Workload-Ahead-Driven Online Energy Minimization Techniques for Battery-Powered Embedded Systems with Time-Constraints

***Abstract*— This paper proposes a new online voltage scaling (VS) technique for battery-powered embedded systems with real-time constraints. The VS technique takes into account the tasks execution times and discharge currents to further reduce the battery charge consumption when compared to the recently reported slack forwarding technique [12], whilst maintaining low online complexity of $O(1)$. Furthermore, we investigate the impact of online rescheduling and remapping on the battery charge consumption for tasks with data dependency which has not been explicitly addressed in the literature and propose a novel rescheduling/remapping technique. We demonstrate and compare the efficiency of the presented techniques using seven real-life benchmarks and numerous automatically generated examples.**

I. Introduction and Previous Work

Dynamic voltage scaling (DVS) is a powerful technique to reduce the energy consumption in embedded computing systems. This reduction is achieved by exploiting the application's temporal performance requirement by dynamically adapting the processing speed and the supply voltage of processing elements. Much research effort has concentrated on DVS algorithms for the calculation of appropriate performance/voltage setting. DVS algorithms can be broadly classified into *offline* and *online* (e.g. [8], [9], [10]) techniques depending on when the voltage settings are computed.

Offline (e.g. [1], [2], [3]) approaches calculate voltage settings at design time before actual execution based on worst case execution times (WCET) to guarantee satisfaction of time constraints. Although offline DVS avoids a run-time overhead due to the voltage calculation, it fails to exploit online slack arising from tasks executing with less than their WCET (differences $>$10 times have been reported [7]). On the contrary, online DVS techniques (e.g. [8], [9], [10]) perform the voltage calculation during run-time to utilize such online slack by taking into account the actual execution times (AET) of tasks. Clearly, online techniques have the potential to achieve higher energy savings, however, it is necessary to carefully design such online DVS algorithms in order to avoid high run-time overheads that could jeopardize the achievable energy savings and the timing constraints. An aggressive online voltage adjustment approach has been presented in [8]. Here only the next task to be executed is considered during the voltage calculation. The online approach introduced in [9] calculates the scaling factor for soft aperiodic tasks and considers run-time variations. The algorithm has an online complexity of $O(m^2)$, where m is the number of tasks which have not been executed on the processing elements. Zhu and Muller [10] utilize a feedback control loop to facilitate DVS and integrated the controller into an earliest deadline first (EDF) scheduler. The online complexity depends linearly, $O(m)$, on the number of tasks m. Task scheduling and online voltage scaling are combined in [11]. This work, however, is limited to identical PE systems and a straightforward extension toward heterogeneous systems is not apparent.

Although the above given offline and online voltage scaling techniques are effective in reducing energy dissipation, they are not efficient in prolonging the battery lifetime of mobile applications, since the non-linear battery characteristics [5], [6] are neglected during the optimization. In [4] an offline DVS technique for battery-powered systems was introduced, and it was demonstrated that up to 56% longer battery lifetimes could be achieved by taking into account the non-linear battery behavior during voltage calculation. More recently the first online and battery-aware DVS technique has been presented in [12]. This technique specifically targets periodic, independent tasks and assumes identical discharge currents for each task. According to this assumption, it is always better to exploit the available slack by the last task in the schedule [12]. Based on this, the authors introduce a slack forwarding technique that delays the utilization of online slack as late as possible. However, for many realistic multiprocessor systems executing heterogeneous tasks, this assumption limits the achievable savings in battery charge consumption.

This paper makes the following contributions: (a) We introduce a *workload-ahead-driven online DVS technique* which explicitly takes into account the workload-ahead (the sum over all products of discharge current and WCET of remaining tasks) to overcome the limitation of [12] discussed above. The proposed algorithm achieves longer battery lifetimes compared to slack forwarding algorithm without sacrificing the online time complexity, which remains constant, i.e. $O(1)$, since the workload used in the algorithm is computed in the offline phase. (b) In addition to the online voltage scaling, we address for the first time the problem of online task *rescheduling* and *remapping* for tasks with dependencies to further reduce the battery charge consumption. The proposed online rescheduling/remapping algorithm facilitates the usage of the workload-ahead-driven DVS technique and also has a complexity of $O(1)$.

The rest of the paper is organized as follows. Section II outlines the system and battery models. Section III presents the problem formulation. The proposed workload-ahead-driven online DVS technique is introduced in Section IV. Section V describes the proposed online rescheduling and remapping approach. Experimental results and conclusions are given in Sections VI and Section VII, respectively.

II. Preliminaries

A. System Model and Task Graph

In this work we consider battery-powered embedded computing systems, which consist of multiple processing elements (PEs) connected by communication links (CLs), illustrated in Fig. 1(a). A dc/dc converter adapts the battery voltage to the system supply voltage. The system functionality is captured by a task graph model $G(\mathcal{T}, \mathcal{C})$, Fig. 1(b). Nodes ($\tau_i \in \mathcal{T}$) in this directed acyclic graph (DAG) represent computational tasks and edges ($\gamma_j \in \mathcal{C}$) denote data communications between tasks. As shown in Fig. 1(b), tasks/edges are associated with worst case execution times (WCETs). The WCETs depend on the worst case number of cycles (K_w) required for execution and the circuit frequency f, which in turn depends on the supply voltage V_{dd} and threshold voltage V_t [1]:

$$t = \frac{K_w}{f} = \frac{K_w V_{dd}}{k(V_{dd} - V_t)^{\alpha}} \tag{1}$$

where k and α are technology related constants. The power dissipation of a task can be expressed as [2]:

$$P = f C_e V_{dd}^2 \tag{2}$$

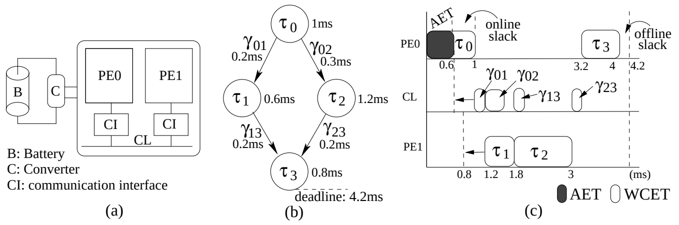

Fig. 1. Task graph and system model

where C_e is the effective switched capacitance of the circuit. Eqs. (1) and (2) provide the well-known energy/delay tradeoff exploited by all DVS approaches. Since the discharge current drawn from the batteries follows $I = P/(V_b \cdot \eta)$ (where V_b and η are the average battery voltage and the converter efficiency respectively), DVS can be used to influence the battery discharge current and, as a results, it can be used to achieve savings in the battery charge consumption [5], [6]. We assume that tasks and edges have been initially (offline) mapped and scheduled onto the target architecture, such that resource and time constraints are satisfied under WCETs, Fig. 1(c). At run-time, however, tasks might finish before their WCET, resulting in online slack. For instance, in Fig. 1(c) τ_0 has an actual execution time (AET) of 0.6ms, leaving an online slack of $0.4ms$.

B. Battery Model

To extend the battery lifetime of portable devices it is not sufficient to "simply" minimize the dissipated energy. But it is essential to take into account the non-linear battery behavior (caused by electro-chemical phenomena) during the system optimization [4], [5]. In general, it is better to discharge batteries with a low, constant current rather than with high current peaks. For an excellent survey on battery modelling see [13]. In this work we use an analytical high-level battery model proposed in [5] whose accuracy has been demonstrated to be within 3% of the physical battery. The battery charge consumption (reflecting the battery lifetimes) is:

$$\sum_{k=0}^{N-1} I_k \cdot F(L, st_k, st_k + \Delta_k, \beta) \tag{3}$$

where N is the total number of steps used to approximate the load current profile (LCP), and I_k, Δ_k and st_k denote the current, the duration and the start time of $step_k$ in the LCP, respectively. Further, L is the time duration that the battery has been charged for and β is a constant related to the non-linear property modelled by function F:

$$F(x, y, z, \beta) = z - y + 2\sum_{m=1}^{10} \frac{e^{-\beta^2 m^2 (x-z)} - e^{-\beta^2 m^2 (x-y)}}{\beta^2 m^2} \tag{4}$$

As smaller charge consumption (Eq. 3) will lead to longer battery lifetime [5], our optimization objective is the minimization of the charge consumption.

III. Problem Formulation

We assume that the tasks $\mathcal{T} = \{\tau_i\}$ and precedence constraints $\mathcal{C} = \{\gamma_j\}$ of task graph $G(\mathcal{T}, \mathcal{C})$ have been initially mapped and scheduled onto a distributed architecture containing voltage scalable processors, which can vary their supply voltage V_{dd} within a continuous range $[V_{min},\ V_{max}]$. The worst case clock cycles (K_w) that each task needs to be executed as well as its discharge current are known. In addition, some tasks may be associated by a deadline dl.

The problem addressed by the proposed online technique is twofold. Firstly, each time when a task τ_{next} is to be executed on a voltage scalable processor, an appropriate voltage V_{next} for its execution has to be selected such that the battery charge consumption is minimized (taking into account the workload-ahead) and all imposed deadlines can be guaranteed. This step is essential to exploit online slack that arises from variations in the execution time of tasks.

Secondly, for the initially (statically) given mapping and scheduling, some online slack could be potentially wasted, as demonstrated in the motivational example of Section V. To avoid this waste, the initial mapping and scheduling should be adapted in accordance to the available online slack, i.e. online rescheduling and remapping should be performed. The online voltage scaling problem is addressed in the next section, while rescheduling and remapping are the subjects of Section V.

IV. Battery-Aware Online Voltage Scaling

A. Motivational Example

The essence of the online voltage scaling problem is the online slack distribution, in order to efficiently exploit slack resulting from tasks that execute faster than their WCETs. In this motivational example we outline two different slack distribution methods using a realistic task graph from the E3S suite [16], namely the office-auto benchmark consisting of 5 tasks, Fig. 2(a). For simplicity we consider

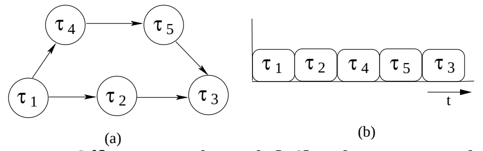

Fig. 2. Office-auto task graph [16] and execution order

here that all tasks have been mapped to a single processing element and the execution order corresponds to Fig. 2(b). We assume that the PE can vary its supply voltage between V_{min} and V_{max}, with $V_{min} = 0.4 \cdot V_{max}$. In accordance, the task execution times follow Eq. (1). Table I gives the worst-case execution time (WCET) and discharge current (I) of each task (in execution order of Fig. 2(b)), when executing at V_{max}. Furthermore, the table shows the actual

	τ_1	τ_2	τ_4	τ_5	τ_3
WCET (ms)	0.79	10.80	4.80	22.81	0.79
I (mA)	0.256	4.066	3.990	2.243	0.256
AET (ms)	0.63	8.64	3.84	18.25	0.63
online slack (ms)	0.16	2.16	0.96	4.56	0.16

TABLE I
WCETs, DISCHARGE CURRENTS, AETs AND ONLINE SLACKS OF AUTO-OFFICE TASKS

execution time (AET) of tasks at run-time (we assume here 80% of WCET), as well as the resulting online slack (WCET-AET). The deadline is assumed to correspond to the finishing time of the last task (τ_3), when all tasks execute with their WCET.

Table II shows the outcome of two different techniques that distribute the available online slack. Note that not all the available online slack might be exploited due to the limited voltage range of the PE. The first technique is based on the slack forwarding idea presented

	"slack forwarding"		proposed technique	
	Online slack (ms)			
	available	exploits	available	exploits
τ_2	0.16	0	0.16	0.04
τ_4	2.32	0	2.28	0.38
τ_5	3.28	0	2.86	2.85
τ_3	7.84	1.18	4.57	1.18

TABLE II
ONLINE SLACK DISTRIBUTION

in [12], in which all available online slack is forwarded to the last task. Accordingly, task τ_3 accumulates an online slack of 7.84ms (0.16+2.16+0.96+4.56) before it starts execution. Nevertheless, due to the limited voltage range of the PE, it is only possible to make use of 1.18ms of the total slack, i.e., 6.66ms of slack remain unexploited.

As a result, a battery charge of 0.189mAs can be calculated from Eqs. (1)–(4) and the task properties given in Table I.

A second approach (the approach we propose in this paper) distributes the available online slack by explicitly considering the discharge currents and WCETs of tasks. That is, each time a task finishes execution, the workload-ahead (sum over products of discharge current and WCET of remaining tasks) is evaluated to make a slack distribution decision. The method is outlined in Section IV-B, however, the resulting slack distribution is given in Table II. As we can observe from the table, using this method all tasks are assigned some of the available slack. For instance, after task τ_1 has finished execution the available online slack that is exploitable by task τ_2 is 0.16ms. However, it exploits only 0.04ms of this slack via voltage scaling, while the remaining 0.12ms are accumulated for the workload ahead. Therefore, after τ_2 finishes the available slack is 2.28ms (2.16+0.12). As shown in Table II, task τ_4 exploits 0.38ms of this slack. Similarly, the slack is forwarded and distributed to the tasks τ_5 and τ_3. When τ_3 is to be executed, the available online slack (4.57ms) is still sufficient to scale its voltage to the lowest level, i.e. τ_3 obtains the same amount of slack then with the slack forwarding approach. According to the second distribution, the consumed battery charge is reduced to 0.154mAs, which is an improvement of 18.5% when compared to the slack forwarding method [12].

B. Workload-Ahead-Driven Online DVS Technique

As we have seen in the motivational example of Section IV-A, slack forwarding is not particularly effective for heterogeneous tasks which draw different currents from the battery and require different WCETs. An effective online DVS algorithm must take these aspects into consideration to achieve a "globally" fair distribution of online slack. To cope with this problem, we define two metrics that capture the effects of tasks on the battery charge consumption. Assume that the next task to be scaled and the set of unscaled (ahead) tasks are denoted as τ_{next} and $\mathcal{T}_r$, respectively, with $\tau_{next} \in \mathcal{T}_r$.

Definition 1: We define the *workload* (W_i) of a task τ_i as the product of its discharge current I_i and WCET$_i$, i.e. $W_i = I_i \cdot WCET_i$.

Definition 2: We define the *workload-ahead* (WA_i) of a task τ_i as the sum of the workloads of all remaining tasks in $\mathcal{T}_r$, i.e. $WA_i = \sum_{\tau_j \in \mathcal{T}_r} W_j$.

The workload-ahead-driven slack distribution gives the slack to the next task based on the ratio of its W and WA:

$$slack_{next} = \frac{W_{next}}{WA_{next}} \cdot os \tag{5}$$

where os is the available online slack. It should be noted that both W and WA for each task are computed in the offline phase, so this computation does not contribute to the online complexity of the algorithm. It is also important to note that it is our aim to develop an effective yet fast online DVS technique, hence we intentionally avoiding a complex online algorithm.

The workload as given in Definition 1 is the main source of battery charge consumption, i.e. larger workloads will consume more battery charge [5]. Hence, by using Eq. (5), tasks with heavy workload are scaled more aggressively than light weight tasks. Although we use here the WCETs to compute the workloads, it is possible to leverage information regarding expected execution times (EETs), if such information is available. In the case that EETs are known in advance, these values should be used rather than WCET in order to compute the workload-ahead more accurately. Another important factor affecting the battery charge consumption is the position of a task in the schedule (the later a task is in the schedule, the smaller should be the current it draws [6]). It is apparent that this factor is also taken into account by Eq. (5): the later a task is in the schedule, the smaller its WA, and as a result, it will receive relatively larger slack and its current will be smaller as expected. For example, from Tables I and II we can observe that task τ_5 has the largest workload (22.81ms·2.243mA) and its position is close to the end of the schedule and as a result, it obtains the largest slack portion (2.85ms). Note that we only calculate the slack distributed to the next task. It is not necessary to distribute slack to tasks beyond the next task because the total amount of online slack will change with the execution of the next task, hence, a recomputation of the distribution is required.

```
Algorithm: WAD-DVS
Input:  - $W_{next}$, $WA_{next}$, $K_w$, $WCET_{next}$, $ST_{next}$, $CurrTime$
Output: - $V_{next}$
01: if the next task can start immediately then
02:     online_slack = $ST_{next}$ – $CurrTime$;
03:     $slack_{next}$ = online_slack × ($W_{next}$/$AW_{next}$);
04:     $frequency_{next}$ = $K_w$/($WCET_{next}$ + $slack_{next}$);
05:     Compute $V_{next}$ by solving Eq. (1)
        with known $V_t$ and $frequency_{next}$;
06:     return $V_{next}$;
07: else
08:     call RM-RS-DVS;         // (Fig. 6)
09: end if
```

Fig. 3. Pseudo code of the workload-ahead-driven online DVS

Based on above outlined workload-ahead principle, Fig. 3 gives the pseudo code of our workload-ahead-driven voltage scaling algorithm. Its input consists of the information regarding the next task. This information includes the task's workload (W_{next}) and workload-ahead (WA_{next}), its worst case number of cycles (K_w) and execution time ($WCET_{next}$), as well as its offline decided start time (ST_{next}). In addition, the algorithm requires the current time ($CurrTime$) in the schedule. When a busy PE finishes executing a task or an idle PE receives an incoming data communication, it calls the online voltage scaling algorithm. If the next task on the PE can start immediately (line 1), the available online slack is computed from the current time and the start time of the next task τ_{next} (line 2). The slack distributed to τ_{next} is calculated based on Eq. (5) in line 3. According to the amount of distributed slack, the frequency and voltage at which τ_{next} has to be executed are computed in lines 4 and 5. Then the algorithm returns V_{next} and terminates in line 6. On the other hand, if the next task could not start at this moment due to the lack of needed input data (e.g. τ_3 in Fig. 1 (c) can not start when τ_0 finishes since γ_{23} has not arrived yet), the algorithm calls the online task rescheduling/remapping procedure described in Section V (line 8). It is important to note that each step in the algorithm can be performed in constant time ($O(1)$), hence the overall complexity is $O(1)$. The constant complexity allows the scaling overhead be incorporated into the WCET of tasks during timg analysis [14]. In the above described online voltage scaling algorithm, no task will start later than its offline decided start time, so the timing constraint of each task is guaranteed and all hard deadlines are satisfied.

V. Online Task Rescheduling and Remapping

Due to the initial static schedule and mapping, it is possible that some of the online slack is wasted when tasks execute faster than their WCET. The reason for this is the fact that earlier finishing tasks might result in other tasks becoming ready for execution earlier, however, the static schedule "unnecessarily" delays such tasks. To avoid this waste of online slack, we introduce online task rescheduling and remapping as supplements of the proposed online DVS (Section IV-B). Fig. 4 outlines the integration of the workload-ahead-driven DVS technique with the rescheduling and remapping strategy. The necessity for online rescheduling and remapping is illustrated through a motivational example.

A. Motivational Example

Fig. 5(a) shows a task graph consisting of 7 nodes. The WCETs of tasks are indicated, and the tasks are mapped and scheduled on 3 PEs, in accordance to Fig. 5(b). For simplicity we neglect communications

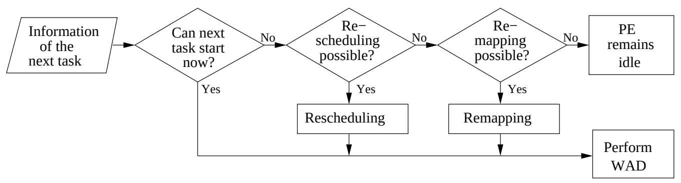

Fig. 4. Integrated workload-ahead-driven DVS and rescheduling/remapping

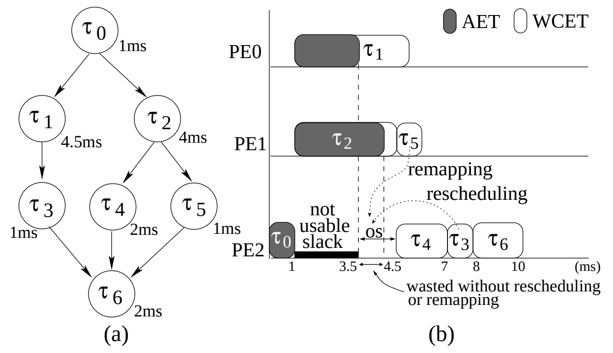

Fig. 5. Illustration of online task rescheduling and remapping

in this example, however, they are considered in our algorithm. As we can observe from Fig. 5(b), τ_1 has a longer WCET (4.5ms) than τ_2 (4ms). However, let us assume that τ_1 requires only 2.5ms for execution at run-time, i.e. it finishes at 3.5ms. When τ_1 finishes, there is an online slack appearing on PE2 (indicated as *os* in Fig. 5(b)), but τ_4 can not start its execution earlier because its parent task τ_2 has not terminated at this moment. Clearly, a large portion of the online slack on PE2 is wasted. To avoid this waste, task τ_3 on PE2 can be placed before τ_4 to fill the available online slack. That is, we change the execution order of the remaining tasks (rescheduling). However, the WCET of the rescheduled task must be smaller than the available online slack to avoid the delay of the start time of the remaining tasks, which could result in potential deadline violations. For example, to be rescheduled, the WCET of τ_3 must be smaller than *os*. If the WCET of τ_3 is longer than the slack, then we can further search the remaining tasks of other PEs to see if there is suitable task. In the example of Fig. 5(b), τ_5 on PE1 can be fetched from PE1 to fill the online slack on PE2, i.e., τ_5 is remapped online.

B. *Online Task Rescheduling and Remapping Technique*

Our aim is to facilitate online voltage scaling to avoid online slack waste. Similar to our online voltage scaling, we want the online rescheduling and remapping techniques to be independent of the number of tasks, in order to minimize its computational overhead. Therefore, we will not take all the remaining tasks into consideration, instead, an effective yet fast local search strategy is proposed. The pseudo code of our online task rescheduling/remapping is given in Fig. 6. Suppose there are n PEs in the system and the pth ($1 \leq p \leq n$) PE is the one with the potentially wasted online slack. Let each PE have an exe_Queue storing tasks to be executed and let M be a constant integer called the *search window*. The search window represents the maximal number of tasks in exe_Queue that are potentially to be rescheduled or remapped. The complexity of the algorithm is bounded by the search window, which is constant. The algorithm first restricts the search window if the number of tasks in exe_Queue is smaller than M (line 1), then searches exe_Queue of PE[p] within the search window M to fill the online slack on PE[p]. To be a rescheduling candiate, a task should satisfy two conditions (line 4). First, its WCET (we still only know WCET of remaining tasks at this moment) is less than the online slack so that the next task will start no later than its offline decided start time. For example, in Fig. 5, the WCET of τ_3 is less than the online slack and τ_4 is

```
Algorithm: RM-RS-DVS
Input:  - $W_{next}$, $WA_{next}$, $WCET_{next}$, $ST_{next}$, $K_w$,
          $CurrTime$, $M$
Output: - $V_{next}$
01: int $k = min(M,\ size\ of\ PE[p].exe\_Queue)$;
02: bool $search\_result = false$;
03: for $j = 1$ to $k$  //online rescheduling
04:   if $PE[p].exe\_Queue[j]\ satisfies$
          $rescheduling\ conditions$ then
05:       $search\_result = true$;
06:       $move\ exe\_Queue[j]\ to\ the\ head\ of\ exe\_Queue$;
07:       $break$;
08:   end if
09: end for
10: if $search\_result == true$ then
11:   call WAD-DVS;    // (Fig. 3)
12: else // online remapping
13:   for $i = 1$ to $n$ && $i\ != p$
14:     $k = min(M,\ size\ of\ PE[i].exe\_Queue)$;
15:     for $j = 1$ to $k$
16:       if $PE[i].exe\_Queue[j]\ satisfies$
              $remapping\ conditions$ then
17:           $search\_result = true$;
18:           $fetch\ task\ PE[i].exe\_Queue[j]\ from\ PE[i]\ and$
              $put\ it\ to\ the\ head\ of\ PE[p].exe\_Queue$;
19:           $break$;
20:       end if
21:     end for
22:   end for
23:   if $search\_result == true$ then
24:     call WAD-DVS;    // (Fig. 3)
25:   else
26:     let $PE[p]$ be idle;
27:   end if
28: end if
```

Fig. 6. Pseudo code of online rescheduling and remapping

guaranteed to start on time. This condition prevents any deadline violation. The second condition is that at the time of the search, all its incoming data communications have arrived so that it can start at this moment. If these two conditions are true, the found task is moved to the head of the execution queue and placed before the next task τ_{next} (line 6). Then the proposed online DVS procedure is called (line 11) to utilize the otherwise wasted online slack. As indicated in Fig. 4, if no suitable task for rescheduling has been found, task remapping will be performed (line 12-28).

Similar to online rescheduling, online remapping checks tasks in the search window of exe_Queue of other PEs to find a task that can utilize the available online slack (line 12-22). Nevertheless, the selection is more strict in remapping phase (line 16). Tasks can only be fetched from another PE if they fulfill the two conditions mentioned in the rescheduling phase as well as if their remapping does not introduce new communications. The reason is that new data communications may delay the transfer of some other scheduled communications on the CLs. This, in turn, may cause some tasks not to start on time and result in the risk of deadlines violation. After a task is remapped it is removed from the task queue of its originally mapped PE to exe_Queue head of PE[p] (line 18), which then will call the proposed voltage scaling procedure (line 24). If no task can be found to be remapped, the idling of PE[p] is not avoided and the online slack is wasted (line 26).

VI. **Experimental Results**

In order to validate the effectiveness of the proposed online voltage scaling and rescheduling/remapping strategies in reducing battery charge consumption, we conducted several experiments using 30 hypothetical examples as well as 7 real-world benchmakrs. The hypothetical examples have been automatically generated using TGFF [17], a pseudo-random task graph generator. The first 5 realistic examples have been taken from the E3S benchmark suit [16] (auto-indust, consumer, office-auto, networking and telecomm), while the task graphs for GSM decoder and encoder have been derived from publicly available C code [18]. All reported results have been

Bench-mark (# task)	battery charge consump. (mAs)				Improvem. (%)	
	ASU $O(n^2X)$	SF $O(1)$	ACD $O(1)$	WAD $O(1)$	$\frac{WAD}{SF}$	$\frac{WAD}{ACD}$
Auto-in. (28)	0.248	0.372	0.372	0.247	33.67	33.67
Consum. (27)	4.687	7.136	7.136	4.690	34.28	34.28
Office-au. (5)	0.104	0.126	0.118	0.104	17.86	11.86
Network. (23)	0.873	1.125	1.125	0.863	23.29	23.39
Telecom. (42)	0.289	0.387	0.387	0.287	25.69	25.69
GSMdec. (34)	4.394	6.785	6.595	4.365	35.66	33.81
GSMenc. (53)	5.877	9.044	8.419	5.755	36.36	31.65

TABLE III
RESULTS OF ONLINE DVS IN SINGLE PE SYSTEMS

obtained using the battery model of Section II-B and the evaluation criterion is the battery charge consumption. Further, the evaluation is based on the same normal distribution (mean: 0.6 times the WCET, standard deviation: 0.13 times the WCET) of the actucal execution times of tasks that has been used in [12].

In the first set of experiments, we evaluate the efficiency of our workload-ahead-driven DVS algorithm (WAD, Fig. 3) by means of a comparison with 4 different online DVS techniques, summarized for reference in the following: **1. ASU**: This heuristic is based on the slack distribution technique *AlterSlackUtilization* presented in [5]. Although this technique was originally proposed as offline technique, we have extended it towards online DVS by calculating voltage setting each time before a task starts execution. The satisfaction of deadlines is imposed by considering WCET for each executing task. It is important to note that this approach has a high computational run-time overhead (for each finishing task $O(n^2X)$, where n is the number of the remaining tasks and X reflects the complexity of the battery model). However, we use this approach due to its solution quality as a baseline for comparison. **2. SF**: The slack forwarding approach is based on the technique presented in [12]. Its time complexity is constant ($O(1)$). **3. ACD**: The average current-based distribution is a heuristic that leverages information regarding the task discharge currents to distribute slack: if the current of the next task to be executed is less than or equal to average current of tasks, it gets no slack; else, it gets some slack such that its current decreases to the average value. When there is only one task left, all slack is assigned to it. The time complexity of this method is constant, too. We use this heuristic to underline the importance of the workload-driven technique that considers discharge currents as well as remaining task execution times. **4. WAD**: This represents our workload-ahead-driven distribution technique, as introduced in Section IV-B. It has also a complexity of $O(1)$.

Since the slack forwarding idea [12] is most suitable for task sets without data communications, we executed the 7 realistic benchmarks on single PE systems[1], in which the inter-PE communications between tasks can be neglected. Table III gives the results in terms of battery charge consumption. In the table, the first column gives the benchmark name and the number of tasks in the benchmark. The results of the 4 online DVS techniques are given in Columns 2–5. In the last two columns we show the percentage of improvement in battery charge consumption using the proposed WAD method over methods SF and ACD. We can observe that ASU yields consistently the lowest battery consumption when compared to SF and ACD, while the results produced by our WAD technique are very close to ASU and in some cases even slightly better. Consider, for instance, the GSM decoder benchmark. Here ASU obtains a battery charge consumption of 4.394mAs, while SF and ACD result in 6.785mAs and 6.595mAs, respectively. Nevertheless, WAD achieves the lowest value with 4.365mAs, resulting in improvements of 35.66% and 33.81% over SF and ACD, respectively. This clearly indicates the high solution quality of the proposed approach at low computational complexity.

[1]Note that not all these benchmarks can be executed on a single PE without violating timing constraints. We therefore adjusted the deadlines such that no violation occurred under WCETs.

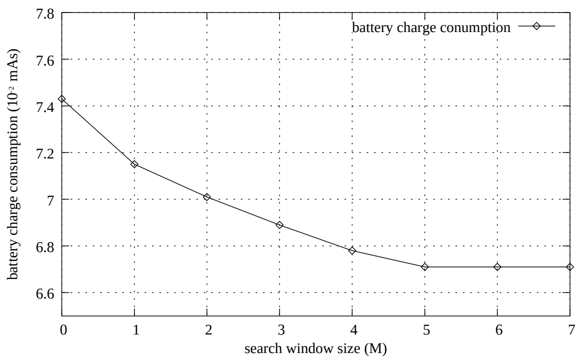

Fig. 7. Influence of search window size on rescheduling/remapping results

The second set of experiments was conducted to validate the workload-ahead-driven DVS as well as the rescheduling/remapping techniques in the context of systems consisting of multiple processing elements. We used LOPOCOS [15], an academic system-level synthesis tool, to find suitable multiple PE implementations and to generate the offline mappings and schedules for all 36 benchmarks (GSM decoder and encoder have been combined into a single benchmark). In all experiments we set the search window size (M) of the rescheduling/remapping algorithm to 10, empirically found to be a good value. Nevertheless, due to the importance of the window size on the solution quality we have devoted an extra set of experiments on this subject, presented later in this section. Since the slack forwarding technique [12] was particularly introduced for independent tasks, we refrain in these experiments from a direct comparison.

The results of our experiments are summarized in Table IV. The first, second, and third columns give the benchmark name, the number of tasks/communication edges, and the number of PEs in the system, respectively. Columns 4–7 show the battery charge consumptions in 4 different scenarios. Column 4 (No DVS) represents the nominal charge consumption, i.e., when no online voltage scaling is employed; Column 5 (WAD) shows the results of the proposed workload-ahead-driven DVS technique; Columns 6 (WAD+RS) and 7 (WAD+RS+RM) give the charge consumption when integrating WAD with online rescheduling and rescheduling with remapping, respectively. Columns 8–10 summarize the achieved battery charge savings in percent. Consider, for instance, benchmark tgff17. Here the nominal and the WAD-based charge consumptions are 5.941610×10^{-2}mAs and 4.4459×10^{-2}mAs, respectively, representing a saving of 25.17%. This can be further improved by using rescheduling as well as rescheduling with remapping to 4.1119×10^{-2}mAs and 4.0206×10^{-2}mAs, respectively, obtaining further saving of 7.78% and 9.83% when compared to using WAD only.

As mentioned above, the window size used by the rescheduling and remapping technique has an influence on the achievable savings in battery charge consumption as well as on the online complexity. The following experiment is used to clarify this aspect and to provide an insight into which window size should be typically used. Fig. 7 shows the battery charge consumption of benchmark tgff27 depending on the window size M. As it can be observed, a window size of zero, i.e. no rescheduling/remapping is performed, results in a charge consumption of 7.43×10^{-2}mAs. However, with an increasing window size this value decreases to 6.7×10^{-2}mAs. In general, we have observed that a window size of 10 provides a good trade-off between solution quality and complexity for all investigated benchmarks.

Bench-marks	# task/ edge	# PE	Battery charge consumption (10^{-2}mAs)				Percentages (%)		
			No DVS	WAD	WAD+RS	WAD+RS+RM	$\frac{NoDVS}{WAD}$	$\frac{WAD+RS}{WAD}$	$\frac{WAD+RS+RM}{WAD}$
Auto-ind.	28/25	2	22.134	17.317	17.317	17.317	21.76	0	0
Consum.	27/30	3	851.32	613.38	613.38	613.38	27.94	0	0
Office-au.	5/5	1	15.578	12.768	12.768	12.768	18.03	0	0
Network.	23/19	2	12.295	10.404	10.404	10.404	15.38	0	0
Telecom.	42/40	2	46.181	36.118	36.054	36.031	21.79	0.17	0.24
GSM	87/138	6	10.399	7.581	7.458	7.458	27.09	1.62	1.62
tgff1	14/17	2	0.0403	0.0351	0.0349	0.0349	12.97	0.51	0.51
tgff2	40/51	2	1.3795	0.9642	0.9334	0.9334	30.10	3.18	3.18
tgff3	45/49	2	1.7483	1.2703	1.2301	1.2060	27.34	3.17	5.06
tgff4	65/77	3	2.4312	1.8067	1.7682	1.7352	25.68	2.13	3.96
tgff5	36/50	2	1.3206	0.9508	0.9493	0.9039	27.99	0.16	4.94
tgff6	46/62	2	0.9061	0.6869	0.6805	0.6805	24.18	0.93	0.93
tgff7	97/114	2	1.9721	1.4557	1.3656	1.3398	26.18	6.18	7.95
tgff8	121/145	4	3.6957	2.6453	2.4891	2.4189	28.42	5.90	8.55
tgff9	94/115	4	1.8398	1.3622	1.2417	1.2276	25.95	8.84	9.88
tgff10	60/85	3	1.8431	1.3428	1.3007	1.2716	27.14	3.14	5.30
tgff11	51/67	3	1.9460	1.4395	1.3534	1.3511	26.03	5.97	6.14
tgff12	102/132	4	4.0853	2.9851	2.8718	2.8128	26.93	3.79	5.77
tgff13	71/91	2	1.5975	1.1866	1.1695	1.1687	25.72	1.44	1.51
tgff14	128/166	5	5.3594	3.9473	3.8053	3.7664	26.34	3.59	4.58
tgff15	145/187	4	2.5630	1.9432	1.8382	1.8221	24.18	5.40	6.23
tgff16	84/109	3	1.6536	1.2294	1.2060	1.1407	25.65	1.90	7.21
tgff17	196/236	3	5.9416	4.4459	4.1119	4.0206	25.17	7.78	9.83
tgff18	149/180	3	4.2885	3.2265	3.0185	2.9791	24.76	6.44	7.66
tgff19	81/103	3	2.4865	1.8633	1.7925	1.7306	25.06	3.81	7.12
tgff20	149/167	5	5.3703	4.0737	3.8586	3.8368	24.14	5.28	5.81
tgff21	70/94	2	1.0458	0.7840	0.7249	0.7248	25.02	7.54	7.55
tgff22	102/152	3	3.9872	2.8624	2.6778	2.6188	28.21	6.45	8.50
tgff23	117/170	3	3.9352	2.9009	2.7473	2.7429	26.28	5.29	5.44
tgff24	316/413	4	8.3134	6.0926	5.7592	5.6844	26.71	5.47	6.70
tgff25	269/348	3	5.4967	4.1520	3.8301	3.8114	24.46	7.75	8.20
tgff26	331/408	4	8.1994	6.0464	5.5650	5.4978	26.25	7.96	9.07
tgff27	280/341	4	10.097	7.4264	6.8689	6.7069	26.45	7.50	9.68
tgff28	378/443	4	1.3623	9.9426	9.1701	8.9537	27.01	7.77	9.94
tgff29	252/312	4	6.6971	5.0638	4.7497	4.6449	24.38	6.20	8.27
tgff30	210/234	3	0.3786	0.3183	0.2746	0.2735	15.92	13.71	14.08
						Ave. percents	24.80	4.36	5.59

TABLE IV
EXPERIMENTAL RESULTS IN MULTI-PE SYSTEMS

VII. Conclusion

In this paper, we have presented a workload-ahead-driven voltage scaling technique which explicitly takes the discharge current and execution times into account to make battery-aware scaling decisions. To further improve the battery charge consumption, we have presented an online rescheduling/remapping technique that aims to reduce the waste of online slack when using static schedules and mappings. To the best of our knowledge, this is the first online approach that addresses voltage scaling as well as rescheduling/remapping in conjunction. All presented techniques are of constant time complexity ($O(1)$), making them suitable for applications with hard real-time systems. The efficiency of the proposed techniques have been experimentally validated using automatically-generated as well as real-life benchmarks. It has been demonstrated that significant savings of up to 36% in the battery charge can be obtained when compared to approaches that delay the slack utilization as late as possible.

References

[1] J. Luo and N. K. Jha, ”Low Power Distributed Embedded Systems: Dynamic Voltage Scaling and Synthesis”, Proc. Int. Conf. High Performance Computing, 2002.

[2] A. Andrei, M. T. Schmitz, P. Eles, Z. Peng and B. M. Al-Hashimi, ”Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems”, Proc. DATE, 518-523, 2004.

[3] M. T. Schmitz and B. M. Al-Hashimi, ”Considering Power Variations of DVS Processing Elements for Energy Minimization in Distributed Systems”, Proc. ISSS, 250-255, 2001.

[4] J. Luo and N. K. Jha, ”Battery-aware Static Scheduling for Distributed Real-time Embedded Systems”, Proc. DAC, 444-449, 2001.

[5] D. Rakhmatov, S. Vrudhula, ”Energy Management for Battery-powered Embedded Systems”, ACM Transactions on Embedded Computing Systems, Vol. 2:3, 277-324, 2003.

[6] P. Chowdhury and C. Chakrabarti, ”Battery Aware Task Scheduling for a System-on-a-chip Using Voltage/Clock Scaling”, Proc. SIPS, 201-206, October 16-18, 2002.

[7] W. Ye and R. Ernst, ”Embedded program timing analysis based on path clustering and architecture classification”, Proc. ICCAD, 598-604, 1997.

[8] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, ”Dynamic and Aggressive Scheduling Techniques for Power-aware Real-time Systems”, Proc. RTSS, 95-105, 2001.

[9] J. Luo and N. K. Jha, ”Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems”, Proc. ASP-DAC, 712-719, Jan. 2002.

[10] Y. Zhu and F. Mueller, ”Feedback EDF scheduling exploiting dynamic voltage scaling”, Proc. RTAS, 84-89, 2004.

[11] D. Zhu, R. Melhem and B. R. Childers, ”Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems”, IEEE Trans. on Parallel and Distributed Systems, Vol. 14:7, 686-700, 2003.

[12] J. Ahmed and C. Chakrabarti, ”A dynamic task scheduling algorithm for battery powered DVS systems”, Proc. ISCAS, 813-816, 2004.

[13] R. Rao, S. Vrudhula and D. Rakhmatov, ”Battery modeling for energy-aware system design”, IEEE Computer, Vol. 36:12, 77-87, 2003.

[14] C. Shen, K. Ramamritham and J. A. Stankovic, ”Resource Reclaiming in Multiprocessor Real-time Systems”, IEEE Trans. on Parallel and Distributed Systems, Volume. 4:4, 382-397, 1993.

[15] M. T. Schmitz, B. M. Al-Hashimi and P. Eles, ”Synthesizing Energy-Efficient Embedded Systems with LOPOCOS”, Design Automation for Embedded Systems, Vol. 6, 401-424, 2002.

[16] http://www.ece.northwestern.edu/ dickrp/e3s/

[17] http://ziyang.ece.northwestern.edu/tgff/

[18] http://kbs.cs.tu-berlin.de/ jutta/toast.html