[15] S. P. Lin and Y. W. Chang, "MR: A new framework for multilevel full-chip routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 793–800, May 2004.
[16] L. He and K. M. Lepak, "Simultaneous shield insertion and net ordering for capacitive and inductive coupling minimization," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 9, no. 3, pp. 290–309, Jul. 2000.
[17] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli, "Digital sensitivity: Predicting signal interaction using functional analysis," in *Proc. ICCAD*, 1996, pp. 536–541.
[18] M. R. Prasad, D. Kirkpatrick, and R. K. Brayton, "Domino logic synthesis and technology mapping," in *Proc. IWLS*, 1997.

Power-Efficient Scheduling for Heterogeneous Distributed Real-Time Embedded Systems

Jiong Luo and Niraj K. Jha

***Abstract*—This paper addresses the problem of variable-voltage scheduling of multirate periodic task graphs (i.e., tasks with precedence relationships) in heterogeneous distributed real-time embedded systems. Such an embedded system may contain general-purpose processors, field-programmable gate arrays, and application-specific integrated circuits. First, we discuss the implications of the distribution of power consumption, i.e., power profile, of tasks and characteristics of voltage-scalable processing elements (PEs) on variable-voltage scaling. Then, we present a power-efficient variable-voltage scheduling algorithm to address these implications. The scheduling algorithm performs execution order optimization of scheduled events to increase the chances of scaling down voltages and frequencies of these voltage-scalable PEs in the distributed embedded system. It also performs power-profile and timing-constraint driven slack allocation to maximize power reduction via voltage scaling, based on the observation that the energy consumption of a task on a voltage-scalable PE is normally a convex function of the clock speed. The scheduling algorithm is also effective in the case where the variations in power consumption of different tasks can be ignored. It can be included in the inner loop of a system-level synthesis tool for design space exploration of real-time heterogeneous embedded systems, since it is very fast. We show its efficacy by comparing it to other approaches from the literature.**

***Index Terms*—Distributed systems, low power, scheduling, voltage scaling.**

I. Introduction

Reducing power consumption is one of the crucial design considerations for modern electronic systems, in order to reduce chip packaging and cooling costs, increase system reliability, as well as extend battery lifetime of portable systems. This paper addresses system-level power-aware scheduling of heterogeneous real-time distributed embedded systems [1], [2], which are generally composed of a heterogeneous network of processing elements (PEs), where a PE can be a general-purpose processor, an application-specific integrated circuit (ASIC), or a field-programmable gate array (FPGA). The embedded-system specification contains periodic task graphs with precedence relationships. Given an embedded-system specification, a hardware-software cosynthesis system determines the number and type of PEs and communication links that need to be used (i.e., the allocation step). In addition, the system assigns each task to a PE and each task communication to a communication link (i.e., the assignment step). Finally, a schedule is provided for each PE and communication link such that all real-time constraints can be met.

Dynamic voltage scaling refers to dynamic adjustment of the supply voltage to the minimum level required for a PE to work at a desired clock frequency. There are many commercially available voltage-scalable processors, including Intel's Xscale [3], Transmeta's Crusoe [4], and AMD's mobile processors with AMD PowerNow! technology support [5]. There are also voltage-scalable systems developed in academia [12], [25]. Voltage scaling has been widely acknowledged as a powerful and feasible technique for trading off power consumption for execution time.

There is a rich literature addressing variable-voltage scheduling for a set of independent tasks with hard deadlines on a single processor [6]. The work in [7] addresses hardware–software cosynthesis with variable-voltage scaling for single-processor core-based systems with independent tasks. The work in [8] exploits the characteristics of convex functions for optimization and gives an offline algorithm, which generates a minimum-energy preemptive schedule for a set of independent tasks. This schedule, which is based on an earliest-deadline-first (EDF) scheme, tries to achieve a uniform scaling of voltage levels of different tasks. The work in [9] provides a heuristic for a similar problem as in [8] for fixed-priority static scheduling. The work in [10] uses an energy-priority heuristic for nonpreemptive scheduling. The work in [11] points out that variations in power consumption among tasks invalidate the conclusion in [8] that a uniform scaling is optimal. It proposes an iterative slack-allocation algorithm based on the Lagrange Multiplier method. There is also work addressing dynamic voltage scaling for soft real-time systems [26], [27]. In such systems, it is preferred, but not required, that tasks meet their deadlines. Dynamic-voltage-scaling techniques for soft real-time systems need to tradeoff power savings for average response times of tasks.

Real-time scheduling for tasks with precedence relationships on distributed systems has been addressed in [22] and [28]–[30]. There is also work addressing variable-voltage scaling for such systems [13]–[16], [31]. The work in [13] uses a combined global/local-search strategy. It uses a genetic algorithm with simulated heating for global search and hill climbing and Monte Carlo techniques for local search. The work in [14] formulates the problem as a linear-programming (LP) problem for continuous voltage levels, which can be solved in polynomial time. Other works are mainly based on list scheduling. The work in [15] is based on a list-scheduling heuristic with a special priority function to tradeoff energy reduction for delay. The schedule is constructed step by step. At each step, a ready task is selected based on its assigned priority and is scheduled in a time step at which the partial schedule can achieve a maximum probabilistic energy reduction. Therefore, the complexity of this approach is high due to the number of discrete time steps that need to be evaluated when scheduling a task. Moreover, probabilistic evaluation of energy reduction of a partial schedule does not necessarily yield the best decision for the final schedule. The work in [16] uses a genetic algorithm to optimize task assignment, a genetic-list scheduling algorithm to optimize the task execution order, and an iterative slack-allocation scheme, which allocates a small time unit to the task that leads to the most energy reduction in each step. The performance and complexity of this approach are dependent on the size of the time unit, which however, cannot be determined systematically. The usage of a small time unit for task extension can lead to large computational

Manuscript received November 18, 2002; revised April 29, 2004, September 14, 2004, and June 10, 2005. This work was supported by DARPA under Contracts DAAB07-00-C-L516 and DAAB07-02-C-P302. This paper was recommended by Associate Editor R. Gupta.

J. Luo was with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA. She is now with Synopsys Inc., Mountain View, CA 94040 USA (e-mail: jiongl@synopsys.com).

N. K. Jha is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: jha@ee.princeton.edu).

Digital Object Identifier 10.1109/TCAD.2006.885736

complexity. The work in [31] performs communication-speed selection for communication links together with dynamic voltage scaling on processors to achieve a tradeoff between communication and computation power.

For distributed systems, allocation/assignment and scheduling have each been proven to be NP-complete. For variable-voltage scheduling, the problem is more challenging since the supply voltages for executing tasks have to be optimized to maximize power savings. This requires intelligent slack allocation among tasks. The previous work using global/search strategy [13] and integer linear programming (ILP) [14] can lead to large computational complexity. The problem is further complicated when slack allocation has to consider variations in power consumption among different tasks. Among previous works, those in [11], [14], and [16] consider variations in power consumption among different tasks. However, the work in [11] only addresses independent tasks in a single-processor system. To overcome this limitation, in this paper, we address the issue of variable-voltage static scheduling in a heterogeneous distributed embedded system for a set of periodic task graphs with precedence relationships, hard deadlines, and tasks with varying power consumption. The work in [14] and [16] also targets distributed embedded systems. However, their approaches are not efficient in terms of run-time. Moreover, in the approach in [16], there is no systematic method provided to determine the time unit for slack allocation. A bad choice of the time unit would adversely affect the power saving that can be achieved. To overcome these limitations, we present a power-efficient variable-voltage scheduling heuristic based on execution-order optimization and power-profile and timing-constraint guided slack allocation. The slack-allocation heuristic is efficient in terms of run-time and can provide close-to-optimal power savings.

This paper is organized as follows. In Section II, we discuss preliminaries of task graphs. In Section III, we discuss the energy-consumption model for variable voltage scaling. In Section IV, we present the variable-voltage scheduling algorithm in detail. Finally, we present the experimental results and conclusions in Sections V and VI, respectively.

II. Preliminaries

In this section, we present some preliminary concepts. For our scheduling algorithm, we assume that the PE/link allocation and task/communication assignment have already been performed. We assume the embedded system is specified in the form of periodic task graphs. A task graph is a directed acyclic graph, in which each node is associated with a task and each edge is associated with the amount of data that must be transferred between connected tasks. For each task (communication edge), the worst case execution (communication) time is specified on the PE (link) it is assigned to. The period associated with a task graph gives the time interval after which it executes again. An arrival time s (deadline d), the time by which the task associated with the node can begin (must complete) its execution, exists for every source (sink) node. Arrival times (deadlines) may exist for some intermediate nodes as well. An embedded system may contain multiple task graphs with different periods. Such a system is called multirate. The hyperperiod of the system is the least common multiple of all the periods in the system. It is known that scheduling in the hyperperiod gives a valid schedule [20]. We use n to denote the total number of tasks in all the task graphs and k to denote the total number of inter-PE communication edges.

Fig. 1 shows an embedded-system specification containing two task graphs, each with the same period of 17 time units. The arrival times for the source nodes $T1$, $T2$, and $T4$ are all zero. The deadlines for the sink nodes $T1$, $T3$, $T6$, and $T7$ are 10, 16.5, 17, and 17 time units, respectively. The worst case execution time (wcet) of tasks $T1$–$T7$ at maximum voltage are shown next to the respective nodes. The communication time on edge $E1$ is assumed to be three time units. The communication times on the other edges are not shown, since their corresponding tasks are assumed to be mapped to the same PE. As in traditional distributed computing, we also assume that intra-PE task communication takes negligible time compared to inter-PE task communication.

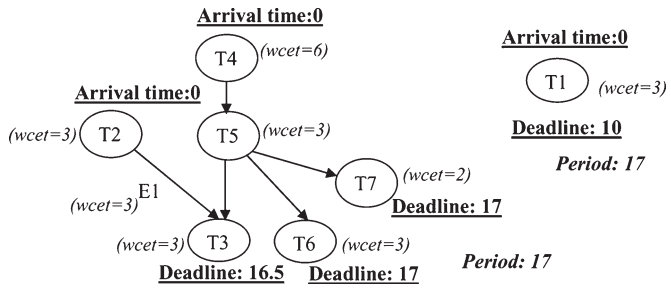

Fig. 1. Task graphs.

III. Energy-Consumption Model for Variable-Voltage Scaling

This section discusses the energy-consumption model for a set of tasks and the effect of switching activities on optimal voltage scaling, under the assumption that the voltage levels can be changed in a continuous way, or in relatively small steps (e.g., in Transmeta Crusoe [4]). For today's deep-submicrometer CMOS technology, the delay t_d of a circuit can be expressed in terms of the supply voltage $V_{\rm dd}$ and threshold voltage V_t as follows (K is a constant) [18]:

$$t_d = KV_{\rm dd}/(V_{\rm dd} - V_t)^{\alpha} \tag{1}$$

where $1 < \alpha \leq 2$. The clock frequency f of a circuit can be represented as

$$f = 1/T \tag{2}$$

where T is the longest path delay of the circuit. The power consumption p can be expressed in terms of the clock frequency f, switching activity N, capacitance C, and the supply voltage $V_{\rm dd}$ as

$$p = 1/2fNCV_{\rm dd}^2. \tag{3}$$

Given the number of clock cycles η_i for executing task i with switching activity N_i, its energy consumption E_i under supply voltage V_i and clock frequency f_i is given by

$$E_i = t_i * \left(1/2f_iN_iCV_i^2\right) \tag{4}$$

where

$$t_i = \eta_i/f_i \tag{5}$$

is the task's corresponding execution time under clock frequency f_i. Based on (1), when voltage scaling is performed, the supply voltage can be represented as a function of the task execution time. Hence, the energy consumption under voltage scaling in (4) can also be represented as a function of the task execution time. The curves of per-cycle energy consumption versus the clock period for different switching activities are shown in Fig. 2. Based on (1), (2), and (4), we can derive the second-order derivative of energy consumption

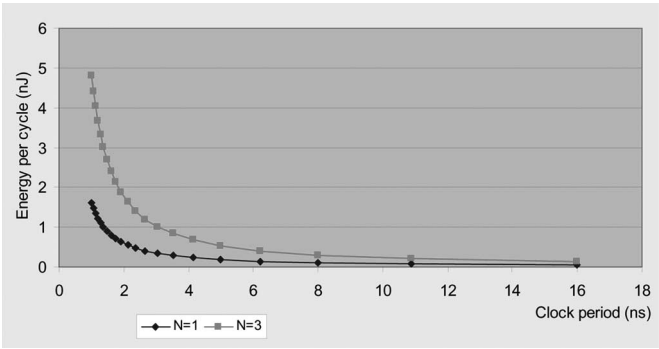

Fig. 2. Per-cycle energy consumption for different switching activities.

under voltage scaling with respect to the task execution time, which is greater than one. Hence, the energy consumption versus task-execution-time function is convex under the variable-voltage scenario. The convexity can also be observed in the curves shown in Fig. 2. The switching activity of a task can be measured through power-analysis tools discussed in the literature, such as the tool presented in [17].

When the execution time of task i is extended by dt time unit, the clock period can be extended by dt/η_i. Correspondingly, the clock frequency and supply voltage can be scaled down based on (1) and (2), and the energy consumption of task i is reduced as well. We define the energy gradient G_i to be the negative of the derivative of energy consumption with respect to the execution time of task i under supply voltage V_i

$$G_i(V_i, N_i) = -\frac{\partial E_i(V_i, N_i)}{\partial t} = 1/K * N_i C * \frac{V_i(V_i - V_t)^{(1+\alpha)}}{V_i(\alpha - 1) + V_t}. \tag{6}$$

A similar equation is given in [11]. The energy gradient indicates the energy reduction rate, if the execution time of a task is extended by dt time unit.

Assume $f_{\max}(V_{\max})$ is the maximum frequency (supply voltage), and $p_{i,f_{\max}}$ is the power consumption of task i at frequency $f_{\max}$. Then, the energy gradient can also be expressed as

$$G_i(V_i, N_i) = \frac{2p_{i,f_{\max}}}{((V_{\max} - V_t)^{\alpha} * V_{\max})} * \frac{V_i(V_i - V_t)^{(1+\alpha)}}{V_i(\alpha - 1) + V_t}. \tag{7}$$

Assume that the total slack in the system is S_{total}. If we represent the overall slack as an interval $[0, S_{\text{total}}]$, then the energy reduction resulting from voltage scaling after the overall execution times of all the tasks are extended by S_{total} can be represented as

$$\delta E = \int_0^{S_{\text{total}}} G(t)dt \tag{8}$$

where $G(t)dt$ refers to the energy reduction achieved by allocating dt unit of slack $[t, t + dt]$ from overall slack to the execution of some task. Optimal energy reduction is achieved by maximizing the integral of energy gradients over the overall slack.

A. Energy Gradient Curve

For a task to be executed on a voltage-scalable PE, its energy-gradient curve is a function of both supply voltage V_{dd} and switching activity N. Fig. 3 shows energy-gradient curves at different switching activities with respect to the supply voltage, assuming $\alpha = 2$. The voltage range for $N = 1$ and $N = 3$ is [0.75 V, 1.8 V] and V_t is assumed to be 0.6 V. The energy gradient at a given switching activity is a monotonically increasing function with respect to the supply voltage. This holds true so long as the energy consumption is a convex function of the clock period.

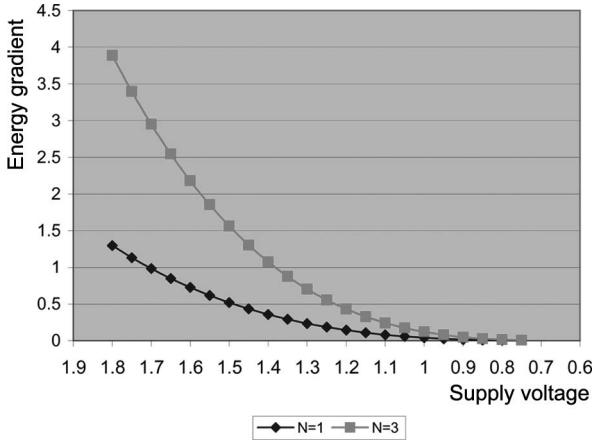

Fig. 3. Energy-gradient curves.

From Fig. 3, we can compare two curves with different switching activities ($N = 1$ and $N = 3$). There are two important regions in these two curves. For the curve with a higher N, initially, the energy gradient is higher under the same supply voltage. Therefore, it is critical to allocate slack to it until it descends to the point where the energy gradient is equal to the initial energy gradient (at the maximum supply voltage) of the curve with a lower N. After that, it is critical to allocate slack to them in a balanced way such that the integral in (8) can be maximized.

B. Case for Single Processor

In hard real-time systems, all the tasks should be executed in the processor in nonoverlapping intervals without violating their arrival time s and deadline d. An optimal slack allocation can be achieved by always allocating slack to the execution cycles of the set of extensible tasks with the highest energy gradient. A task is defined as extensible if there exists a time unit δt such that extending its execution time by δt would not lead to violations of its own deadline or that of other tasks. The allocation of slack to a set of tasks at the highest energy-gradient level should be done in a balanced way such that the integral in (8) can be maximized. If the initial schedule for a set of tasks under the maximum supply voltage and frequency is given, the conditions for optimal slack allocation are stated in the following theorem.

Theorem 1: Suppose a set of independent tasks is given whose voltage levels can be scaled in a continuous fashion. For optimal slack allocation, at any point, the supply voltages for the set of all extensible tasks, $T_1, T_2, \ldots, T_n$, should satisfy

$$\begin{aligned} & G_i(V_i) = \max_{j \in [1,\ldots,n]} G_j(V_j) \\ \text{or} \quad & V_i = V_{\max} \\ \text{or} \quad & V_i = V_{\min} \qquad \forall i \in [1, n] \end{aligned} \tag{9}$$

where $V_{\max}$ and $V_{\min}$ are the maximum and minimum voltage levels of the voltage-scalable PE, respectively.

Proof: After slack allocation, assume that the slack allocated to task T_i is δ_i. Assume that its energy gradient is $G_i(t)$ over the interval $[0, \delta_i]$, where $G_i(0)$ is the initial energy gradient before slack allocation. Furthermore, assume that the total available slack to

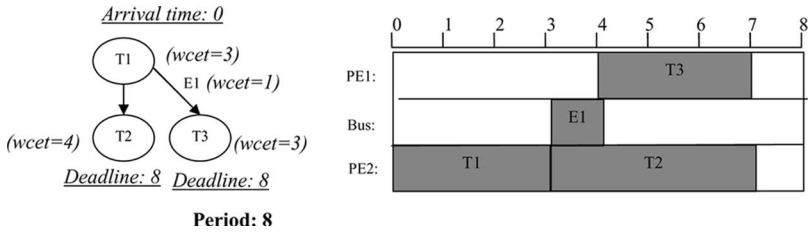

Fig. 4. Distributed-embedded-system example.

these extensible tasks is Z. Then, the energy saving $\triangle E$ after slack allocation can be represented as

$$\triangle E = \sum_{j \in [1,\ldots,n]} \int_0^{\delta_j} G_j(t)dt \tag{10}$$

under the constraint that

$$\sum_{j \in [1,\ldots,n]} \delta_j = Z \tag{11}$$

and

$$G_i(V_{\min}) \leq G_i \leq G_i(V_{\max}). \tag{12}$$

The optimal solution should satisfy

$$\frac{\partial \triangle E}{\partial \delta_i} + \frac{\partial \left(\lambda \left(Z - \sum_j \delta_j\right)\right)}{\partial \delta_i} = 0 \qquad \forall i \tag{13}$$

where λ is the Lagrange Multiplier.

Based on (11)–(13), we can derive the solution shown in (9). If for any task T_i extending its execution time further by dt would lead to violations of timing constraints of itself or other tasks, then T_i is marked as nonextensible and no more slack is allocated to it. Theorem 1 gives a similar result as the Lagrange Multiplier method presented in [11]. The optimal task priority assignment for preemptive scheduling should be an EDF schedule [19], even in the case where the power profile of tasks is considered since it can maximize S_{total}. ■

C. Case for Distributed Systems

The case for distributed systems is different due to the complications arising from precedence relationships among tasks and resource contentions on PEs and communication links. One difference for distributed systems is that S_{total} is not fixed when the initial schedule is given. It is dependent on how the slack is distributed among different tasks. For example, Fig. 4 shows a distributed system containing two PEs connected by a bus. If we fully extend the execution time of task $T1$, the achieved S_{total} is one time unit. On the other hand, if we fully extend the execution time of both tasks $T2$ and $T3$, the achieved S_{total} are two time units. Thus, allocation of slack based on Theorem 1 is no longer optimal. However, experimental results given later show that this is still an effective heuristic. In Section IV, we discuss the scheduling algorithms for distributed embedded systems in detail.

IV. Static Variable-Voltage Scheduling for Multirate Periodic Task Graphs

This section presents a variable-voltage static scheduling algorithm for multirate periodic task graphs under a given PE/link allocation and task/communication assignment. For a multirate system, the schedule is generated for all the instances of tasks and communication edges in the hyperperiod. It is worth mentioning that in our approach, we decouple the problems of allocation/assignment and scheduling. Allocation/assignment can be done using any algorithm, e.g., constructive, iterative improvement, genetic, etc. Our scheduling algorithm is based on critical-path-based list scheduling [22], which has been proven to be able to provide near-optimal solutions for many applications (when no voltage scaling is done). The list-scheduling algorithm maintains a pending list of ready tasks whose parent tasks have been scheduled. The pending list is ordered by task priorities. Each time, the task with the highest priority in the pending list is selected for scheduling. First, all its incoming edges are scheduled, and then, it is scheduled in the earliest possible slot without violating precedence relationships. The pending list is then updated with new ready tasks.

List scheduling has been widely accepted for scheduling of distributed systems. To adapt a list-scheduling algorithm to variable-voltage scheduling, two aspects need to be addressed. First, a priority assignment, which determines the execution order of events, should be able to maximize the slack available in the schedule. Second, slack needs to be allocated efficiently to maximize energy savings. In this section, the first aspect is addressed through initial priority assignment and execution-order optimization, as discussed in Section IV-A and B, respectively. The second aspect is addressed through power-profile and timing-constraint driven slack allocation, as discussed in Section IV-C.

There is a rich literature for deriving different priority assignments for list scheduling in order to optimize schedule length or schedulability. Priority assignment can be based on earliest start time, latest start time, and relative mobility (which is defined as the difference between the latest start time and earliest start time), as summarized in [22]. More sophisticated priority-assignment schemes have been proposed too. Dynamic level scheduling [30] utilizes a priority function-based dynamic level of a node, which is defined as the difference between the critical-path length of the node and its earliest start time. Here, the critical-path length of a node is defined as the longest path from the node to a sink node. The work in [28] uses a priority function based on a node's earliest start time plus its critical-path length. The priority function for list scheduling in [29] is based on the partial length of a node's critical path, which only takes into account the execution times of nodes not assigned to the same PE as this node. The tradeoffs involved in different list-scheduling algorithms are discussed in [22].

Normally, priorities are assigned statically based on the characteristics of task graphs before the scheduling process starts. A variant of list scheduling is to determine priorities dynamically during the scheduling process. The priorities of events are recomputed after an event is scheduled, in order to capture the changes in the relative criticality of the event. However, even with dynamic adjustment, it is still impossible to capture all resource contentions during the scheduling process since bus contention and resource sharing on each PE and communication link can only be fully determined after the schedule is generated. Therefore, after a valid schedule is initially generated, there should be room for execution-order adjustment of scheduled events, which may lead to better schedule flexibility and, correspondingly, more power savings. The execution order of scheduling events can be optimized through greedy iterative improvement. However, the

solution may be trapped into local minima. In order to achieve a globally optimal solution, we chose to use a simulated-annealing approach to further optimize the execution order of events based on the initial priority assignment.

This section is organized as follows. In Section IV-A, we discuss briefly the initial priority assignment. In Section IV-B, we discuss in detail execution-order optimization based on simulated annealing. In Section IV-C, we present the power-profile and timing-constraint driven slack-allocation algorithm for variable-voltage scaling. In Section IV-D, we present an example to illustrate the different steps.

A. Initial Priority Assignment

Initially, we utilize a priority assignment using the inverse of the as-late-as-possible start time ($ALAP$_start). $ALAP$_start refers to the latest time at which a scheduling event can start, without violating deadlines of any task and the precedence relationships specified in the task graphs when no resource constraint is considered. Let d_i denote the deadline specified in the task graphs for task i, $\text{out}_{\text{edges}}(i)$ denote the set of out-going edges of task i in the task graphs, $\text{successor}(e)$ denote the successor node of an edge e, and wcet_j denote the wcet of the task or communication edge j. The wcet of a task is a function of its working clock frequency, as discussed in Section II. We assume that intra-PE communication takes zero time. We define $ALAP$_start of a task and edge as follows.

For a task i

$$ALAP\_\text{start}_i = \min\left(d_i, \min_{j \in \text{out}_{\text{edges}}(i)}(ALAP\_\text{start}_j - \text{wcet}_j)\right) - \text{wcet}_i. \quad (14)$$

For a communication edge e

$$ALAP\_\text{start}_e = ALAP\_\text{start}_{\text{successor}(e)} - \text{wcet}_e. \quad (15)$$

Let s_i denote the arrival time specified in the task graphs for task i, $\text{in}_{\text{edges}}(i)$ denote the set of incoming edges of task i in the task graphs, and $\text{predecessor}(e)$ denote the predecessor node of an edge e. Similarly, we define the as-soon-as-possible start time ($ASAP$_start) of a task i as

$$ASAP\_\text{start}_i = \max\left(s_i, \max_{j \in \text{in}_{\text{edges}}(i)}(ASAP\_\text{start}_j + \text{wcet}_j)\right). \quad (16)$$

We define $ASAP$_start of a communication edge e as

$$ASAP\_\text{start}_e = ASAP\_\text{start}_{\text{predecessor}(e)} + \text{wcet}_{\text{predecessor}(e)}. \quad (17)$$

In (14) and (16), the wcet of a task is based on the maximum supply voltage and frequency.

The $ALAP$_finish of a scheduling event i is

$$ALAP\_\text{finish}_i = ALAP\_\text{start}_i + \text{wcet}_i. \quad (18)$$

In our experimental results, for comparison, we will also present the results of using the inverse of $ASAP$_start as the priority assignment. According to [22], $ALAP$_start-based list scheduling provides comparable or better results than most other scheduling approaches in terms of schedule length or schedulability. More sophisticated scheduling algorithms, such as dynamic critical-path scheduling [22], have the potential to reduce the schedule length, introduce more slack into the system, and possibly achieve better power savings. However, although in this paper we utilize an $ALAP$_start-based list scheduling, the slack-allocation techniques discussed later are general and also applicable to other scheduling algorithms.

B. Execution-Order Optimization of Scheduled Events

Instead of exploring dynamic adjustment of task priorities during the process of scheduling, as in some previous work [15], [22], we use a simulated-annealing approach, since it is impossible to incorporate the information on all resource contentions during the scheduling process. Simulated annealing is an optimization technique, which tries to find a good solution by making greedy changes to the current solution. Unlike greedy-iterative-improvement algorithms, in which a change is accepted only if it results in an improvement, simulated-annealing algorithms also accept changes that result in degradation. In this way, solutions can avoid being trapped in the local minima. In simulated annealing, a change that results in degradation is accepted as the new solution with a probability that decreases as the computation proceeds. Simulated-annealing algorithms conduct Boltzmann trials between solutions before and after changes. Changes are accepted with probability $1/(1+\exp((N-P)/T))$, where T is the global temperature, N is the cost of the old solution, and P is the cost of the modified solution. Similar to the natural annealing process, T is the temperature which determines the annealing schedule. T is initially high, allowing some inferior changes to be initially accepted. This helps the algorithm to escape the local minima. As T decreases slowly, the probability of accepting inferior changes drops too. This allows the algorithm to converge to good solutions in the neighborhood.

Starting from an initial schedule, we perturb the priority assignment of each event. For every event, we generate a random number δ and use the inverse of $ALAP$_start $+ \delta$ as the new priority assignment. Then, based on the new priority assignment, we regenerate the execution order of the scheduled events on every PE and link using the list-scheduling algorithm mentioned in the beginning of Section IV. We then generate the new variable-voltage schedule through a slack-allocation algorithm, which is discussed in Section IV-C, and re-evaluate the new power consumption after voltage scaling. A better solution (i.e., with lower power consumption) will always be accepted. An inferior solution will be accepted with a probability $1/(1+\exp((\text{new\_power}-\text{power})/T))$, where T is the annealing temperature and new_power and power are the power consumptions after and before the perturbations, respectively, of $ALAP$_start. The overall framework for execution-order optimization is illustrated in Fig. 5. Note that in the step in which the new power consumption is evaluated, the slack-allocation algorithm discussed in Section IV-C is called to generate the variable-voltage schedule. Under a given temperature, the perturbations stop if the number of consecutive perturbations that was not accepted, or the total number of perturbations, has exceeded some predefined threshold.

C. Power-Profile and Timing-Constraint Driven Slack Allocation and Variable-Voltage Scaling

This section discusses power-profile and timing-constraint driven slack allocation for a given execution order to maximize power reduction using an iterative approach based on Theorem 1. The slack-allocation algorithm is presented as Algorithm 1. It addresses how to evaluate the validity of the schedule when multiple tasks need to update their execution times (due to voltage scaling) as well as to annul any invalid updates in linear time ($O(n+k)$), where n is the number of tasks and k is the number of inter-PE communication edges, as mentioned in Section II.

Algorithm 1 $power\_profile\_slack\_allocation(G(V,E))$

1) initialize $v_l[i] = V_{\max}^{\text{PE}_i}, \forall tasks\ i$
2) initialize task list T_s containing all the tasks on voltage-scalable PEs
3) sort T_s in the order of decreasing energy gradients

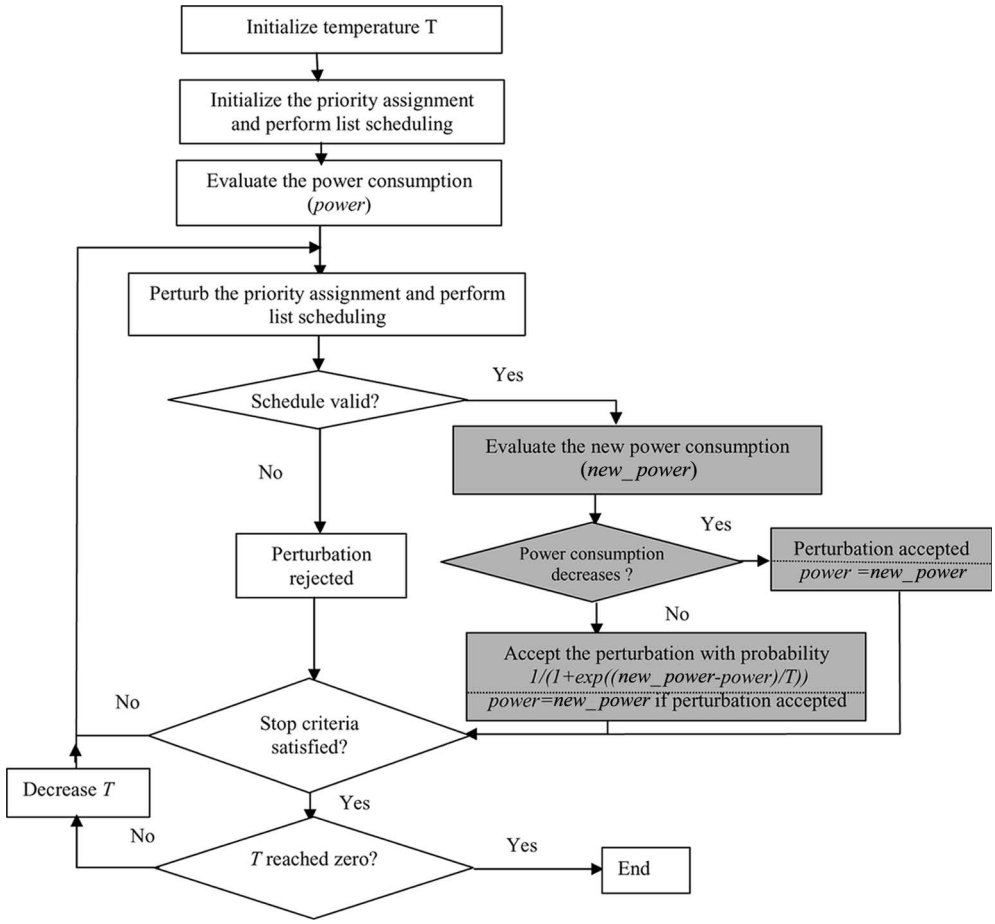

Fig. 5. Framework for execution-order optimization of scheduled events.

4) initialize active task list A_s with the set of tasks on voltage-scalable PEs with the highest energy gradient
5) **while** A_s is not empty or T_s is not empty **do**
6) **for** each event k in the order of $top\_sort(G(V,E))$ **do**
7) compute $ASAP$_start$_k$ of event k based on $v_l[i], \forall i$
8) **end for**
9) $v_l^{\text{old}} = v_l$
10) $j =$ the element in A_s with the highest voltage level
11) $v_l[j] -= dv$
12) $reference\_gradient = G(v_l[j], N_j)$
13) **for** each task i in A_s **do**
14) $v_l[i] = \min_V |G(V, N_I) \geq reference\_gradient$
15) **end for**
16) **for** each event k in the order of $reverse\_top\_sort(G(V,E))$ **do**
17) compute $ALAP$_finish$_k$ of event k based on $v_l[i], \forall i$
18) **if** ($ALAP$_finish$_k -$ wcet$_k(v_l[k]) < ASAP$_start$_k$) **then**
19) k marked as nonextensible
20) $v_l[k] = v_l^{\text{old}}[k]$
21) **end if**
22) **end for**
23) delete any task i from A_s, if it is marked as nonextensible, or if $v_l[i]$ has reached $V_{\min}^{\text{PE}_i}$
24) **if** A_s is empty **then**
25) append A_s with tasks with the highest energy gradient from T_s
26) **else**
27) append A_s with those tasks from T_s whose energy gradients become higher than the $reference\_gradient$
28) **end if**
29) **end while**
30) return v_l

Algorithm 1 is based on an augmented directed graph $G(V, E)$. After a valid schedule is generated through list scheduling and the order of scheduled events is determined, $G(V, E)$ can be created based on the task graphs as well as the constraints imposed by resource sharing and bus contention after scheduling. In $G(V, E)$, V is the set of vertices containing all the scheduled events in the initial schedule and E is the set of directed edges between vertices. An edge is inserted from one event to another if one is a direct predecessor of another in the task graphs or if one is scheduled just ahead of another on the same PE or link. Therefore, these edges represent all the precedence relationships in the original task graphs as well as execution ordering information in the initial schedule. The creation of $G(V, E)$ can be illustrated through the following example.

Example 1: Consider the embedded-system specification given in the form of two task graphs in Fig. 1. For the feasible schedule shown in Fig. 6, which is generated based on $ALAP$_start-based task priority assignment, the derived directed $G(V, E)$ is shown in Fig. 7(a). In Fig. 7(a), the dependencies introduced by execution ordering are represented by dotted arrows. One path with zero slack in the augmented graph, ($T1, T2, E1, T3, T6,$ and $T7$), is represented

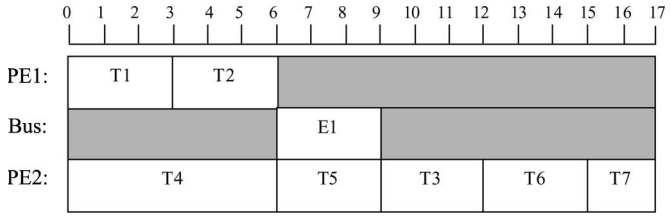

Fig. 6. Initial schedule using $ALAP$_start-based priority assignment.

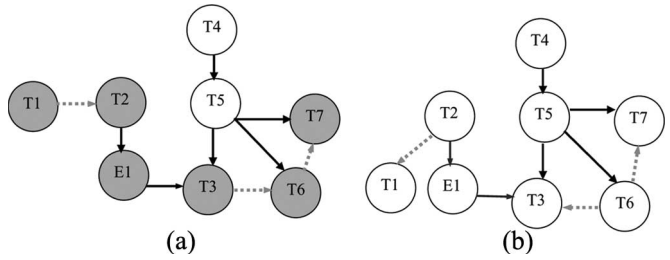

Fig. 7. Augmented directed graph $G(V, E)$. (a) $G(V, E)$ for initial $ALAP$_start based priority assignment. (b) $G(V, E)^{\text{opt}}$ after execution-order optimization.

by shaded nodes. The distributed system consists of two PEs, PE1 and PE2, connected by a bus. The schedule is based on worst case task execution and communication times, assuming a supply voltage of 1.8 V. We assume that all PEs have communication buffers.

For any scheduled event, its earliest possible start time ($ASAP$_start) and latest possible finish time ($ALAP$_finish) are calculated based on the augmented $G(V, E)$. Note that in $G(V, E)$, the node can be either a task or a communication event and the edge's execution time is zero. Based on $G(V, E)$, precedence relationships introduced by task execution ordering are also taken into consideration.

In Algorithm 1, v_l is an array of voltage levels for all the tasks. All the task voltage levels are initialized to $V_{\max}^{\text{PE}_i}$, where PE_i is the PE to which task i is assigned and $V_{\max}^{\text{PE}_i}$ is the maximum supply voltage on PE_i. The task list is initialized with all the tasks on voltage-scalable PEs in the order of decreasing energy gradients. The active task list is initialized by appending tasks with the highest energy gradient from the task list. Whenever a task is added to the active task list, it is deleted from the task list. In each iteration step, the voltage level of the task in the active task list with the highest voltage level is decreased by dv and its new energy gradient is chosen as the reference level. The update of voltage levels of other tasks in the active task list is done such that their energy gradients do not drop below the reference level, since the rate of decrease of their energy gradients is higher, as discussed in Section III-A. In Algorithm 1, $ASAP\_\text{start}_k$ ($ALAP\_\text{finish}_k$) is computed in the order of topological sort (reverse topological sort) of the augmented graph $G(V, E)$ with the wcets based on $v_l[i], \forall i$. During each iteration, if for a scheduled event, $ALAP\_\text{finish}_k - \text{wcet}_k(v_l[k]) < ASAP\_\text{start}_k$ is true, the update of its voltage level gets invalidated and is switched back to the old value in v_l^{old}. Such a task is marked as nonextensible. All the tasks, which are marked nonextensible or have reached the minimum voltage level, are deleted from the active task list at the end of the iteration. The active task list is then updated in the following way. Whenever the active task list is empty, it is appended with tasks with the highest energy gradient from the task list. Otherwise, it is appended with those tasks from the task list whose energy gradients become higher than the reference level. The iteration stops when both the task list and active task list become empty.

The overall complexity of Algorithm 1 is $O((n + k)\log(n + k) + M(n + k))$, where M is the number of iteration steps. M is dependent

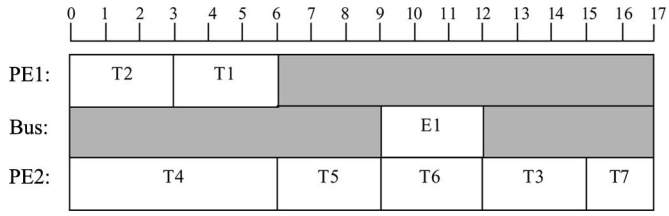

Fig. 8. New schedule after execution-order optimization.

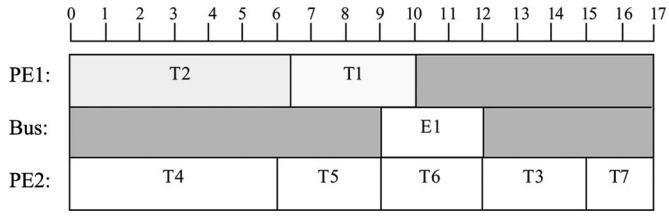

Fig. 9. Variable-voltage schedule for the new schedule.

on the distribution of switching activities of tasks in the system. Suppose $L = (V_{\max} - V_{\min})/dv$ is the number of voltage levels. In the case of $V_{\max} = 1.8$ V, $V_{\min} = 0.75$ V and $V_t = 0.6$ V, M should be of the same order as L when the variations of switching activities among tasks are in a normal range (i.e., the ratio of maximum to minimum switching activities does not exceed 100). When the power profile of the tasks is uniform, we have $M = L$. Therefore, our algorithm is also applicable to the case when the variations in power consumption of different tasks can be ignored. It is worth mentioning that Algorithm 1 can handle a fully heterogeneous system, in which different voltage-scalable PEs may have different $V_{\max}$ and $V_{\min}$. The complexity of the overall scheduling algorithm is the complexity of Algorithm 1 plus the complexity of $ALAP$_start-based list scheduling, which is $O(n(n + k)\log n)$.

D. Illustrative Example

The execution-order optimization scheme and the slack-allocation algorithm are illustrated through the following example.

Example 2: In Example 1, for the initial schedule shown in Fig. 6, no voltage scaling can be performed on any task since all the scheduled events are on paths with zero slack, as shown in the augmented graph in Fig. 7(a). After we apply the execution-order optimization algorithm from Fig. 5, the new schedule is obtained, as shown in Fig. 8. The shaded zero-slack path in the original $G(V, E)$ now gets broken after the above interchanges, as shown in the optimized $G(V, E)^{\text{opt}}$ in Fig. 7(b). In the new schedule, more flexibility is introduced in the system, which can lead to a more efficient voltage scaling.

We use Algorithm 1 to perform power-profile and timing-constraint driven slack allocation on the schedule in Fig. 8. The corresponding variable-voltage schedule is shown in Fig. 9, in which the execution times of the tasks are extended according to their new execution clock frequencies. Suppose the switching activities of tasks $T1$ and $T2$ are 1 and 3, and their new supply voltages are 1.3 and 1.65 V, respectively. The power consumptions of PE1 for the schedules in Figs. 6 and 9 are $2.287C$ and $1.375C$ W, respectively, where C is some constant. The improvement of power consumption on PE1 after execution-order optimization is, therefore, 39.9%.

V. Experimental Results

This section presents experimental results to verify the efficacy of the proposed algorithms. The experiment is performed on 16 embedded systems represented by task graph sets $TG1$–$TG16$. $TG1$–$TG3$

TABLE I
EFFICIENCY OF POWER-PROFILE DRIVEN SLACK ALLOCATION

Task graphs	#tasks/ #inter-PE edges	#PEs/ #links	*No-voltage-scaling* power(W)	*Single-task-extension* [16]		Our approach	
				#iterations	power(W)	#iterations	power(W)
TG1	88/42	2/1	2.078	3579	0.666	81	0.666
TG2	69/12	2/1	0.788	1488	0.334	30	0.334
TG3	12/5	3/2	1.470	141	0.705	24	0.705
TG4	120/27	4/1	3.521	1101	2.704	34	2.704
TG5	189/221	4/3	3.722	2301	2.648	33	2.648
TG6	361/52	2/1	1.846	4079	1.366	42	1.366
TG7	34/21	3/1	2.266	662	1.237	47	1.237
TG8	57/28	3/1	1.580	1044	0.822	55	0.822
TG9	98/78	3/3	0.872	872	0.742	21	0.743
TG10	85/39	2/1	0.678	441	0.615	21	0.616
TG11	111/72	2/2	0.658	867	0.577	21	0.577
TG12	59/61	4/4	0.958	394	0.797	21	0.797
TG13	40/23	2/2	0.232	267	0.107	21	0.107
TG14	51/37	2/4	0.452	352	0.307	21	0.307
TG15	36/0	1/0	0.317	243	0.248	21	0.248
TG16	120/83	4/4	1.193	1001	0.889	21	0.888

[24] are based on telecom, automotive, and consumer applications. $TG4$ is based on a DSP example in [21]. $TG5$ and $TG6$ [22] are based on LU decomposition of a large matrix and mean value analysis, respectively. $TG7$–$TG16$ are obtained using a randomized task-graph generator [23]. In this paper, voltage-scalable PEs are allowed to have different $V_{\max}$ and $V_{\min}$. We assume that the worst case transition-time overhead for voltage and frequency change is 20 μs. We compare this paper against the work in [16]. Note that the work in [14] has presented a polynomial-time optimal solution for slack allocation for the case of continuous voltage ranges based on a linear programming formulation. For discrete voltage levels, the ILP formulation in [14] is polynomial-time solvable only under special conditions. Since our experimental results are based on discrete voltage levels, we are thus unable to compare this paper with the work in [14].

A. *Efficiency and Complexity of Iterative Power-Profile Driven Slack Allocation*

We compare the slack-allocation algorithm presented in Algorithm 1 with two other approaches in Table I for $TG1$–$TG16$. One approach is single-task-extension in [16], and the other approach is no-voltage-scaling, in which tasks are executed under maximum supply voltages. For $TG1$–$TG8$, the switching activities of tasks range from 0.10 to 3.8. For $TG9$–$TG16$, we assume a uniform distribution of task switching activities. Iterative slack allocation is difficult to avoid, if variations in the power consumption of different tasks need to be addressed. In each iteration step, the schedule validity needs to be checked, which requires linear time complexity $(O(n+k))$. Therefore, the number of iterations can represent the scheduler run-time. In Schmitz's approach [16], a minimum extension time δt is allocated to the candidate task at each step which leads to maximum energy reduction. The determination of δt, which is empirical in their approach, is critical for achieving a tradeoff between the performance and run-time of their algorithm. For a fair comparison, we assume a similar approach that uses a minimum voltage drop dv, as in our algorithm, instead of a minimum extension time δt. The results for this approach are shown in column single-task-extension in Table I. The quality of our results is comparable with a much smaller run-time complexity, as shown in the table. Compared to their approach, Algorithm 1 can judiciously update the execution times of multiple tasks at the same time when their energy gradients are at about the same level. In our algorithm, the number of slack-allocation iterations is $O(L)$, where L is the number of available voltage levels. However, in Schmitz's approach [16], the number of slack-allocation iterations is $O(n \cdot L)$, where n is the number of tasks in the system. This clearly indicates the advantage of our algorithm in terms of run-time complexity. All the comparisons are based on the same initial schedule, which is generated using $ALAP$_start-based list scheduling. We set a minimum voltage drop dv in Algorithm 1 to be 0.05 V, since reducing dv further only results in a minimal improvement in terms of power savings.

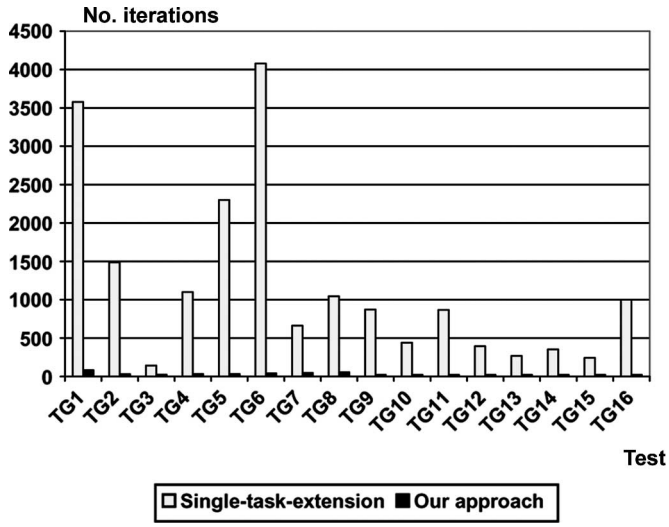

Fig. 10. Comparison of number of slack-allocation iterations between single-task-extension and our approach.

The power consumption for the initial schedule without variable-voltage scaling is also shown in column no-voltage-scaling in Table I. The CPU time range for our power-profile driven slack-allocation algorithm is 0.001–0.2 s, while it is 0.015–8 s for the single-task-extension scheme, on a Pentium-III 933-MHz PC with 256 MB of memory running under Linux. When the voltage/frequency transition-time overhead is incorporated, the slack time for a task is reduced by the overhead. The comparison of number of slack-allocation iterations between single-task-extension and our approach is illustrated in Fig. 10.

B. *Effect of Execution-Order Optimization on Variable-Voltage Scaling*

In Table II, we compare the results of power consumption for the schedules generated by $ALAP$_start-based list scheduling,

TABLE II
POWER CONSUMPTION FOR DIFFERENT EXECUTION ORDERS

Task graphs	$ALAP\_start$ based schedule(W)	$ASAP\_start$ based schedule(W)	Execution order optimized schedule(W)
TG1	0.666	0.764	0.666
TG2	0.334	0.332	0.331
TG3	0.705	0.898	0.705
TG4	2.704	2.743	2.663
TG5	2.648	-	2.648
TG6	1.366	1.366	1.273
TG7	1.237	1.149	1.037
TG8	0.822	-	0.677
TG9	0.743	0.758	0.666
TG10	0.616	-	0.559
TG11	0.577	0.518	0.506
TG12	0.797	0.758	0.601
TG13	0.107	0.099	0.090
TG14	0.307	0.280	0.264
TG15	0.248	0.300	0.221
TG16	0.888	0.962	0.815

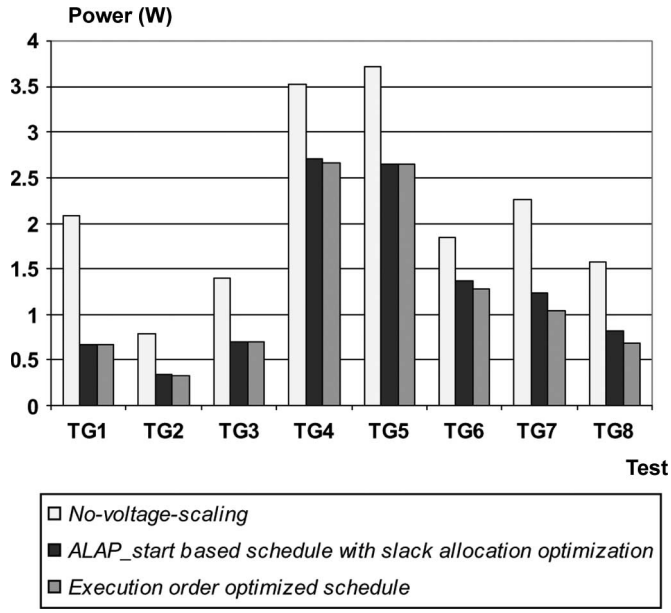

Fig. 11. Comparison of power consumption for different approaches.

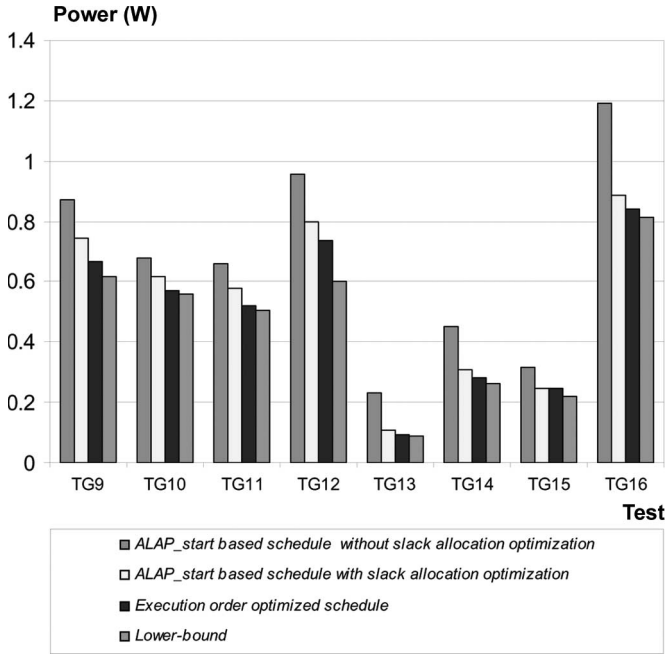

Fig. 12. Comparison against lower bound and no slack-allocation optimization.

$ASAP_$start-based list scheduling, and the schedule after execution-order optimization through simulated annealing for the 16 task-graph sets $TG1$–$TG16$. Algorithm 1 is used to perform slack-allocation optimization for these schedules. A "-" entry in the table indicates that the corresponding priority assignment fails to generate a valid schedule. One can see that execution-order optimization leads to the best quality variable-voltage schedule in terms of power consumption in all the cases. The execution-order-optimized schedule achieves an average (maximum) power reduction of 6.5% (18%) over $ALAP_$start-based schedule with power-profile driven slack allocation, and $ALAP_$start-based schedule with power-profile driven slack allocation achieves an average (maximum) power reduction of 33% (68%) over the no-voltage-scaling scheme. It has similar advantages over the $ASAP_$start-based schedules. The overall CPU time for execution-order-optimized schedule ranges from 0.03 to 40 s on a Pentium-III 933-MHz PC with 256 MB of memory running under Linux. If we set dv to 0.02 V, the results show only a negligible improvement. This justifies the advantage of using a minimum voltage drop, which can provide a good tradeoff between performance and complexity for the continuous voltage-scaling scenario.

The power consumption results for no-voltage-scaling, $ALAP_$start-based schedule optimized by Algorithm 1, and execution-order-optimized schedule, are illustrated in Figs. 11 and 12.

C. Comparison Against Lower Bound

To further illustrate the performance of our approach, we compare the schedules generated by our scheduling algorithms with a lower bound. The comparison is performed for task-graph sets $TG9$–$TG16$ and is shown in Fig. 12. For these task graphs, we assume a uniform distribution of task switching activities, allowing a lower bound on power consumption after variable-voltage scaling to be calculated. A lower bound on power consumption with the given task/communication assignment is computed based on the constraints that the execution time of a task i is restricted in the range $[ASAP\_\text{start}_i, ALAP\_\text{start}_i + \text{wcet}_i]$. Therefore, no resource constraints and precedence relationships are considered after voltage scaling to compute this lower bound. Note that the way we evaluate

the lower bound on power consumption only works for the case when the switching activities are the same for all the tasks. This is why we have selected a subset of examples for this paper. In all the eight test cases, the average (minimum) deviation of the execution-order-optimized schedule from the lower bound power consumption is only 7.6% (2.1%). Note that the lower bound is a loose one, since it is based on assumptions that are not valid (e.g., no resource constraints and precedence relationships exist after voltage scaling).

VI. Conclusion

We proposed an efficient variable-voltage scheduling algorithm, which can address variations in power consumptions, characteristics of different voltage-scalable PEs, precedence relationships, and hard deadlines of different tasks. It shows an order-of-magnitude run-time improvement over the previous works and can achieve close to optimal power savings. The scheduling algorithm is based on execution-order optimization of scheduling events, as well as slack budgeting motivated by the fact that the per-cycle energy consumption is normally a convex function of the processor clock period. Execution-order optimization can introduce more flexibility and increase the slack available to the system. The slack-budgeting algorithm can effectively distribute the slack among scheduling events to facilitate power saving through voltage scaling.

References

[1] W. H. Wolf, "Hardware–software co-design of embedded systems," *Proc. IEEE*, vol. 82, no. 7, pp. 967–989, Jul. 1994.

[2] R. P. Dick and N. K. Jha, "MOGAC: A multiobjective genetic algorithm for hardware–software co-synthesis of hierarchical heterogeneous distributed embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 10, pp. 920–935, Oct. 1998.

[3] *Intel Xscale*. [Online]. Available: http://www.intel.com/design/intelxscale/

[4] *Transmeta Crusoe*. [Online]. Available: http://www.transmeta.com

[5] *AMD PowerNow!* [Online]. Available: http://www.amd.com

[6] N. K. Jha, "Low power system scheduling and synthesis," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2001, pp. 259–263.

[7] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1702–1714, Dec. 1999.

[8] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. Symp. Foundations Comput. Sci.*, Oct. 1995, pp. 374–382.

[9] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proc. Des. Autom. Conf.*, Jun. 2001, pp. 828–833.

[10] J. Pouwelse, K. Langendoen, and H. Sips, "Energy priority scheduling for variable voltage processors," in *Proc. Int. Symp. Low-Power Electron. and Des.*, Aug. 2001, pp. 26–31.

[11] A. Manzak and C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy," in *Proc. Int. Symp. Low Power Electron. and Des.*, Aug. 2001, pp. 279–282.

[12] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proc. Int. Symp. Low Power Electron. and Des.*, Aug. 1998, pp. 76–81.

[13] N. Bambha, S. S. Bhattacharyya, J. Teich, and E. Zitzler, "Hybrid search strategies for dynamic voltage scaling in embedded multiprocessors," in *Proc. Int. Workshop Hardware/Software Co-Design*, Apr. 2001, pp. 243–248.

[14] Y. Zhang, X. Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. Des. Autom. Conf.*, Jun. 2002, pp. 183–188.

[15] F. Gruian and K. Kuchcinski, "LEneS: Task-scheduling for low-energy systems using variable voltage processors," in *Proc. Asian South Pacific Des. Autom. Conf.*, Jan. 2001, pp. 449–455.

[16] M. T. Schmitz and B. M. Al-Hashimi, "Considering power variations of DVS processing elements for energy minimisation in distributed systems," in *Proc. Int. Symp. Syst. Synthesis*, Oct. 2001, pp. 250–255.

[17] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 4, pp. 437–445, Dec. 1994.

[18] M. Pedram and Q. Wu, "Design considerations for battery-powered electronics," in *Proc. Des. Autom. Conf.*, Jun. 1999, pp. 861–866.

[19] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[20] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors," *Inf. Process. Lett.*, vol. 12, no. 1, pp. 9–12, Feb. 1981.

[21] C. M. Woodside and G. G. Monforton, "Fast allocation of processes in distributed and parallel systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 164–174, Feb. 1993.

[22] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 506–521, May 1996.

[23] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. Int. Workshop Hardware/Software Code.*, Mar. 1998, pp. 97–101.

[24] *E3S: The Embedded System Synthesis Benchmark Suite*. [Online]. Available: http://www.ece.northwestern.edu/~dickrp/e3s

[25] V. Gutnik and A. P. Chandrakasan, "Embedded power supply for low-power DSP," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 5, no. 4, pp. 425–435, Dec. 1997.

[26] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *Proc. ACM Symp. Operating Syst. Principles*, Oct. 2003, pp. 149–163.

[27] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," in *Proc. ACM SIGMETRICS*, Jun. 2001, pp. 50–61.

[28] P. B. Jorgensen and J. Madsen, "Critical path driven cosynthesis for heterogeneous target architectures," in *Proc. Int. Workshop Hardware/Software Code.*, Mar. 1997, pp. 15–19.

[29] P. Eles, A. Doboli, P. Pop, and Z. Peng, "Scheduling with bus access optimization for distributed embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 5, pp. 472–491, Oct. 2000.

[30] G. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

[31] J. Liu, P. H. Chou, and N. Bagherzadeh, "Communication speed selection for embedded systems with networked voltage-scalable processors," in *Proc. Int. Workshop Hardware/Software Code.*, May 2002, pp. 169–174.