Optimal Integration of Inter-Task and Intra-Task Dynamic Voltage Scaling Techniques for Hard Real-Time Applications

Jaewon Seo
KAIST, Daejeon, KOREA
jwseo@jupiter.kaist.ac.kr

Taewhan Kim
Seoul National University, Seoul, KOREA
tkim@ssl.snu.ac.kr

Nikil D. Dutt
University of California, Irvine, CA
dutt@ics.uci.edu

***Abstract*— It is generally accepted that the dynamic voltage scaling (DVS) is one of the most effective techniques for energy minimization. According to the granularity of units to which voltage scaling is applied, the DVS problem can be divided into two subproblems: (i) inter-task DVS problem and (ii) intra-task DVS problem. A lot of effective DVS techniques have addressed either one of the two subproblems, but none of them have attempted to solve both simultaneously, which is mainly due to an excessive computation complexity to solve it optimally. This work addresses this core issue, that is, *Can the combined problem be solved effectively and efficiently?* More specfically, our work shows, for a set of inter-dependednt tasks, that the combined DVS problem can be solved optimally in polynomial time. Experimental results indicate that the proposed integrated DVS technique is able to reduce energy consumption by 10.6% on average over the results by [11]+ [7] (i.e., a straightforward combination of two optimal inter- and intra-task DVS techniques.)**

I. Introduction

Over the past decades there have been enormous efforts to minimize the energy consumption of CMOS circuit systems. Dynamic voltage scaling (DVS) – involving dynamic adjustments of the supply voltage and the corresponding operating clock frequency – has emerged as one of the most effective energy minimization techniques. A one-to-one correspondence between the supply voltage and the clock frequency in CMOS circuits imposes an inherent constraint to DVS techniques to ensure that voltage adjustments do not violate the target system's timing constraints.

Many previous works have focused on hard real-time systems with multiple tasks. Their primary concern is to assign a proper operating voltage to each task while satisfying the task's timing constraint. In these techniques, determination of the voltage is carried out on a *task-by-task* basis and the voltage assigned to the task is unchanged during the whole execution of the task (**Inter-Task DVS**). Yao *et al.* [10] proposed an optimal inter-task voltage scaling algorithm for independent tasks, which determines the execution speed of each task at any given time so that the total energy consumption is minimized. Although their formulation does not impose any constraint on the number of operating voltages assigned to each task, by convexity of the power function, every task is given only one voltage and executed at a constant speed. This result comes from the underlying assumption that the required number of cycles to finish the task is constant, which is unacceptable in practice. With the same assumption, many works in the literature have tried to formulate their own inter-task DVS problems considering other issues such as dependent task sets [1], [3], [6], [9], [11], discretely variable voltage processors [3], [4], multi-processor environments [1], [3], [6], [9], [11], voltage transition overheads [1], etc.

In the past few years, several studies (e.g., [7], [8]) added a new dimension to the voltage scaling problem, by considering energy saving opportunities within the task boundary. In their approach, the operating voltage of the task is dynamically adjusted according to the execution behavior (**Intra-Task DVS**). Shin *et al.* [8] proposed the remaining worst-case path-based algorithm which achieves the best granularity by executing each basic block with possibly different operating voltage. To obtain tight operating points for minimum energy consumption, the algorithm updates the remaining path length as soon as the execution deviates from the previous remaining worst-case path. More recently, a profile-based optimal intra-task voltage scaling technique was presented in [7]. It shows the best energy reduction by incorporating the task's execution profile into the calculation of the operating voltages. The algorithm is proved to be optimal in the sense that it achieves minimum average energy consumption when the task is executed repeatedly.

To the best of our knowledge, no work has attempted to solve the inter-task and intra-task DVS problems simultaneously so as not to miss the energy saving opportunities at both granularities. In this paper, we present the first work that solves this combined inter- and intra-task DVS problem to fully exploit the advantages of both techniques. We propose an optimal integrated approach based on two energy-optimal inter-task [10] and intra-task [7] DVS algorithms.

II. Definitions and Problem Formulation

We consider that an application consists of N communicating tasks, $T = \{\tau_1, \tau_2, \ldots, \tau_N\}$, of which relations are represented by a directed acyclic graph (DAG). Each node in the graph represents task and each arc between two nodes indicates control/data dependency for two tasks. Fig. 3(a) shows an example of this graph. If two tasks τ_i and τ_j are connected by arc (τ_i, τ_j), the execution of τ_i must be completed before the execution of τ_j. A deadline d_i might be assigned to task

0-7803-9254-X/05/$20.00 ©2005 IEEE.

τ_i in order to ensure correct functionality (e.g., τ_4 and τ_6 in Fig. 3(a)), which we call *local deadline*. In addition, to restrict all tasks' ending time we add a special node called *sink* to the graph. Every task of which execution is not constrained by any of the deadlines is connected to the sink. (e.g., τ_5 and τ_7 in Fig. 3(a)) Those tasks should be executed before the deadline of the sink, which we call *global deadline*. More detailed explanation on this task model can be found in the recent inter-task DVS works e.g., in [3], [6], [11].

Each task $\tau_i \in T$ is associated with the following parameters:

- r_i : the required number of CPU cycles to complete,
- s_i : the starting time,
- e_i : the ending time ($e_i \geq s_i$), and
- t_i : the execution time ($t_i = e_i - s_i$)

Note that r_i is given for each task τ_i, while the values of s_i, e_i and t_i are determined after the task is actually scheduled.

The amount of energy consumption during the execution of the task τ_i is expressed as:

$$E_i = C_{eff} \cdot {V_i}^2 \cdot r_i \tag{1}$$

where C_{eff} and V_i denote the effective charged capacitance and the voltage supplied to the task, respectively. The relationship between clock frequency f_{clk} and supply voltage V_{dd} in CMOS circuits is [4]:

$$f_{clk} \propto 1/T_d = \frac{\mu C_{ox}(W/L)(V_{dd} - V_{th})^2}{C_L V_{dd}} \approx V_{dd} \tag{2}$$

where T_d is the delay, C_L is the total node capacitance, μ is the mobility, C_{ox} is the oxide capacitance, V_{th} is the threshold voltage and W/L is the width to length ratio of transistors. Based on the above one-to-one correspondence, we use *voltage* and corresponding *speed* (i.e., clock frequency) interchangeably throughout the paper and rewrite *Eq.(1)* as:

$$E_i = K \cdot {f_i}^2 \cdot r_i \tag{3}$$

where K is the system-dependent constant and f_i is the operational speed of the task τ_i. Although this equation seems somewhat oversimplified, our method is generally applicable if the energy consumption is represented as a power function of the speed i.e., $E_i = a f_i^b$. For the most of the commercial DVS processors it gives a good approximation of the actual energy consumption with an admissible error. (e.g., less than 3.4% error for Transmeta TM5900 1000MHz [12])

Since in most cases a task shows different behaviors depending on input data, i.e. r_i is not given as a constant, we cannot directly apply the simple energy equation, *Eq.(3)*. One way to deal with the different execution paths is to consider average (i.e., expected) energy consumption, which can be expressed as:

$$\overline{E}_i = \sum_{\forall \text{exec. path } \pi \text{ of } \tau_i} \big\{ Prob(\pi) \cdot E(\pi) \big\} \tag{4}$$

where $Prob(\pi)$ is the probability that the execution of task τ_i follows path π and $E(\pi)$ is the energy consumption on that path. The fact that many tasks in real-time environments are executed repeatedly provides the rationale for the use of this model. The overall energy consumption for the entire task set T is given as:

$$E_{tot} = \sum_{i=1}^{N} \overline{E}_i \tag{5}$$

We define a *feasible schedule* of tasks to be a schedule in which all the timing constraints of the tasks are satisfied. Note that the inter-task DVS problem is to find a (static) schedule of each task and a voltage applied to the task to minimize the amount of energy consumption. On the other hand, the intra-task DVS problem is to find a (dynamic) voltage-adjustment scheme for each basic block in the task to minimize the amount of (average) energy consumption while satisfing the deadline constraint of the task. Then, the combined problem of the inter- and intra-task DVS problems can be described as:

Problem 1 (The Combined DVS Problem) *Given an instance of tasks, find an inter-task schedule and an intra-task voltage scaling scheme that produces a feasible schedule for every possible execution path of tasks and minimizes the quantity of E_{tot} in Eq.(5).*

III. The Combined DVS Technique

The proposed integrated DVS approach is a two-step method:

1) Statically determine energy-optimal starting and ending times (s_i and e_i) for each task τ_i considering future use of an intra-task voltage scaling.
2) Execute τ_i within $[s_i, e_i]$ while varying the processor speed according to the voltage scales obtained by an existing optimal intra-task DVS scheme.

Our key concern is to develop a new inter-task scheduling algorithm that finds starting and ending times (s_i and e_i) for each task, which leads to a minimum value of E_{tot} in *Eq.*(5) when an optimal intra-task scheme is applied to the tasks. (Note that the new inter-task scheduling algorithm is different from any existing inter-task DVS scheme, which is not aware of intra-task DVS at all.)

In the following, let us first introduce an optimal intra-task DVS technique [7] since it provides a basis on the development of our combined enegy-optimal DVS technique.

A. An optimal intra-task DVS technique

A task τ is represented with its CFG (Control Flow Graph) $G_\tau = (V, A)$, where V is the set of basic blocks in the task and A is the set of directed edges which impose precedence relations between basic blocks. (For example, see Fig. 1(a).) The set of immediate successor basic blocks of any $b_i \in V$ is denoted by $succ(b_i)$. Each basic block b_i is annotated with its non-zero number of execution cycles n_i and each arc (b_i, b_j) is given a probability p_j that the execution follows the arc.

Following theorem states the lower bound energy consumption that any intra-task DVS technique can achieve.

Theorem 1: *Given a task's CFG and its execution profile that offers the probabilities, the minimum energy consumption of*

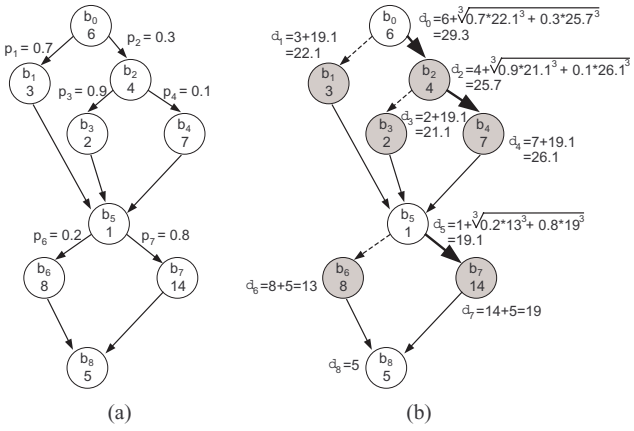

Fig. 1. (a) CFG of a task τ_{simple}; (b) Calculation of δ values.

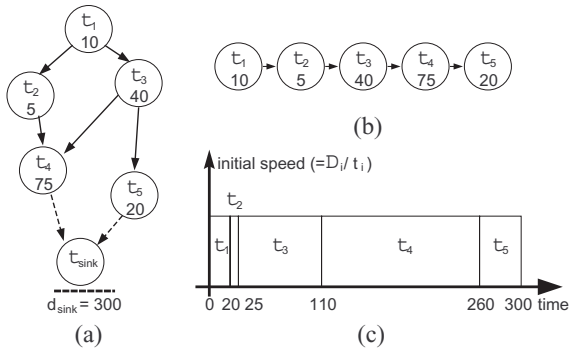

Fig. 2. (a) A simple task set T_{simple}; (b) A task order preserving dependencies; (c) Scheduled tasks.

any intra-task DVS technique is bounded by:

$$E_{intra} = K \cdot \left(\frac{\delta_0}{(relative)\ deadline}\right)^2 \cdot \delta_0 \qquad (6)$$

where δ_i is defined as:

$$\delta_i = \begin{cases} n_i, & \text{if } succ(b_i) = \emptyset \\ n_i + \sqrt[3]{\sum_{\forall b_j \in succ(b_i)} p_j \cdot {\delta_j}^3}, & \text{otherwise} \end{cases} \qquad (7)$$

and δ_0 is the δ value of the top basic block b_0. (See [7] for proof.)

One interesting interpretation of *Eq.(6)* is that it can be considered as the energy consumed in the execution of δ_0 cycles at the speed of $\delta_0/deadline$. We call δ_0 *energy-optimal path length* of the task.

The above theorem leads to the following optimal voltage scaling scheme [7]: *Before executing the task, we compute the δ_i value of each basic block b_i and insert the following instruction code at the beginning of b_i*

`change_f_V(`δ_i`/remaining_time());`

where the instruction `change_f_V(`f_{clk}`)` *changes the current clock frequency (i.e., speed) to f_{clk} and adjusts the supply voltage accordingly and* `remaining_time()` *returns the remaining time to deadline.*

For example, consider a real-time task τ_{simple} shown in Fig. 1(a). The number within each node indicates its number of execution cycles n_i. Fig. 1(b) shows the procedure of calculating each δ_i value according to *Eq.(7)*. Once we have finished the calculation, the operating speed of basic block b_i is simply obtained by dividing δ_i by the remaining time to deadline. Suppose that $deadline$ = 10 (unit time) and the execution follows the path $(b_0, b_2, b_3, b_5, b_6, b_8)$. Then, the corresponding speed changes as follows:

$$\text{Speed}: 2.93 > 3.24 > 3.14 > 3.14 > 2.26 > 2.26$$

In Fig. 1(b), the thick, dotted, and regular arrows indicate the increase, decrease and no change of the processor speed, respectively and the basic blocks with changed operating speed are marked with gray-color. More details including practical issues (e.g. handling loop constructs, voltage transition overheads, discretely variable voltages, etc.) can be found in [7].

B. An intra-task DVS-aware inter-task DVS technique

One characteristic of the previous intra-task DVS technique is that after an execution time (or relative deadline) is assigned, the scheduler finishes the task exactly at the deadline no matter which execution path is taken, producing no slack time. Thus, the next step is to determine the starting and ending times of each task. We first give a characterization of an energy-optimal schedule for any set of tasks that have no local deadlines.

Case 1 - Tasks with a global deadline only: For the case where each task has no local deadline and of which ending time is restricted only by the global deadline d_{sink} (For example, see Fig. 2(a)), the problem can be solved easily by properly distributing the permitted execution time (= d_{sink}) over the tasks and then determining the order of the tasks. Following lemma states the optimal distribution of d_{sink} which leads to the minimum energy consumption when the previous optimal intra-task scheme is applied.

Lemma 1: Given a task set $T = \{\tau_1, \tau_2, \ldots, \tau_N\}$ with no local deadlines and the global deadline d_{sink} the total energy consumption E_{tot} is minimized when each task is assigned the execution time proportional to its energy-optimal execution path length i.e., when t_i is given as:

$$t_i = \frac{\Delta_i}{\Delta_1 + \cdots + \Delta_N} \cdot d_{sink} \qquad (8)$$

where Δ_i denotes the δ_0 of *Eq.* (7) for τ_i.

Proof: For $N = 2$, if we set $t_1 = x$ and $t_2 = y$, the total energy consumption is expressed as:

$$K \cdot \left(\frac{\Delta_1}{x}\right)^2 \cdot \Delta_1 + K \cdot \left(\frac{\Delta_2}{y}\right)^2 \cdot \Delta_2 \qquad (9)$$

where $x + y = d_{sink}$. By substituting $d_{sink} - x$ for y and then differentiating with respect to x, it can be verified that *Eq.* (9) has the minimum $K \cdot \left(\frac{\Delta_1+\Delta_2}{d_{sink}}\right)^2 \cdot (\Delta_1+\Delta_2)$ where $x = d_{sink} \cdot \frac{\Delta_1}{\Delta_1+\Delta_2}$. For $N = m$, assume the lemma holds for $m-1$ tasks. Then the energy equation becomes essentially identical

to *Eq.* (9). (Δ_2 is replaced by $\Delta_2 + \cdots + \Delta_m$.) Therefore the lemma holds. ■

Now we know how much time should be given to each task for execution. Thus the next step is to determine the starting time (and automatically ending time) for each task, which is obtained directly from a task order. Since there is no local deadline for each task, any order that preserves all the dependencies between the tasks suffices the minimum energy schedule. Without loss of generality assume the sequence $(\tau_1, \tau_2, \ldots, \tau_N)$ does not violate the precedence relations. Then the starting and ending time of each task are determined simply as:

$$s_i = \begin{cases} 0, & \text{for } i = 0 \\ e_{i-1}, & \text{for } i \geq 1 \end{cases} \quad \text{and} \quad e_i = s_i + t_i \tag{10}$$

For example, suppose a simple task set $T_{simple} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$ of which DAG is shown in Fig. 2(a). Each task has no local deadline and there is only one global deadline d_{sink} = 300 unit time specified by the sink. The number inside each node indicates the δ_0 value of the corresponding task. Fig. 2(b) shows an order that preserves the dependency relations, from which we calculate the time stamps for each task as shown in Fig. 2(c) using *Eq.* (8) and *Eq.* (10)

Case 2 - Tasks with arbitrary deadlines: For the general case where tasks possibly have local deadlines as well as the global deadline, we divide the problem into a collection of subproblems of *Case 1* and then apply the result of *Case 1*.

Firstly, we order the tasks with the Earliest Deadline First (EDF) policy while preserving the dependency relations[1]. The EDF policy is proved to give the best opportunity for energy savings [11]. Let $(\tau_1, \tau_2, \ldots, \tau_N)$ be the obtained sequence and for the sake of illustration assume only two tasks τ_i and τ_j $(i < j)$ have local deadlines. Then we partition the task set into three task groups $\{\tau_1, \ldots, \tau_i\}$, $\{\tau_{i+1}, \ldots, \tau_j\}$, and $\{\tau_{j+1}, \ldots, \tau_N\}$ and apply the method in *Case 1* to each group separately, where we set the starting times of τ_1, τ_{i+1}, and τ_{j+1} to 0, d_i, and d_j, respectively.

For example, consider a task set shown in Fig. 3(a). We can order the tasks by the EDF policy as shown in Fig. 3(b). Since only two tasks τ_4 and τ_6 have local deadlines, the task set is divided into three groups $\{\tau_1, \tau_2, \tau_4\}$, $\{\tau_3, \tau_6\}$, and $\{\tau_5, \tau_7\}$ as shown in Fig. 3(c), and we set the starting time of each group 0, 40(= d_4) and 120(= d_6), respectively. Then the permitted execution times 40, 80(= 120 − 40), and 80(= 200 − 120) are distributed proportionally to the tasks in each group according to Lemma 1.

C. An improved inter-task DVS technique

It should be noted that the previous method of assigning a starting time to each task group does not guarantee to produce the minimum energy schedule. To overcome this limitation, we transform the original problem into an inter-task scheduling problem for independent tasks and use the well-known energy-optimal scheduling algorithm by Yao *et al.* [10].

[1]Every task of which deadline is not specified inherits the deadline of its earliest successor task.

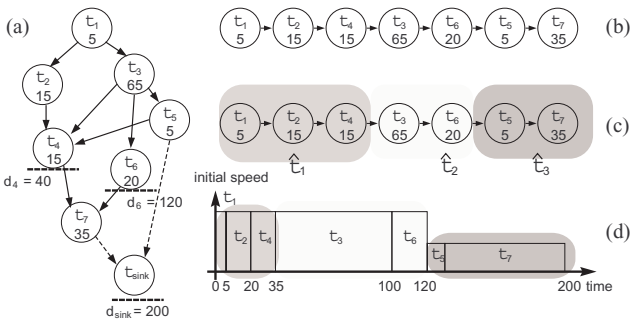

Fig. 3. (a) Tasks with local deadlines; (b) EDF order; (c) Dividing into task groups/supertasks; (d) Scheduled tasks.

In their task model, each task is specified by the triple of the arrival time, deadline, and the required number of CPU cycles to complete. The algorithm identifies a 'critical' interval where a set of tasks should be completed within that interval and no other tasks should be executed in the interval to minimize the energy consumption, and then schedules those 'critical' tasks at a maximum constant speed by the earliest deadline policy. The process is repeatedly applied to the subproblem for the remaining unscheduled tasks until there remains no unscheduled task. As a result we have a feasible schedule of which energy consumption:

$$E_{inter} = \sum_{i=1}^{N} \left\{ K \cdot \left(\frac{r_i}{t_i} \right)^2 \cdot r_i \right\} \tag{11}$$

is minimized.

Transforming the original problem: Each task group $\{\tau_1, \ldots, \tau_i\}$, $\{\tau_{i+1}, \ldots, \tau_j\}$, and $\{\tau_{j+1}, \ldots, \tau_N\}$ in the previous section is now considered as a single task, which we call *supertask* and denote $\hat{\tau}_1$, $\hat{\tau}_2$, and $\hat{\tau}_3$. Note that we use the hat symbol($\hat{}$) to differentiate supertasks from ordinary tasks in Yao *et al.*'s model. The arrival time $\hat{a}_i$ of each supertask is set to 0 and the deadline $\hat{d}_i$ is inherited from the original task group. One difference is that the required number of CPU cycles $\hat{r}_i$ for each supertask is not set to the sum of consisting tasks' r_i but to the sum of Δ_i, more specifically they are set as follows:

$$\hat{r}_1 = \Delta_1 + \cdots + \Delta_i$$
$$\hat{r}_2 = \Delta_{i+1} + \cdots + \Delta_j$$
$$\hat{r}_3 = \Delta_{j+1} + \cdots + \Delta_N$$

The resulting schedule after applying those supertasks to Yao *et al.*'s algorithm directly determines the starting and ending times of each supertask. We then continue the same procedure of the previous section.

The detailed description of the algorithm is shown in Fig. 4. The procedure Construct_Supertask$(1, T)$ constructs the set of supertasks recursively from the given task set T. The function $h(i)$ returns the index of the ith task that has local deadline in the EDF order and the $pred(\tau_i)$ returns the set of all immediate predecessor tasks of the task τ_i.

Fig. 3 shows an example. The original task set T is shown in Fig. 3(a). Fig. 3(b) represents the sequence obtained by the EDF ordering. Note that there are two tasks τ_4 and τ_6 that have local deadlines. (Therefore $h(1)$ = 4 and $h(2)$ = 6). We partition the task set into three task groups or equivalently three supertasks $\hat{\tau}_1$ = $\{\tau_1, \tau_2, \tau_4\}$, $\hat{\tau}_2$ = $\{\tau_3, \tau_6\}$, and $\hat{\tau}_3$ = $\{\tau_5, \tau_7\}$ as shown in Fig. 3(c). The arrival time, deadline, and the required number of cycles are set as follows:

$$\hat{a}_1 = \hat{a}_2 = \hat{a}_3 = 0$$
$$\hat{d}_1 = d_{h(1)} = 40$$
$$\hat{d}_2 = d_{h(2)} = 120$$
$$\hat{d}_3 = d_{h(3)} = 200$$
$$\hat{r}_1 = \Delta_1 + \Delta_2 + \Delta_4 = 35$$
$$\hat{r}_2 = \Delta_3 + \Delta_6 = 85$$
$$\hat{r}_3 = \Delta_5 + \Delta_7 = 40$$

Fig. 3(d) shows the resultant schedule of the supertasks using Yao *et al.*'s algorithm. Finally, each individual task in the supertask receives the execution time according to Lemma 1, which is depicted also in Fig. 3(d).

Theorem 2: DVS-intgr *produces a feasible and minimum energy schedule when followed by the optimal intra-task voltage scaling.*

Proof Sketch. Since every suppertask has zero arrival time, for any critical interval (with supertasks), each supertask has all its preceding (unscheduled) supertasks in the same interval. Thus the scheduling for the supertask set does not alter the EDF order, which leads to a feasible schedule. (See [10] for more details.) The obtained supertask schedule has the minimum value of the following energy equation:

$$E_{supertask} = \sum_{i=1}^{\hat{N}} \Big\{ K \cdot \Big(\frac{\sum \Delta_j}{\hat{t}_i} \Big)^2 \cdot \sum_{\forall \tau_j \in \hat{\tau}_i} \Delta_j \Big\} \qquad (12)$$

where $\hat{N}$ is the number of supertasks. Note that this equation is essentially identical to *Eq.* (11). It can be proved that the inner term of the summation in *Eq.* (12) represents the lower bound of the energy consumption that the consisting tasks can have within the interval of $\hat{t}_i$. Since each individual task is assigned the execution time proportional to its energy-optimal path length, by Lemma 1 the lower bound is achieved. Therefore the theorem holds.

IV. Experimental Results

We implemented the proposed integrated DVS technique, called DVS-intgr, in C++ and tested it on the task sets generated by TGFF v3.0 [2]. We used the example input files (*.tgffopt) provided with the package to obtain the task sets.

Each individual task was generated by inserting a left/right child and a grandchild into an arbitrarily chosen basic block repeatedly until the number of conditional branches reaches some fixed number in the range of [1, 100]. The probability at each branch is drawn from a random normal distribution with standard deviation of 1.0 and mean of 0.5. The number of execution cycles (i.e., length) of each basic block is restricted such that the length of the longest basic block does not exceed 100 times of the length of the shortest one. Then the overall lengths in each task are multiplied by a factor such that the worst-case execution of the corresponding supertask produces 20% slack time (within its bound) when the maximum speed is used.

```
DVS-intgr {
• Input : Set T of tasks
• Output : Tasks with starting/ending time specified
   T̂ = Construct_Supertask(1, T);
   Yao_Inter_Scheduler(T̂);
   for ∀τ̂_i ∈ T̂ {
      current_time = ŝ_i;
      while (τ̂_i ≠ ∅) {
         τ_j = Retrieve_First_Task(τ̂_i);
         s_j = current_time;      // schedule each task
         e_j = s_j + (Δ_j/r̂_i) · t̂_i;  //  in the supertask
         current_time = e_j;
}}}

Construct_Supertask(i, T) {
   if (T = ∅) return ∅;
   τ̂_i = {τ_h(i)};                    //
   â_i = 0;                           // initialize the ith
   d̂_i = d_h(i);                      //  supertask
   r̂_i = Δ_h(i);                      //
   T = T − {τ_h(i)};
   T1 = pred(τ_h(i));
   while (∃τ_j ∈ T1) {
      τ̂_i = τ̂_i ∪ {τ_j};              // add a task to the ith
      r̂_i = r̂_i + Δ_j;                //  supertask
      T = T − {τ_j};
      T1 = T1 − {τ_j} ∪ pred(τ_j);
   }
   return {τ̂_i}∪ Construct_Supertask(i + 1, T); }

   // h(i) returns index of ith task that has local deadline.
   // pred(τ_i) returns all immediate predecessors of τ_i
```

Fig. 4. A summary of the proposed algorithm.

Table I shows a summary of the experimental results. Our competitors include the inter-task DVS technique by Zhang [11] (Zhang-only) and the same technique followed by application of the optimal intra-task DVS technique (Zhang+Intra). Zhang *et al.* solved the inter-task DVS problem for dependent tasks using integer programming. Although their technique produces an optimal solution to the inter-task DVS problem, they did not consider future application of any intra-task DVS techniques, which inevitably leads to non-optimal energy consumption when combined with intra-task DVS techniques. To give the best chance to Zhang-only, it was assumed that when the task finishes earlier than the scheduled time, the target system becomes idle with the *power-down* mode consuming no energy. We repeated the experiment for each task set 1000 times and then obtained the average reduction of energy consumption after discarding higher and lower 100 results, respectively. The sixth and seventh columns in the table represent the average energy consumptions of Zhang+Intra and DVS-intgr respectively, normalized to the corresponding energy consumption of Zhang-only. From the sixth column, one can easily see that the follow-up application of the optimal intra-task DVS technique drastically reduces the total energy consumption. The eighth column shows the average energy reduction of our technique over Zhang+Intra for each

Task Set (Script) [2]	Number of Tasks	Number of Edges	No. of Tasks with Deadline	Normalized Energy Consumption (Avg.)			Avg. Reduction over Zhang + Intra
				Zhang-only	Zhang + Intra	**DVS-intgr**	
bus0	15	30	5	1	0.3840	0.3549	7.8%
bus1	15	30	5	1	0.2564	0.2321	9.4%
kbasic_task0	42	50	17	1	0.2760	0.2516	9.3%
kbasic_task1	64	79	23	1	0.3590	0.3286	9.1%
kbasic_task2	42	46	17	1	0.2617	0.2321	10.3%
kbasic_task3	36	47	12	1	0.3287	0.3007	8.7%
kbasic_task4	21	22	10	1	0.3259	0.2838	13.3%
kbasic_task5	21	23	7	1	0.2595	0.2345	11.6%
kbasic_task6	18	18	10	1	0.2405	0.2127	12.8%
kbasic_task7	32	36	15	1	0.3609	0.3234	8.8%
kbasic_task8	23	27	9	1	0.2856	0.2567	10.2%
kbasic_task9	35	40	15	1	0.3052	0.2888	6.3%
kextended0	23	25	8	1	0.3527	0.3184	10.0%
kextended1	21	28	6	1	0.3496	0.3148	11.7%
kextended2	22	27	6	1	0.3486	0.3141	9.9%
kseries_parallel0	30	33	4	1	0.3070	0.2718	12.2%
kseries_parallel1	20	19	4	1	0.2472	0.2175	11.4%
kseries_parallel2	62	61	9	1	0.3003	0.2770	8.0%
kseries_parallel3	47	46	6	1	0.3163	0.2871	10.2%
kseries_parallel_xover0	30	37	3	1	0.3057	0.2603	15.1%
kseries_parallel_xover1	21	24	4	1	0.2618	0.2422	8.2%
kseries_parallel_xover2	38	41	4	1	0.3458	0.3209	7.2%
kseries_parallel_xover3	27	30	4	1	0.2494	0.2128	15.3%
simple0	12	19	1	1	0.3691	0.3181	13.9%
simple1	20	25	6	1	0.2832	0.2563	9.7%
simple2	24	28	6	1	0.2897	0.2556	13.9%
simple3	11	12	3	1	0.2732	0.2337	14.2%
simple4	22	32	4	1	0.2713	0.2459	9.2%
Avg.				1	0.3041	0.2731	**10.6**%

TABLE I

A COMPARISON OF ENERGY CONSUMPTIONS FOR THE TASK SETS IN [2].

task set. (Note that this value does not come directly from the corresponding energy values in the sixth and seventh column since it is the average of reduction percentages.) In comparison, DVS-intgr reduces energy consumption by 72.7% and 10.6% on average compared to that of Zhang-only and Zhang + Intra, respectively.

V. CONCLUSIONS

We have presented a novel DVS technique to deal with the combined inter- and intra-task DVS problem. To solve this problem, first we examined the lower bound of the energy consumption that any intra-task DVS technique can achieve and then using this property we devised a method setting energy-optimal execution time to each task. To obtain the globally optimal task schedule we divided the task set into several task groups such that each task in the group can be scheduled optimally within the group boundary and then transformed the problem of determining the group boundary into an inter-task DVS problem for independent tasks, which is then solved using the existing optimal inter-task DVS technique. The experimental results showed that our proposed integrated DVS technique was able to reduce the energy consumption by 10.6% over that of the combination of the existing techniques of [11] followed by [7].

REFERENCES

[1] A. Andrei et al., "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," *Prof. of Design Automation and Test in Europe*, 2004.

[2] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: Task graphs for free," *Proc. of Sixth International Workshop on Hardware/Software Codesign*, Mar. 1998.

[3] B. Gorji-Ara et al., "Fast and efficient voltage scheduling by evolutionary slack distribution," *Proc. of Asia-South Pacific Design Automation Conference*, 2004.

[4] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," *Proc. of Design Automation Conference*, 2003.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, Vol. 20, Jan. 1973.

[6] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," *Prof. of Design Automation and Test in Europe* , 2002.

[7] J. Seo, T. Kim, and K. Chung, "Profile-based optimal intra-task voltage scheduling for hard real-time applications," *Proc. of Design Automation Conference*, 2004.

[8] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications, *IEEE Design and Test of Computers*, Vol. 18, 2001.

[9] G. Varatkar and R. Marculescu, "Communication-aware task scheduling and voltage selection for total systems energy minimization," *Proc. of International Conference on Computer-Aided Design*, 2003.

[10] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *Proc. of IEEE Symposium on Foundations of Computer Science*, 1995.

[11] Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," *Proc. of Design Automation Conference*, 2002.

[12] http://www.transmeta.com