of micro-architecture mechanisms," in *Proc. Int. Symp. Microarchitect.*, 2004, pp. 43–54.
[27] F. Gruian, "Energy-centric scheduling for real-time systems," Ph.D. dissertation, Dept. Comput. Sci., Lund Univ., Lund, Sweden, 2002.
[28] Y. Zhang, X. Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. IEEE Design Autom. Conf.*, Jun. 2002, pp. 183–188.
[29] A. Andrei, "Energy efficient and predictable design of real-time embedded systems," Ph.D. dissertation, Dept. Comput. Inf. Sci., Linkoping Univ., Linkoping, Sweden, 2007, SBN 978-91-85831-06-7.
[30] S. Hsu, A. Alvandpour, 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," *J. Solid State Circuits*, vol. 38, no. 5, pp. 755–761, May 2003.
[31] A. Macii, E. Macii, and M. Poncino, "Improving the efficiency of memory partitioning by address clustering," in *Proc. Design Autom. Test Eur. Conf.*, Mar. 2003, pp. 18–22.
[32] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Systemc cosimulation and emulation of multiprocessor SOC designs," *Computer*, vol. 36, no. 4, pp. 53–59, 2003.
[33] M. Loghi, M. Poncino, and L. Benini, "Cycle-accurate power analysis for multiprocessor systems-on-a-chip," in *Proc. ACM Great Lakes Symp.Very Large Scale Integr. (VLSI)*, New York, NY, 2004, pp. 410–406.
[34] J. Ogniewski, "Development and optimization of an MPEG2 video decoder on a multiprocessor embedded platform," M.S. thesis, Linkoping Univ., Linkoping, Sweden, 2007, LITH-IDA/DS-EX-07/006-SE.

Alexandru Andrei received the M.S. degree in computer engineering from Politehnica University, Timisoara, Romania, in 2001 and the Ph.D. degree from Linköping University, Linköping, Sweden, in 2007.

He is currently affiliated with Ericsson, Linköping, Sweden. His research interests include low-power design, real-time systems, and hardware–software codesign.

Petru Eles (M'99) received the Ph.D. degree in computer science from the Politehnica University of Bucharest, Bucharest, Romania, in 1993.

Currently, he is a Professor at the Department of Computer and Information Science, Linköping University, Linköping, Sweden. His research interests include embedded systems design, hardware–software codesign, real-time systems, system specification and testing, and computer-aided design (CAD) for digital systems. He has published extensively in these areas and coauthored several books.

Olivera Jovanovic received the diploma in computer science from the Technical University of Dortmund, Dortmund, Germany, in 2007, where she is currently working towards the Ph.D. degree in the Embedded Systems Group.

Her research interests lie in the development of low-power software for MPSoCs.

Marcus T. Schmitz received the diploma degree in electrical engineering from the University of Applied Science Koblenz, Germany, in 1999 and the Ph.D. degree in electronics from the University of Southampton, Southampton, U.K., in 2003.

He joined Robert Bosch GmbH, Stuttgart, Germany, where he is currently involved in the design of electronic engine control units. His research interests include system-level codesign, application-driven design methodologies, energy-efficient system design, and reconfigurable architectures.

Jens Ogniewski received the M.S. degree in electrical engineering from Linköping University, Linköping, Sweden, in 2007, where he is currently working towards the Ph.D. degree in parallel systems for video applications.

He worked on security systems for mobile phones at Ericsson, Lund, Sweden.

Zebo Peng (M'91–SM'02) received the Ph.D. degree in computer science from Linköping University, Linköping, Sweden, in 1987.

Currently, he is Professor of Computer Systems and Director of the Embedded Systems Laboratory at Linköping University. His research interests include design and test of embedded systems, design for testability, hardware–software codesign, and real-time systems. He has published over 200 technical papers and coauthored several books in these areas.