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

YUAN CAI University of Iowa
MARCUS T. SCHMITZ Robert Bosch GmbH
BASHIR M. AL-HASHIMI University of Southampton
SUDHAKAR M. REDDY University of Iowa

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 execution times and discharge currents of tasks to further reduce the battery charge consumption when compared to the recently reported slack forwarding technique [Ahmed and Chakrabarti 2004], 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. Finally, we take leakage power into consideration and extend the proposed online techniques to include adaptive body biasing (ABB) which is used to reduce the leakage power. We demonstrate and compare the efficiency of the presented techniques using seven real-life benchmarks and numerous automatically generated examples.

Categories and Subject Descriptors: J.6 [**Computer-aided engineering**]: Computer-aided design

General Terms: Design, Algorithms

Additional Key Words and Phrases: Dynamic voltage scaling, Embedded systems, Battery, Adaptive body biasing

1. INTRODUCTION AND PREVIOUS WORK

Dynamic voltage scaling (DVS) is a powerful technique to reduce the energy consumption in embedded computing systems. DVS algorithms can be broadly classified into *offline* and *online* techniques depending on when the voltage settings are computed. Offline (e.g. [Luo and Jha 2002a], [Andrei et al. 2004], [Schmitz and Al-

Authors' addresses: Yuan Cai and Sudhakar M. Reddy, Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242, email: {yucai, reddy}@engineering.uiowa.edu; Bashir M. Al-Hashimi, Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, Southampton, UK, email: bmah@ecs.soton.ac.uk; Marcus T. Schmitz, Robert Bosch GmbH, Stuttgart D-70442, Germany, email: marcus.schmitz@de.bosch.com;
This work is supported in part by the EPSRC, U.K., under grant GR/S95770

Hashimi 2001]) 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 to compute voltage settings, it fails to exploit online slack arising from tasks executing with less than their WCET (differences >10 times have been reported [Ye and Ernst 1997]). On the contrary, online DVS techniques (e.g. [Aydin et al. 2001], [Pillai and Shin 2001], [Kim et al. 2002], [Luo and Jha 2002b], [Zhu and Mueller 2004]) calculate voltage settings 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. Many online voltage adjustment approaches for independent tasks have been proposed ([Aydin et al. 2001], [Pillai and Shin 2001], [Kim et al. 2002]). These approaches depend on the schedulability check of the earliest deadline first (EDF) or the rate monotonic (RM) algorithm, which can not be applied to task graphs where there are dependent relationships among tasks. The online approach introduced in [Luo and Jha 2002b] calculates the scaling factor for soft aperiodic tasks and considers run-time variations. Zhu and Muller [Zhu and Mueller 2004] utilize a feedback control loop to facilitate DVS and integrated the controller into an earliest deadline first scheduler. Task scheduling and online voltage scaling are combined in [Zhu et al. 2003]. This work, however, is limited to identical processing element (PE) systems and a straightforward extension toward heterogeneous systems is not apparent. Shin and Kim [Shin and Kim 2001] give a path based intra-task DVS algorithm. The task is modeled as a conditional flow graph in which there is a worst case execution path (WCEP) and an average case execution path (ACEP). Their algorithm inserts voltage scaling points at branch or loop nodes to scale the voltage online based on the ACEP instead of the WCEP. The work in [Shin and Kim 2001] is orthogonal to the proposed algorithm which determines the voltage settings on task-by-task basis (i.e., inter-task voltage scaling). At inter-task level, tasks on every path of the task graph will be executed and there is no separation between WCEP and ACEP. However, for each task, possibility of a difference between its WCET and AET exists. It is this difference that the proposed algorithm utilizes.

Although the offline and online voltage scaling techniques discussed above are effective in reducing energy dissipation, they are not efficient in prolonging the battery lifetime of mobile applications, since the non-linear battery characteristics [Rakhmatov and Vrudhula 2003], [Chowdhury and Chakrabarti 2002] are neglected during the optimization. In [Luo and Jha 2001] 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 the calculation of voltage settings. Recently the first online and battery-aware DVS technique has been presented in [Ahmed and Chakrabarti 2004]. 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 [Ahmed and Chakrabarti 2004]. 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 [Ahmed and Chakrabarti 2004] 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) 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, which is not addressed in [Ahmed and Chakrabarti 2004]. The proposed online rescheduling/remapping algorithm facilitates the usage of the workload-ahead-driven DVS technique and also has a constant complexity. (c) We extend the power model to include the leakage power and adaptive body biasing (ABB) technique is utilized to reduce the leakage power. We believe that the leakage power issue has not been studied by the battery aware design procedures proposed earlier. A look-up-table (LUT) method is utilized to keep the complexity of the combined DVS and ABB process to be still $O(1)$.

Rao et al. [Rao et al. 2005] point out that in certain cases, energy-aware design ($\mathcal{E}$ policy) should be chosen over battery-aware design ($\mathcal{B}$ policy) to reduce the battery charge consumption. Specifically, when there is a long rest period in the task schedule or when the task execution times are on the order of ms, it is better to use $\mathcal{E}$ policy. The proposed workload-ahead-driven voltage scaling is suitable for both $\mathcal{B}$ and $\mathcal{E}$ policies. First, the workload of tasks is in fact the actual battery charge lost, which must be considered in both $\mathcal{B}$ policy and $\mathcal{E}$ policy, as shown in [Rao et al. 2005]. Second, it can be seen that in the proposed method, time unit is eliminated during the calculation of the slack distribution. Slack allocated to a task is a function of the ratio of its workload and the total workload of the tasks yet to be scheduled. Hence, the proposed procedure is insensitive to the real task execution times and can be applied with either $\mathcal{B}$ policy or $\mathcal{E}$ policy. For both policies, the reduction of the battery charge consumption in the experimental results will not be changed and the conclusions on the effectiveness of the proposed methods will hold independent of the policy used. In the paper, for the sake of illustration, we assume battery-aware design or $\mathcal{B}$ policy.

The rest of the paper is organized as follows. Section 2 outlines the system and battery models. Section 3 presents the problem formulation. The proposed workload-ahead-driven online DVS technique is introduced in Section 4. Section 5 describes the proposed online rescheduling and remapping approach. The combined online DVS and ABB is presented in Section 6. Experimental results and conclusions are given in Sections 7 and 8, respectively.

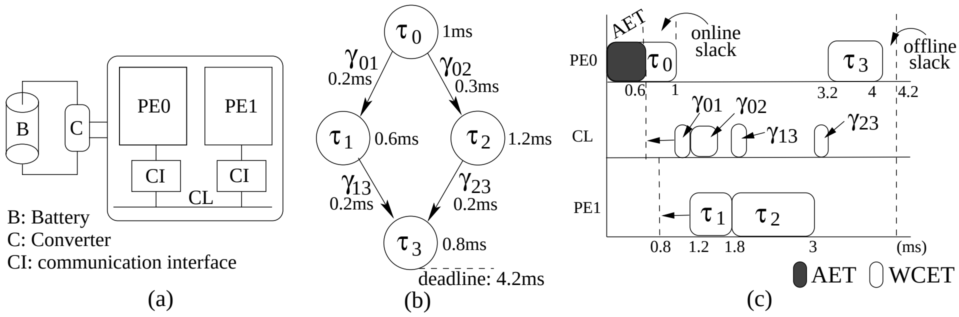

Fig. 1. Task graph and system model

2. PRELIMINARIES

2.1 System Model and Task Graph

We consider battery-powered embedded computing systems, illustrated in Fig. 1(a), which consist of multiple processing elements (PEs) connected by communication links (CLs). A dc/dc converter adapts the battery voltage to the system supply voltage. The system functionality is captured by a task graph model $G(\mathcal{T}, \mathcal{C})$, shown in Fig. 1(b). Nodes ($\tau_i \in \mathcal{T}$) in this directed acyclic graph (DAG) represent computational tasks. 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 [Luo and Jha 2002a]. The following equation gives the relationship between these parameters and the execution time.

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

where k and α are technology related constants. The power dissipation of a task can be expressed as [Andrei et al. 2004]:

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

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 procedures. In a battery-powered system, the discharge current drawn from the battery, I, equals $P/(V_b \cdot \eta)$, where V_b and η are the average battery voltage and the converter efficiency respectively and both can be regarded as constants [Rakhmatov and Vrudhula 2003]. In this paper, we set V_b and η as 5V and 0.9 respectively. Since DVS can reduce the power, P, it can be used to down scale the battery discharge current and achieve savings in the battery charge consumption [Rakhmatov and Vrudhula 2003], [Chowdhury and Chakrabarti 2002]. 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, as illustrated in 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 $0.4ms$.

2.2 Battery Model

Rao *et al.* give a comprehensive survey on battery modeling in [Rao et al. 2003]. In this work we use an analytical high-level battery model proposed in [Rakhmatov and Vrudhula 2003] whose accuracy has been demonstrated to be within 3% of the physical battery. The battery charge consumption at time t is modeled as:

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

where N is the total number of steps used to approximate the load current profile (LCP), t is the time that the battery has been discharged for, 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, β is a constant related to the non-linear property modelled by function F:

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

If the capacity of the battery is α, then solving equation $\alpha = \sigma(L)$ will give the battery lifetime L. Here the values of α and β are set to 40375 and 0.273 respectively according to [Rakhmatov and Vrudhula 2003]. Since smaller charge consumption will lead to longer battery lifetime [Rakhmatov and Vrudhula 2003], our optimization objective is the minimization of the charge consumption.

3. 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, there may be a deadline dl associated with a task. 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 5. 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 5.

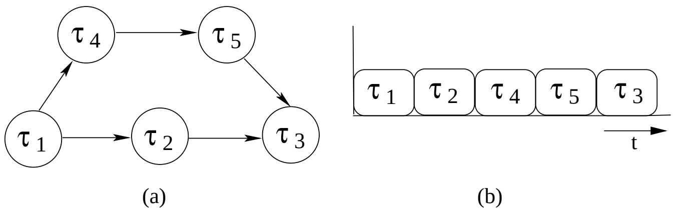

Fig. 2. Office-auto task graph [Dick] and execution order

4. BATTERY-AWARE ONLINE VOLTAGE SCALING

4.1 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 [Dick], namely the office-auto benchmark consisting of 5 tasks illustrated in Fig. 2(a). For simplicity we consider 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 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 in [Ahmed and Chakrabarti 2004], 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 is consumed, which is 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

Table I. WCETs, discharge currents, AETs and online slacks of auto-office tasks

	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	4.243	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 II. Online slack distribution

	"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

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 4.2, 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, an improvement of 18.5% when compared to the slack forwarding method [Ahmed and Chakrabarti 2004].

4.2 Workload-Ahead-Driven Online DVS Technique

As we have seen in the motivational example of Section 4.1, 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. Let τ_{next} be the next task to be executed. Denote $\mathcal{T}_r$ the set of tasks that start later than τ_{next} and τ_{next} itself. Mathematically, $\mathcal{T}_r = \{\tau_{next}, \tau_i | \tau_i \text{ starts later than } \tau_{next}\}$.

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

Definition 2: The *workload-ahead* (WA_i) of a task τ_i is the sum of the workloads of all 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. In Eq. 5, the slack allocated to a task is a function of the ratio of its workload and the total workload of the tasks yet to be scheduled, so the time unit of the task execution times is eliminated during the calculation of the slack distribution. Hence, this procedure is insensitive to the real execution times and can be applied with either $\mathcal{B}$ or $\mathcal{E}$ policies [Rao et al. 2005]. 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. Though there exist DVS algorithms that can achieve much lower battery charge consumption ([Rakhmatov and Vrudhula 2003]), the high complexity of these methods prevent them to be utilized in the online phase.

The underlying idea behind of Eq. 5 is based on the battery model, which is described by Eq. 3 and Eq. 4. Substituting Eq. 4 into Eq. 3, we obtain

$$\begin{aligned}\sigma(t) &= \sum_{k=0}^{N-1} I_k \left(\Delta_k + 2 \sum_{m=1}^{10} \frac{e^{-\beta^2 m^2 (t - st_k - \Delta_k)} - e^{-\beta^2 m^2 (t - st_k)}}{\beta^2 m^2} \right) \\ &= \sum_{k=0}^{N-1} I_k \Delta_k + 2 \sum_{k=0}^{N-1} \sum_{m=1}^{10} \frac{e^{-\beta^2 m^2 (t - st_k - \Delta_k)} - e^{-\beta^2 m^2 (t - st_k)}}{\beta^2 m^2}\end{aligned}$$

Here Δ_k is the WCET of task τ_k and $I_k \cdot \Delta_k$ is the *workload* of task τ_k. Hence a task with larger workload will consume more battery charge and should be scaled more aggressively. This has been reflected in Eq. 5, which gives more slack to a task with larger workload so that the voltage of the task can be more aggressively scaled. 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 [Chowdhury and Chakrabarti 2002]). 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 desired. 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.

Based on the 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. If the resulting frequency is less than the minimum frequency (f_{min}) of the PE, then the minimum frequency will be used.

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: $slack_{next} = online\_slack \times (W_{next}/WA_{next})$;
04: $frequency_{next} = max\{K_w/(WCET_{next} + slack_{next}),\ f_{min}\}$;
05: *Compute* V_{next} *by solving Eq.* (1)
with known V_t *and* $frequency_{next}$;
06: *return* $max\{V_{next},\ V_{min}\}$;
07: **else**
08: *call* RM-RS-DVS; // (Fig. 6)
09: **end if**

Fig. 3. Pseudo code: Workload-ahead-driven online DVS

If the resulting voltage is larger than the minimum voltage (V_{min}) of the PE, it will be returned. Otherwise, V_{min} will be returned (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 5 (line 8). It is important to note that each step in the algorithm can be performed in constant time, hence the overall complexity is constant. The constant complexity allows the scaling overhead be incorporated into the WCET of tasks during timing analysis [Shen et al. 1993]. 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. In the above DVS procedure, we did not consider the time and energy overheads of the voltage transition. The issue of handling the transition overheads has been addressed in [Andrei et al. 2004] and [Mochocki et al. 2005] for offline DVS and online DVS respectively. According to [Mochocki et al. 2005], in the online phase, when a task is ready to be executed, we can subtract the time overhead of the transition from online slack and use the remaining slack as the available slack in our DVS algorithm. If the overhead is equal to or greater than the online slack, the DVS will not be involved. Similarly, we can compare the energy saving of the ready task due to DVS with the energy overhead. If the energy saving is larger, DVS will be called, otherwise, it is rejected. In this paper, however, we omit the transition overheads for simplicity since they are not the focus of this paper.

5. 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

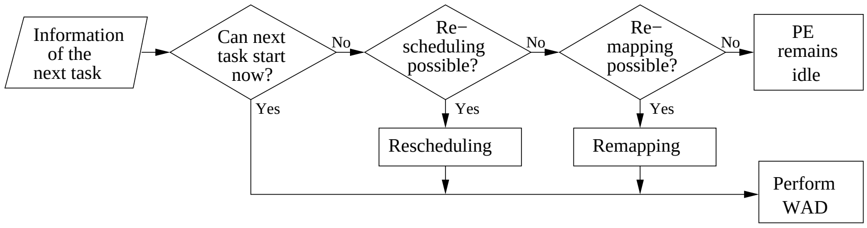

Fig. 4. Integrated workload-ahead-driven DVS and rescheduling/remapping

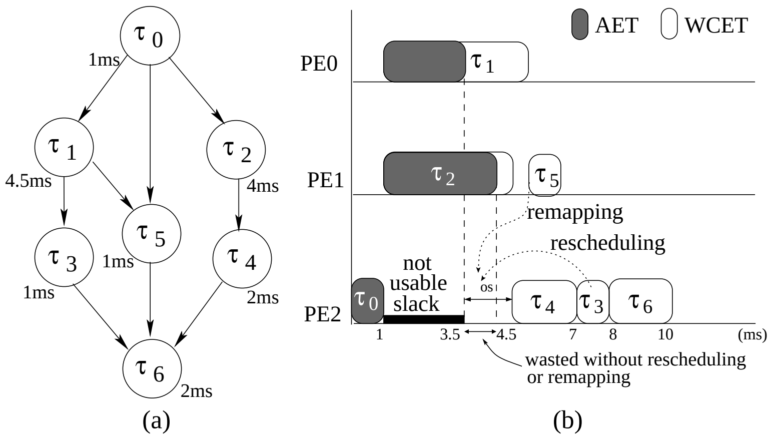

Fig. 5. Online task rescheduling and remapping

remapping as supplements of the proposed online DVS (Section 4.2). 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.

5.1 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 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

Algorithm: RM-RS-DVS

Input: - W_{next}, WA_{next}, $WCET_{next}$, ST_{next}, K_w, $CurrTime$, M

Output: - V_{next}

```
int k = min(M, size of PE[p].exe_Queue);
bool search_result = false;
for j = 1 to k  //online rescheduling
  if PE[p].exe_Queue[j] satisfies rescheduling conditions then
      search_result = true;
      move PE[p].exe_Queue[j] to the head of PE[p].exe_Queue;
      break;
  end if
end for
if search_result == true then
  call WAD-DVS;    // (Fig. 3)
else // online remapping
  for i = 1 to n && i != p
    k = min(M, size of PE[i].exe_Queue);
    for j = 1 to k
      if PE[i].exe_Queue[j] satisfies remapping conditions then
          search_result = true;
          fetch task PE[i].exe_Queue[j] from PE[i] and
          put it to the head of PE[p].exe_Queue;
          break;
      end if
    end for
    if search_result == true then
      break;
    end if
  end for
  if search_result == true then
    call WAD-DVS;    // (Fig. 3)
  else
    let PE[p] be idle;
  end if
end if
```

Fig. 6. Pseudo code: Online rescheduling/remapping

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.

5.2 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 size*. The search window size represents the maximal number of tasks in *exe_Queue* that may be rescheduled or remapped. The complexity of the algorithm is bounded by the search window size, which is constant. The algorithm first restricts the search window size 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 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 12-31). 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 27). If no task can be found remappable, the idling of PE[p] is not avoided and the online slack is wasted (line 29).

The complexities of the rescheduling and remapping algorithms are $O(M)$ and $O(n \times M)$ respectively, where M is the search window size and n is the number of PEs. Neither M nor n will change at runtime, resulting in a constant complexity of the rescheduling/remapping algorithm. Usually, n is a small integer and as we will see in Section 7, the search window size is also a small number, hence the computational overhead of the rescheduling/remapping algorithm is very low.

6. COMBINED ONLINE DVS AND ABB

In Section 4, we mainly considered the dynamic power, which has been the dominant source of power consumption in contemporary CMOS digital systems. However, with ever-shrinking feature sizes, leakage power is becoming comparable to dynamic power [Martin et al. 2002]. For the $0.05\mu m$ predictive technology, the leakage power is estimated to be almost equivalent to the dynamic power [Yan et al. 2003].

Though DVS can decrease the dynamic power significantly, it is not very effective in reducing leakage power. To tackle this problem, adaptive body biasing (ABB) has been introduced. ABB takes advantage of the fact that the leakage current can be exponentially reduced by dynamically changing the body-bias voltage (V_{bb}) of transistors [Keshavarzi et al. 2001]. A microprocessor prototype which can change V_{bb} continuously was introduced in [Miyazaki et al. 2002]. Kim and Roy proposed a scheme to scale V_{bb}, which can be included in any processor [Kim and Roy 2002]. In order to minimize the overall power, combined DVS and ABB can be utilized [Martin et al. 2002], [Yan et al. 2003], [Andrei et al. 2005]. In this section, we extend the workload ahead-driven DVS technique introduced in Section 4 towards the consideration of the leakage power, i.e., we combine DVS and ABB (referred to DVSABB from now on) to achieve power efficiency in terms of leakage and dynamic power.

The leakage power is mainly caused by subthreshold leakage currents in the CMOS circuitry and it can be modeled as [Martin et al. 2002]

$$P_{leakage} = V_{dd} \cdot K_3 \cdot e^{K_4 V_{dd}} \cdot e^{K_5 V_{bs}} \tag{6}$$

where V_{bs} is the body-bias voltage and the fitting parameters K_3, K_4, K_5 are technology dependent constants. For clarity reasons we maintain the same indices as used in [Martin et al. 2002]. Actual values for these constants are provided in [Martin et al. 2002], given a Transmeta Cruose processor. Considering Eq.2, the total power of one task can then be expressed as

$$P_{total} = P_{dynamic} + P_{leakage} = f \cdot C_e \cdot V_{dd}^2 + V_{dd} \cdot K_3 \cdot e^{K_4 V_{dd}} \cdot e^{K_5 V_{bs}} \tag{7}$$

The operational frequency f depends on V_{dd} as well as V_{bs} and can be expressed as [Martin et al. 2002]

$$f = \frac{((1 + K_1) \cdot V_{dd} + K_2 \cdot V_{bs} - V_{th1})^{\alpha}}{K_6 \cdot L_d \cdot V_{dd}} \tag{8}$$

where L_d is the logic depth, and α, K_1, K_2, K_6 and V_{th1} denote circuit dependent constants [Martin et al. 2002]. From Eq.7, we can find that scaling V_{dd} (DVS) and V_{bs} (ABB) simultaneously can effectively reduce the overall power, which will in turn reduce the current drawn from the battery. Extending the online DVS to online DVSABB does not change the *workload* and *workload-ahead* defined in Section 4.2. The only difference is that the current of the task is modified to $I = P_{total}/(V_b \cdot \eta)$.

In online DVS, we only need to compute the V_{dd} of the next task to be executed based on the slack assigned to the task (see Fig.3). The difference in online DVSABB is that except for V_{dd}, the body bias voltage V_{bs} of the next task has to be calculated. Now the problem of online DVSABB can be formulated as given a task and a certain amount of slack, finding a pair of (V_{dd}, V_{bs}) such that the total power of the task is minimized. According to Eq.8, both DVS and ABB will cause the reduction of the operational frequency and slow down the task execution. This indicates that both techniques have to utilize the online slack and they compete with each other for the available slack. Accordingly, we have to decide the optimal portion of the slack distributed between DVS and ABB in order to calculate the voltage pair (V_{dd}, V_{bs}). Once the amount of slack assigned to DVS and ABB is known, it is easy to compute the V_{dd} and V_{bs} based on Eq.8. Since the total power depends nonlinearly on V_{dd} and V_{bs} (Eq. 7), this results in a nonlinear optimization problem. Numerical methods that are commonly used to address such optimization problems are unsuitable for online techniques due to their large time overhead compared to the application running on the system. Therefore, we utilize the idea of a look-up-table (LUT) approach proposed by [Andrei et al. 2005] to solve the problem.

In this approach, each task will have a specific LUT which is set up before the actual execution (i.e., in the offline phase). Each entry of the LUT corresponds to a possible slack that the task may have. A pair of V_{dd} and V_{bs} are pre-computed based on this slack and stored in this entry. Then in the online phase, each time when a task is to be executed, the PE will look up the voltage pair in the LUT according to the actual online slack obtained by the task. In the LUT, the step between two entries is a constant. For a given slack, we can use $\lceil \frac{slack}{step} \rceil$ to find its entry. For example, the slacks in the table are 0, 0.2, 0.4, 0.6, 0.8 ms and the step is 0.2ms. If the slack distributed to the next task is 0.5ms, then $\lceil \frac{0.5}{0.2} \rceil = 3$, i.e., (V_{dd}, V_{bb}) in the 3rd entry will be used. Since this entry found procedure is quite simple, the overhead due to the table lookup is very small and constant. If the actual slack falls between two entries of the LUT, the voltage pair in the lower entry will be simply used. Though this is conservative and the slack is not fully used, the time-constraint will be guaranteed. The space complexity of the LUTs depends on the number of tasks, T, and the average number of entries of one table, E. We assume it takes 4 bytes to store the slack, V_{dd} and V_{bs} respectively, then the space to store the LUTs will be $12 \times E \times T$. For example, if $E = 20$ and $T = 100$, then the LUTs needs 24,000 bytes (approximately 23.44 KB) memory.

The LUT setup procedure is outlined next. First, we compute the possible online slack range of each task, i.e., the difference between the worst case slack (WCS) and the best case slack (BCS) that each task may have. The WCS that every task would have is clearly zero. This is the case where all tasks are executed with their WCETs. We assume each task will obtain the BCS when the AETs of all tasks are 30 percent of their WCETs. For each task, the LUT is set to be empty initially, then starting from its WCS, we add entries to its LUT with certain interval until its BCS is reached. Hence each entry of the LUT corresponds to a possible slack that the task may obtain at run time. For each entry, we use exhaustive search to find the optimal pair of (V_{dd}, V_{bs}) . Let p be the percentage of the slack distributed to DVS, then $1 - p$ percent of slack will be given to ABB. We change p from 0 to 100 percent with an interval of 1 percent. With each p, a pair of (V_{dd}, V_{bs}) is computed according to Eq.8. Then we calculate the total power P_{total} based on the computed voltage pair and Eq.7. The voltage pair corresponding to the minimum P_{total} is the optimal pair and will be chosen to fill the table entry. The computation of (V_{dd}, V_{bs}) is repeated until every entry of the LUT is filled. It can be seen that the above LUT setup procedure is time consuming, but the setup speed is not our concern since the it is carried out in the offline phase. Once the tables are set up, the online lookup will be very fast and the complexity is $O(1)$.

Table III. Results of online DVS in single PE systems

Bench-mark (# task)	battery charge consump. (mAs)				Improvem. (%)		
	SF $O(1)$	ACD $O(1)$	WAD $O(1)$	BE $O(nn_rC)$	$\frac{WAD}{SF}$	$\frac{WAD}{ACD}$	$\frac{BE}{WAD}$
Auto-ind. (28)	0.314	0.314	0.224	0.177	28.66	28.66	20.98
Consumer (27)	12.141	12.141	7.004	4.383	42.31	42.31	37.42
Office-auto (5)	1.224	0.991	0.933	0.861	23.77	5.85	7.72
Network (23)	0.237	0.237	0.166	0.115	29.96	29.96	30.72
Telecom (42)	0.438	0.438	0.310	0.189	29.22	29.22	39.03
GSM decoder (34)	3.906	3.826	2.831	1.224	27.52	26.01	56.76
GSM encoder (53)	3.534	3.249	2.417	1.168	31.61	25.61	51.68

7. 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 15 hypothetical examples as well as 7 real-world benchmakrs. The hypothetical examples have been automatically generated using TGFF [Rhodes and Dick], a pseudo-random task graph generator. The first 5 realistic examples have been taken from the E3S benchmark suit [Dick] (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 [Schmitz]. All reported results have been obtained using the battery model of Section 2.2 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 [Ahmed and Chakrabarti 2004].

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. SF**: The slack forwarding approach is based on the technique presented in [Ahmed and Chakrabarti 2004]. Its time complexity is constant ($O(1)$). **2. 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. **3. WAD**: This represents our workload-ahead-driven distribution technique, as introduced in Section 4.2. It has also a complexity of $O(1)$. Since the slack forwarding idea [Ahmed and Chakrabarti 2004] 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. **4. BE**: A best effort slack distribution adapted from an offline DVS

[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.

Table IV. Battery charge consumption of multiple periods

Bench-mark (# task)	battery charge consump. (mAmin)			Improvem. (%)	
	SF $O(1)$	ACD $O(1)$	WAD $O(1)$	$\frac{WAD}{SF}$	$\frac{WAD}{ACD}$
Auto-ind. (28)	5.041	5.041	3.618	28.22	28.22
Consumer (27)	24.406	24.406	17.187	29.58	29.58
Office-auto (5)	5.114	4.246	4.068	20.47	4.20
Network (23)	3.771	3.771	2.634	30.14	30.14
Telecom (42)	0.705	0.705	0.501	28.91	28.91
GSM decoder (34)	0.270	0.265	0.198	26.69	25.07
GSM encoder (53)	0.249	0.230	0.173	30.58	24.68

procedure of [Rakhmatov and Vrudhula 2003]. It divides the online slack into small steps. For each step, every remaining task is tried to be scaled by using this step of slack and the task which can cause the minimum battery charge consumption will be assigned with this step. This procedure is repeated until all the steps are distributed, i.e., the available online slack is used up. The complexity of this method is $O(nn_rC)$, where n is the number of all tasks, n_r is the number of the remaining tasks and C is the number of slack steps. This technique can achieve much lower battery charge consumption than the above three heuristics due to the complicated search it uses. However, its high complexity prevents it to be utilized at run-time

Table III gives the results of the four different DVS methods 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 four online DVS techniques are given in Columns 2–5. In columns 6-7, we show the percentage of improvement in battery charge consumption using the proposed WAD method over methods SF and ACD. The improvement of BE over WAD is given in the last column. Consider, for instance, the GSM decoder benchmark. Here BE obtains the minimum battery charge consumption of 1.224mAs. SF and ACD result in 3.906 mAs and 3.826mAs, respectively and WAD achieves 2.831mAs, resulting in improvements of 27.52% and 26.01% over SF and ACD. We can observe that BE yields consistently the lowest battery charge consumption among the four techniques, but its computational complexity is also much higher than the other three. Among the three heuristics with constant complexity, WAD achieves lower battery charge consumption than SF and ACD. Table III gives the battery charge consumption that the applications run for one period. We also repeat the experiments of the three heuristics with constant complexity for several thousand periods so that the total operation time reaches the scale of minutes. The resulting battery charge consumptions are shown in Table IV. From this table we can find that WAD is still more effective than SF and ACD.

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 [Schmitz et al. 2002], 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

Table V. Experimental results in multi-PE systems

Bench-marks	# task/ edge	# PE	Battery charge consumption (10^{-2}mAs)				Percentages (%)		
			No DVS	WAD	WAD+RS	WAD+RSRM	1	2	3
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	84/109	3	1.6536	1.2294	1.2060	1.1407	25.65	1.90	7.21
tgff2	196/236	3	5.9416	4.4459	4.1119	4.0206	25.17	7.51	9.57
tgff3	149/180	3	4.2885	3.2265	3.0185	2.9791	24.76	6.44	7.66
tgff4	81/103	3	2.4865	1.8633	1.7925	1.7306	25.06	3.81	7.12
tgff5	149/167	5	5.3703	4.0737	3.8586	3.8368	24.14	5.28	5.81
tgff6	70/94	2	1.0458	0.7840	0.7249	0.7248	25.02	7.54	7.55
tgff7	102/152	3	3.9872	2.8624	2.6778	2.6188	28.21	6.45	8.50
tgff8	117/170	3	3.9352	2.9009	2.7473	2.7429	26.28	5.29	5.44
tgff9	316/413	4	8.3134	6.0926	5.7592	5.6844	26.71	5.47	6.70
tgff10	269/348	3	5.4967	4.1520	3.8301	3.8114	24.46	7.75	8.20
tgff11	331/408	4	8.1994	6.0464	5.5650	5.4978	26.25	7.96	9.07
tgff12	280/341	4	10.097	7.4264	6.8689	6.7069	26.45	7.50	9.68
tgff13	378/443	4	1.3623	9.9426	9.1701	8.9537	27.01	7.77	9.94
tgff14	252/312	4	6.6971	5.0638	4.7497	4.6449	24.38	6.20	8.27
tgff15	210/234	3	0.3786	0.3183	0.2746	0.2735	15.92	13.71	14.08
						Ave. percents	24.80	4.89	6.03

1: $\frac{WAD}{NoDVS}$; 2: $\frac{WAD+RS}{WAD}$; 3: $\frac{WAD+RSRM}{WAD}$

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 [Ahmed and Chakrabarti 2004] was particularly introduced for independent tasks, we refrain in these experiments from a direct comparison.

The results of our experiments are summarized in Table V. 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+RSRM) 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 tgff2. 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 7.51% and 9.57% when compared to using WAD only.

As mentioned above, the window size used by the rescheduling and remapping

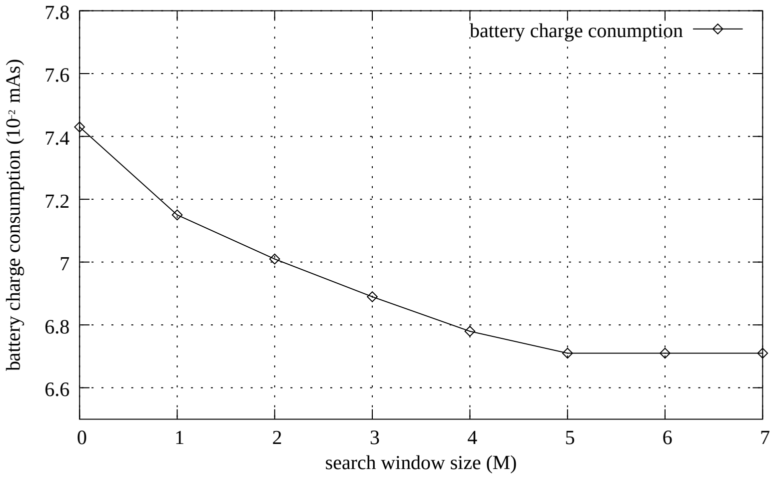

Fig. 7. Influence of search window size on rescheduling/remapping results

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 tgff12 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 for all investigated benchmarks, the window size is no larger than 10.

The above experiments were also repeated for 10 thousand times and the overall schedule lengths reach the scale of minutes and the results are given in Table VI. From this table, it can be seen that the online DVS still achieves remarkable battery charge saving compared to the situation where there is no DVS. While averagely, the rescheduling/remapping algorithms do not further reduce the battery charge consumption too much, they are effective in some benchmarks, for instance, more than 10 percent battery charge can be saved in tgff15 by using rescheduling or remapping. Hence,the rescheduling/remapping algorithm can still be utilized in certain applications.

The third set of experiments was carried out to evaluate the effectiveness of the proposed online DVSABB when the leakage power is taken into account. In the sinlge PE systems, we compare the online DVSABB with the online DVS. The proposed LUT based DVSABB (DVSABB_LUT) is also compared with the optimal DVSABB obtained through exhaustive search (DVSABB_ES). All three scaling methods are workload-ahead-driven. Initially, we assume that before scaling the leakage power of each task is equal to its dynamic power, a realistic assumption for the next generation technology [Yan et al. 2003], and the results are shown in Table VII. We generalize this assumption in a later experiment. The battery charge consumption of the three scaling approaches are given in columns 2-4 and column 5

Table VI. Experimental results in multi-PE systems with multiple periods

Bench-marks	# task/ edge	# PE	Battery charge consumption (mAmin)				Percentages (%)		
			No DVS	WAD	WAD+RS	WAD+RSRM	1	2	3
Auto-ind.	28/25	2	88.858	59.580	59.580	59.580	32.95	0	0
Consum.	27/30	3	916.53	777.34	777.34	777.34	15.19	0	0
Office-au.	5/5	1	18.780	15.289	15.289	15.289	18.59	0	0
Network.	23/19	2	29.450	20.653	20.653	20.653	29.87	0	0
Telecom.	42/40	2	57.569	43.400	43.327	43.311	24.61	0.17	0.21
GSM	87/138	6	11.576	7.725	7.601	7.601	33.26	1.61	1.61
tgff1	84/109	3	4.918	2.982	2.931	2.702	39.37	1.71	9.39
tgff2	196/236	3	10.156	6.399	5.911	5.658	36.99	7.63	11.58
tgff3	149/180	3	8.501	5.265	4.938	4.810	38.07	6.21	8.64
tgff4	81/103	3	6.515	4.032	3.891	3.668	38.11	3.50	9.03
tgff5	149/167	5	10.047	6.240	5.906	5.804	37.89	5.35	6.98
tgff6	70/94	2	3.918	2.457	2.303	2.301	37.29	6.26	6.35
tgff7	102/152	3	7.419	4.412	4.134	4.021	40.53	6.30	8.86
tgff8	117/170	3	7.893	4.735	4.498	4.442	40.01	5.02	6.19
tgff9	316/413	4	12.159	7.501	7.093	6.913	38.31	5.44	7.83
tgff10	269/348	3	9.832	6.137	5.680	5.631	37.58	7.45	8.24
tgff11	331/408	4	11.861	7.507	6.915	6.756	36.71	7.88	10.00
tgff12	280/341	4	13.644	8.631	7.981	7.588	36.74	7.53	12.08
tgff13	378/443	4	16.431	10.664	9.832	9.495	35.10	7.79	10.96
tgff14	252/312	4	10.982	6.935	6.510	6.285	36.85	6.12	9.37
tgff15	210/234	3	1.686	1.340	1.156	1.140	20.52	13.73	14.89
						Ave. percents	33.34	4.96	6.61

1: $\frac{WAD}{NoDVS}$; 2: $\frac{WAD+RS}{WAD}$; 3: $\frac{WAD+RSRM}{WAD}$

shows the improvement of DVSABB_LUT over DVS. It can be seen that when the leakage power is considered, DVSABB_LUT is more effective than DVS in reducing the battery charge consumption. For example, in the Auto-indust benchmark, DVS achieves the battery charge consumption of 0.973 mAs while DVSABB_LUT consumes only 0.715 mAs, resulting an improvement of 26.53%. Although DVS-ABB_ES can obtain even lower battery charge consumption, its high run time overhead prevents it to be used online. More importantly, from Table VII it can be seen that the results of DVSABB_LUT are quite close to that of DVSABB_ES. This demonstrates the high quality of DVSABB_LUT which is fast enough to be involved online.

In the above experiment, we assume that the leakage power of each task is equal to the dynamic power. However, the ratio between the leakage power and the dynamic power is both technology and application dependent [Wu et al. 2005] and this ratio is an important factor heavily affecting the effectiveness of DVSABB. In this experiment, we use the improvement of DVSABB over DVS as the measure of the effectiveness of DVSABB. In Table VIII, we show the effectiveness of DVSABB with six different leakage/dynamic ratios, and the results are in columns 2-7. It can be seen that with the increase of leakage/dynamic ratio, DVSABB becomes more effective compared to DVS. When the leakage power is 25% of the dynamic power (column 7), the improvement of DVSABB over DVS is marginal, e.g., 2.51% in the Auto-ind. benchmark. This indicates that when the leakage power is below

Table VII. Results of online DVSABB in single PE systems

Benchmark (# task)	battery charge consump. (mAs)			Improvement (%)
	DVSonly	DVSABB_LUT	DVSABB_ES	$\frac{DVSABB\_LUT}{DVSonly}$
Auto-ind. (28)	0.973	0.715	0.697	26.53
Consumer (27)	9.337	6.684	6.464	28.41
Office-auto (5)	0.574	0.494	0.490	13.86
Network (23)	0.395	0.317	0.313	19.87
Telecom (42)	0.875	0.627	0.613	28.38
GSM decoder (34)	22.22	16.86	16.58	24.11
GSM encoder (53)	31.70	22.92	22.37	27.39

Table VIII. Effectiveness of DVSABB with different leakage/dynamic ratio

Benchmark (# task)	$P_{leakage}/P_{dynamic}$ (more leakage power <———-> more dynamic power)					
	1.5	1.25	1	0.75	0.5	0.25
Auto-ind. (28)	33.37	29.65	26.53	18.91	12.50	2.51
Consumer (27)	34.05	31.25	28.41	23.44	12.83	3.58
Office-auto (5)	18.73	15.17	13.86	7.53	6.27	1.58
Network (23)	32.68	26.23	19.87	13.34	9.47	2.55
Telecom (42)	32.75	30.31	28.38	21.83	13.05	2.91
GSM decoder (34)	38.32	33.71	24.11	20.84	11.72	1.97
GSM encoder (53)	34.12	29.40	27.39	22.49	13.83	3.01

25% of the dynamic power, ABB technique is not necessary and we can utilize DVS solely. On the other hand, when the ratio is above 1, which represents the case of the incoming technologies ($0.05\mu m$ and below), much more battery charge can be saved by combining DVS with ABB, e.g., in the Auto-ind. benchmark, DVSABB can save 33.37% battery charge compared to only DVS when the leakage power is 1.5 times of the dynamic power.

We also compare the workload-ahead-driven DVSABB (WAD_DVSABB) with DVSABB combined with slack forwarding (SF_DVSABB) and the results are summarized in Table IX. After the online DVS is extended to online DVSABB, the proposed WAD approach also performs better than SF in terms of battery charge consumption, with up to 50% improvement.

Finally, we implement the DVSABB technique in the multiple PE systems and integrate it with online rescheduling and remapping. We compare the DVSABB with DVS in two frames: with rescheduling (RS); with both rescheduling and remapping (RSRM), and assume that for each PE, the leakage power equals the dynamic power. The battery charge consumptions and the improvement of DVSABB over DVS are summarized in Table X. From the table we can see that in each frame, DVSABB results in lower battery charge consumption compared to DVS in both RS case and RSRM case. This indicates that DVSABB is also more effective than DVS in reducing the battery charge consumption in the multiple PE systems, with up to 29% improvement.

Table IX. Comparison of WAD_DVSABB with SF_DVSABB

Benchmark (# task)	battery charge consump. (mAs)		Improvemment (%)
	WAD	SF	$\frac{WAD}{SF}$
Auto-ind. (28)	0.715	1.407	49.17
Consumer (27)	6.684	13.59	50.82
Office-auto (5)	0.494	0.718	31.16
Network (23)	0.317	0.494	35.93
Telecom (42)	0.626	1.186	47.16
GSM decoder (34)	16.863	31.178	45.91
GSM encoder (53)	22.928	41.356	44.56

Table X. Results of online DVSABB in multiple PE systems

Bench-marks	RS			RS+RM		
	DVS	DVSABB	Imp.(%)	DVS	DVSABB	Imp.(%)
Auto-ind.	28.516	25.917	9.11	28.516	25.917	9.11
Consumer	925.78	787.78	14.91	925.78	787.78	14.91
Office-au.	57.395	49.441	13.86	57.395	49.441	13.86
Network.	15.419	12.330	20.03	15.419	12.330	20.03
Telecom.	51.904	42.179	18.74	51.853	42.092	18.81
GSM	17.010	11.977	29.59	17.010	11.977	29.59
tgff1	1.6245	1.3233	18.55	1.4975	1.2298	16.48
tgff2	5.4656	4.5889	16.04	5.1114	4.3027	14.70
tgff3	4.0303	3.4909	13.38	4.0248	3.3696	16.26
tgff4	2.6146	2.1238	18.77	2.6146	2.0979	19.76
tgff5	5.4529	4.4664	18.09	5.3567	4.3950	17.64
tgff6	1.0527	0.8846	15.97	1.0527	0.8846	15.97
tgff7	3.8230	3.2150	15.90	3.6591	3.0970	14.70
tgff8	3.9305	3.3675	14.33	3.9270	3.3541	14.57
tgff9	7.7105	6.7814	12.05	7.1859	6.4756	9.21
tgff10	5.2881	4.6182	12.67	5.1270	4.4986	11.88
tgff11	7.5737	6.4379	15.00	7.3026	6.2677	13.66
tgff12	9.3853	7.8316	16.55	8.9440	7.4716	15.69
tgff13	1.2847	1.0868	15.40	1.2226	1.0516	13.31
tgff14	6.5825	5.5682	15.41	5.5939	5.2927	9.82
tgff15	0.3499	0.3142	10.20	0.3481	0.3128	10.15

8. 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, 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. All proposed online techniques are extended to combine the adaptive body biasing with dynamic voltage scaling so that the leakage power can also be effectively reduced.

ACKNOWLEDGMENTS

This work is supported in part by the EPSRC, U.K., under grant GR/S95770, the authors would like to acknowledge this support. The authors also wish to acknowledge the reviewers for their comments based on which we could improve the clarity of this work.

REFERENCES

Ahmed, J. and Chakrabarti, C. 2004. A dynamic task scheduling algorithm for battery powered dvs systems. In *Proceedings of Int. Symp. Circuits and Systems*. 813–816.

Andrei, A., Schmitz, M. T., Eles, P., Peng, Z., and Al-Hashimi, B. M. 2004. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *Proceedings of Design, Automation and Test in Europe Conf.* 518–523.

Andrei, A., Schmitz, M. T., Eles, P., Peng, Z., and Al-Hashimi, B. M. 2005. Quasi-static voltage scaling for energy minimization with time constraints. In *Proceedings of Design, Automation and Test in Europe Conf.* 514–519.

Aydin, H., Melhem, R., Mosse, D., and Mejia-Alvarez, P. 2001. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of Real-Time System Symp.* 95–105.

Chowdhury, P. and Chakrabarti, C. 2002. Battery aware task scheduling for a system-on-a-chip using voltage/clock scaling. In *Proceedings of IEEE Workshop on Signal Processing Systems*. 201–206.

Dick, R. E3s benchmark suite. http://www.ece.northwestern.edu/ dickrp/e3s/.

Keshavarzi, A., S. Ma, S. N., Blocechel, B., K. Mistry, T. G., Borkari, S., and De, V. 2001. Effectiveness of reverse body bias for leakage control in scaled dual-vt cmos ics. In *Proceedings of Int. Symp. Low Power Electronics and Design*. 207–212.

Kim, C. H. and Roy, K. 2002. Dynamic v_{TH} scaling scheme for active leakage power reduction. In *Proceedings of Design, Automation and Test in Europe Conf.* 163–167.

Kim, W., Shin, D., Yun, H. S., Kim, J., and Min, S. L. 2002. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symp.* 219–228.

Luo, J. and Jha, N. K. 2001. Battery-aware static scheduling for distributed real-time embedded systems. In *Proceedings of IEEE Design Automation Conf.* 444–449.

Luo, J. and Jha, N. K. 2002a. Low power distributed embedded systems: Dynamic voltage scaling and synthesis. In *Proceedings of the International Conference on High Performance Computing*.

Luo, J. and Jha, N. K. 2002b. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *Proceedings of Asia and South Pacific Design Automation Conf.* 712–719.

Martin, S., Flautner, K., Mudge, T., and Blaauw, D. 2002. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of IEEE/ACM Int. Conf. Computer-Aided Design*. 721–725.

Miyazaki, M., Ono, G., and Nagamatsu, T. 2002. A 1.2-gips/w microprocessor using speed-adaptive threshold-voltage cmos with forward bias. *IEEE Trans. on Solid-State Circuits 37,* 2, 210–217.

Mochocki, B., Hu, X. S., and Quan, G. 2005. Practical on-line dvs scheduling for fixed-priority real-time systems. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symp.* 224–233.

PILLAI, P. AND SHIN, K. G. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of ACM Symp. Operating Systems Principles*. 89–102.

RAKHMATOV, D. AND VRUDHULA, S. 2003. Energy management for battery-powered embedded systems. *ACM Trans. on Embedded Computing Systems 2,* 3, 277–324.

RAO, R., VRUDHULA, S., AND CHANG, N. 2005. Battery optimization vs energy optimization: Which to choose and when? In *Proceedings of IEEE/ACM Int. Conf. Computer-Aided Design*. 438–444.

RAO, R., VRUDHULA, S., AND RAKHMATOV, D. 2003. Battery modeling for energy-aware system design. *IEEE Computer 36,* 12, 77–87.

RHODES, D. AND DICK, R. Task graph for free (tgff). http://ziyang.ece.northwestern.edu/tgff/.

SCHMITZ, C. Gsm phone task graphs. http://kbs.cs.tu-berlin.de/ jutta/toast.html.

SCHMITZ, M. T. AND AL-HASHIMI, B. M. 2001. Considering power variations of dvs processing elements for energy minimization in distributed systems. In *Proceedings of Int. Symp. System Synthesis*. 250–255.

SCHMITZ, M. T., AL-HASHIMI, B. M., AND ELES, P. 2002. Synthesizing energy-efficient embedded systems with lopocos. *Design Automation for Embedded Systems 6*, 401–424.

SHEN, C., RAMAMRITHAM, K., AND STANKOVIC, J. A. 1993. Resource reclaiming in multiprocessor real-time systems. *IEEE Trans. on Parallel and Distributed Systems 4,* 4, 382–397.

SHIN, D. AND KIM, J. 2001. A profile-based energy-efficient itra-task voltage scheduling algorithm for hard real-time applications. In *Proceedings of Int. Symp. Low Power Electronics and Design*. 271–274.

WU, D., AL-HASHIMI, B. M., SCHMITZ, M. T., AND ELES, P. 2005. Power-composition profile driven co-synthesis with power management selection for dynamic and leakage energy reduction. In *Proceedings of Euromicro Digital System Design*. 34–41.

YAN, L., LUO, J., AND JHA, N. K. 2003. Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. In *Proceedings of IEEE/ACM Int. Conf. Computer-Aided Design*. 30–37.

YE, W. AND ERNST, R. 1997. Embedded program timing analysis based on path clustering and architecture classification. In *Proceedings of IEEE/ACM Int. Conf.Computer-Aided Design*. 598–604.

ZHU, D., MELHEM, R., AND CHILDERS, B. R. 2003. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. *IEEE Trans. on Parallel and Distributed Systems 14,* 7, 686–700.

ZHU, Y. AND MUELLER, F. 2004. Feedback edf scheduling exploiting dynamic voltage scaling. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symp.* 84–89.

...