Battery-aware Dynamic Voltage Scaling in Multiprocessor Embedded System

Yuan Cai, Sudhakar M.Reddy
Department of Electrical and Computer Engineering
University of Iowa
yucai@engineering.uiowa.edu
reddy@engineering.uiowa.edu

Irith Pomeranz
School of Electrical and Computer Engineering
Purdue University
pomeranz@ecn.purdue.edu

Bashir M Al-Hashimi
School of Electronics and Computer Science
University of Southampton
bmah@ecs.soton.ac.uk

***Abstract*— In a battery powered system, a primary design consideration is the battery lifetime. Profile of current drawn from a battery determines its lifetime. Recently in [4] dynamic voltage scaling has been applied to alter the battery load current profile in distributed systems to reduce battery charge consumption. Load current profile is changed by utilizing the slack in the execution of the scheduled tasks. In this paper we propose a new dynamic voltage scaling procedure that alters load current profile by considering the total battery current instead of the method of [4] that considers the current dawn by individual task with the latest finish times in the schedule. The task schedule is partitioned into steps defined in this work and the load currents during selected steps are targeted for reduction by scaling the supply voltage of the processing elements. Experimental results on a large set of task graphs show that battery charge consumption reductions of up to 89.80% are achieved by the new algorithm.**

I. Introduction

Battery-aware system design has become a new frontier in low power design in recent years[1]. Due to the continuously increasing functionality and performance, applications are consuming more and more energy while the capacity of the batteries grows relatively slowly. Since in battery-powered system, the battery lifetime impacts the utility and duration of the system directly, extending the battery lifetime should be a primary design metric.

In most cases, after task scheduling, there exists a slack between the deadline and the real finish time of the execution of the tasks in distributed embedded systems. Dynamic voltage scaling (DVS) is used to exploit the slack to reduce the supply voltage of the processing elements (PEs) so that their energy consumption can be reduced. The side effect is that the task execution time will be increased. The existing slack accommodates the extra execution time. This technique is very powerful because the energy consumption is proportional to the square of the supply voltage. As a result, many DVS algorithms have been developed recently. [2] gives a good summary.

Since voltage scaling changes the current drawn by a PE as well, DVS can also be used to extend the battery lifetime which heavily depends on the load current profile (LCP). LCP is the instantaneous battery current expressed as a function of time. In this work we approximate LCP as a stepcase function of time. Only a few works have investigated DVS for battery-aware system level synthesis. In [5] [6], DVS is used in a single PE system to reduce the battery charge consumption. In [7], task scheduling is battery-aware, but the DVS is not. In [4], which is briefly discussed next, DVS for multi-processor distributed systems is investigated. The algorithm in [4] considers scheduled tasks one by one, starting from the last scheduled task. The latest finished task is scaled to the lowest possible voltage subject to the deadline constraint. Then the task that is finished the last among the remaining tasks is scaled. This process is repeated until there is no slack left or every task uses the lowest allowed voltage level. In distributed systems, tasks are executed concurrently on multiple PEs. Therefore, the instantaneous load current of the battery is the sum of the currents consumed by all the tasks executed concurrently on the PEs in the system. Thus in distributed systems it is necessary to consider the total load current of the battery instead of currents drawn by individual tasks to derive a good DVS strategy. Based on this observation, we propose a new DVS algorithm to modify the total battery LCP to reduce the battery charge consumption.

The rest of the paper is organized as follows. Section II includes preliminaries and introduces the battery model used in this work. The proposed DVS algorithm is described in Section III. Experimental results are given in Section IV and conclusions are in Section V.

II. Preliminaries

In battery-aware system design, one of the most important items is battery modelling. Due to the complex non-linear electrochemical phenomena occurring inside the battery, it is difficult to model the battery behavior. Around 10 battery models exist currently and a full survey of battery modelling can be found in [8].

In this work we use an analytical high-level battery model proposed in [6] [9]. Two important battery properties, rate-capacity effect and recovery effect, are both well modelled. Work in [4] also used this battery model. The input to the model is the battery load current profile which is approximated by a stepcase function. The current and duration of $step_k$ is I_k and Δ_k respectively. The load current profile of tasks can be obtained by method introduced in [10]. The core part of the model is the following function

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

where N is the number of the total load current profile steps and β is a constant related to the non-linear property of the battery. The function $F(x, y, z, \beta)$ is defined as

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

The battery lifetime is computed by solving the equation

$$\alpha = \sum_{k=0}^{N-1} I_k F(L, t_k, t_k + \Delta_k, \beta)$$

where α is the battery capacity and L is the unknown. Details of the computation of L are given in [9].

This model is not limited as a tool to measure battery lifetime, more

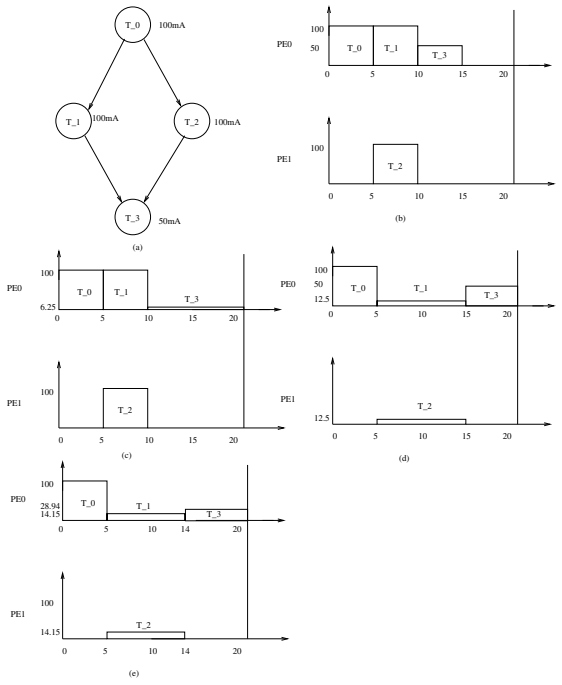

Fig. 1. A scheduled task graph.

important usage of this model given above is that equation (1) can be used as a cost function and an optimization objective in battery-aware system design. It has been analyzed in [6] that design targeting the energy minimization won't result in the longest battery lifetime. Only when this cost function is minimized, the battery lifetime will be maximized[6]. Hence, the proposed algorithm will focus on reducing this cost function. We do so by modifying the battery LCP through scaling the voltage of the selected tasks. A motivational example given in Section III illustrates how to utilize the voltage scaling to modify LCP. Throughout the paper, we use the same assumption as [4] that if the voltage of a task is scaled by s, then the worst case execution time ($wcet$) of the task will become by $wcet/s$ and the current of the task will be decreased by s^3. Note that $0 < s < 1$ because we assume the voltage is down-scaled. In the remainder of the paper, when we say the current of a task we mean the current drawn from the battery due to the execution of the task.

III. Proposed battery-aware DVS algorithm

We use an example to motivate the main ideas of the proposed algorithm.

Example 1: A task graph with 4 nodes is given in Figure 1(a). The number next to each node is the current of the task. The four tasks are executed on two PEs, as shown in Figure 1(b). The length of a rectangle in Figure 1(b) is the execution time of the task and its height is the current. In the given schedule, there is a slack of 5 time units. The supply voltage of both PEs are scalable with the highest supply voltage set at $3.3V$, and β of the battery model is 0.273 [9]. We assume that the communication times are all zero for convenience. We compare three DVS methods.

Method 1 This method is the one used in [4]. Scale the voltage of the last task, T_3, until T_3 finishes just at the deadline, that is, utilize all the slack to scale the task with the latest finish time. The scaled result is given in Figure 1(c).

We notice that T_1 and T_2 are executed concurrently, and as a result, the load current of the battery is very large during their executions. There are two possibilities for distributing the slack. These are illustrated by the two methods given next.

Method 2 We use all the slack to scale the two tasks T_1 and T_2 simultaneously until the finish time of T_3 reaches the deadline. The scaled schedule is given in Figure 1(d).

Method 3 Scale the voltages of T_1 and T_2 simultaneously until their $wcet$ is prolonged from 5 to 9. Scale the voltage of T_3 until its $wcet$ is prolonged from 5 to 6. Figure 1(e) shows the scaled result.

	$step_1$	$step_2$	$step_3$	$step_4$	$cost\ function$
$No\ scale$	100/5	200/5	50/5	0/5	3226.1
$Method1$	100/5	200/5	6.25/10	–	2865.4
$Method2$	100/5	25/10	50/5	–	2634.4
$Method3$	100/5	34.30/9	28.94/6	–	2259.3

TABLE I
Different voltage scalings for the task schedule in example 1

The load current profile without voltage scaling and the load current profiles of the three methods are shown in Table I. The $No\ scale$ profile is made up of four steps, the last of which has zero current. The profiles of Method 1 through Method 3 each has three steps. $Step_1$ and $Step_3$ contain T_0 and T_3 respectively, so the current and length of these two steps are just the current and length of T_0 and T_3. $Step_2$ is made up of T_1 and T_2, and the current in this step is the sum of the currents of these two tasks. In Table I, the first number of a step is its current and the second number is its length. The last column of the table gives the value of the cost function based on the current profiles of the three methods.

From Table I we notice that the DVS procedure of [4] given as Method 1 above applies all the slack to the last step and reduces the battery cost function by about 11% relative to the case of not using DVS. The DVS of Method 2 reduces the battery cost function by over 18% by assigning all the slack to $Step_2$ where two tasks are executed in parallel. Method 3 on the other hand distributes the slack to two steps, $Step_2$ and $Step_3$, and reduces the battery cost function by about 30%. This motivates the DVS algorithm proposed in this work in which the slack is distributed over several steps of the schedule.

The reason for the higher effectiveness of Method 3 in reducing the battery cost function is that when $Step_2$ is scaled, its current decreases and consequently its potential for reducing the cost function decreases as explained later in the proposed algorithm. Thus, after we scale $Step_2$ to some level, if we then consider the total current profile again, it can be found that at this time it is better to scale T_3 ($Step_3$) than continuing to scale $Step_2$. Though the result of Method 3 is not necessarily optimum it gives us a heuristic to use. That is, we should not scale any step maximally, but scale the voltage of one step for a small value each time and determine which other steps should be scaled based on their contribution to the total load current. The details of the procedure based on this heuristic are given in Section III.B.

A. Partitioning the task schedule

Before scaling the voltages of the different steps, we first need to partition the task schedule to obtain the steps during which scaling is used. In Example 1, the task graph and the task schedule are very simple, and it is easy to partition the schedule into four steps. The first and the third steps each contain a single task. The second step contains two parallel tasks and the fourth step contains no task. In reality, the task schedule is more complex. The $wcet$s of tasks may be different for different tasks and the parallel tasks on different PEs

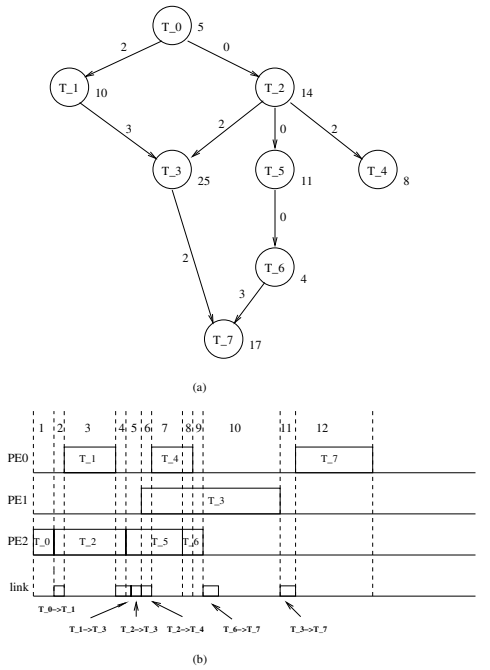

Fig. 2. Partition of task schedule.

do not necessarily have the same start time or the same finish time. As a result, the tasks may be only partially executed in parallel.

The procedure used to partition a schedule into steps is as follows. Start from the start time of the first task in the task schedule. When the start time of any task or the finish time of any task is met, mark this time point as the end of the current step and the beginning of the next step. The duration of a step is also its length. The current of a step is the sum of the currents of the tasks contained in this step. The length and current of all the steps form the load current profile for the battery model. Example 2 given next uses an eight node task graph to show the process of partitioning a schedule.

Example 2: The task graph for this example is shown in Figure 2(a). The number next to a node is the $wcet$ of the corresponding task, and the number on an edge is the communication time of the edge. Tasks are executed on three allocated PEs. All PEs are connected by a bus. The schedule is shown in Figure 2(b). The schedule includes the communication times on the "link" time line. We use the procedure given above to partition the schedule. As a result, the schedule is partitioned into 12 steps. Step 1 contains T_0, step 2 contains part of T_2, and the communication time between T_0 and T_1. Step 3 contains the whole of T_1 and part of T_2, and so on as shown in Figure 2(b).

It is important to make the following point with respect to the proposed DVS algorithm. It scales the voltages of the steps of the schedule obtained by the partitioning procedure given above. Since a task may be contained in several steps of the partitioned schedule, its assigned PE may have to execute it under different supply voltages in these steps. For example, in Figure 2(b), task T_2 is contained in three steps, so after voltage scaling, the three portions of T_2 may have different voltages. Thus the physical realization of the system will need to support executing a task at different supply voltages.

B. The DVS algorithm

Here we use the general DVS assumption that the initial task schedule is known and task execution order is fixed [2] [3]. Also, as[4], [5] and [6], we assume that the current of each task is constant.

Now the problem becomes the following. Given a partitioned task schedule, scale the voltage of certain steps such that the battery model cost function is minimum.

We use a constructive heuristic to solve this DVS optimization problem. Each time one step is chosen and the voltage of the tasks inside the step are reduced by a small value. The chosen step should cause the largest cost function reduction among all steps if they are scaled by the same amount of voltage. This procedure is repeated until the available slack is zero or the supply voltage of the tasks in all the steps reaches the lowest level which is usually set to the threshold voltage of the PE. Below is the pseudocode of the algorithm.

$while(true)$
  $\Delta\sigma_{max} = 0$;
  $k = 0$;
  $store\ the\ length\ and\ current\ of\ all\ the\ steps$;
  $store\ the\ voltage\ of\ all\ the\ tasks\ inside\ each\ step$;
  $for\ each\ step_i$
    $for\ each\ PE_j$
      $reduce\ V_j\ by\ dV_j\ such\ that\ s_i = \frac{V_j - dV_j}{V_j}$;
    $end\ for$
    $length_i = length_i / s_i$;
    $compute\ current_i$;
    $compute\ cost\ function\ reduction\ \Delta\sigma$;
    $if(\Delta\sigma_{max} < \Delta\sigma)$
      $\Delta\sigma_{max} = \Delta\sigma$;
      $k = i$;
    $end\ if$
  $set\ the\ length\ and\ current\ of\ step_i\ to\ its\ old\ value$;
  $set\ the\ voltage\ of\ all\ the\ tasks$
    $inside\ step_i\ to\ their\ old\ values$;
  $end\ for$
  $scale\ step_k\ with\ scaling\ factor\ s_k$;
  $if(slack\ is\ used\ up\ ||$
    $the\ supply\ voltage\ of\ all\ the\ tasks\ inside$
    $each\ step\ reaches\ the\ lowest\ level)$
    $break$;
  $end\ if$
$end\ while$

The input to the procedure given above is the partitioned task schedule and the output is the supply voltages of all the tasks during each step. In each iteration we consider every step and select the one that will affect the current most. For each $step_i$, we reduce the supply voltage of the tasks included in $step_i$ by a small value, which equals giving a small part of the slack to that step, and increase the length of the step accordingly. Then we compute the current of the step and compute the reduction in the cost function. After we try a $step_i$, we need to set the length, current and PE supply voltages of this step to the old values. Next we find the step which can reduce the cost function most. We use k to indicate the index of this step and $\Delta\sigma_{max}$ to store the maximum reduction in the cost function. We scale the supply voltages of all the PEs in $step_k$. When the slack is used up or the supply voltages of all the PEs in every step reach the lowest level, the while loop is broken. In each step, we choose a different dV for different PEs such that the time extension for all the tasks inside the step would be the same.

The complexity of the algorithm is $O(KN^2)$ where N is the total number of steps, and the factor K depends on dV and the amount of slack. Smaller dV and larger slack values will cause K to be

Task graph	*No. of tasks/ edges*	*No. of PEs*	*Reduction by method in* [7] (%)/ *cpu time*(s)	*Reduction by proposed method* (%)/ *cpu time*(s)
$T1$	8/9	3	75.3/ 0.05	89.8/ 0.17
$T2$	26/43	2	18.66/ 0.23	35.74/ 0.76
$T3$	40/77	5	76.15/ 26.76	86.18/ 69.69
$T4$	20/33	3	24.86/ 0.46	38.60/ 0.83
$T5$	40/77	4	19.04/ 5.51	30.05/ 7.22
$T6$	20/33	3	5.10/ 0.15	19.58/ 0.26
$T7$	20/27	2	22.24/ 0.54	42.20/ 1.12
$T8$	18/26	3	13.51/ 0.10	40.27/ 0.58
$T9$	16/15	2	14.01/ 0.14	18.76/ 0.17
$T10$	16/21	2	29.82/ 0.13	32.82/ 0.23
$T11$	30/29	3	10.53/ 0.90	47.46/ 3.91
$T12$	36/50	4	19.66/ 2.92	39.56/ 3.13
$T13$	37/36	4	27.26/ 4.46	57.22/ 5.50
$T14$	24/33	3	49.84/ 1.76	70.94/ 4.23
$T15$	40/63	4	33.95/ 14.26	77.45/ 48.33
$T16$	31/56	2	29.19/ 3.93	54.02/ 5.42
$T17$	29/56	3	47.18/ 8.24	86.64/ 20.28
$T18$	12/15	2	22.31/ 0.07	52.12/ 0.34
$T19$	14/19	2	74.55/ 0.52	85.98/ 0.91
$T20$	19/25	2	46.30/ 0.51	55.65/ 0.76
$T21$	70/99	4	13.75/ 17.97	24.59/ 18.05
$T22$	100/135	4	26.47/ 166.85	46.59/ 399.78
$T23$	84/151	2	11.93/ 53.88	30.01/ 86.12
$T24$	80/112	3	6.58/ 7.64	13.03/ 14.2
$T25$	46/92	2	19.01/ 14.54	35.73/ 25.67
RAC	9/8	2	53.50/ 0.02	67.89/ 0.06

TABLE II
THE VOLTAGE SCALING RESULTS

larger.

We ignore the battery charge consumption during the voltage transition. However it is possible to include this by appropriate adjustment to the scaled current when a voltage transition occurs.

Finally, the proposed algorithm may be integrated with allocation and assignment reported by any system level synthesis tool.

IV. Experimental results

The main aim of the presented experimental results is to show that considering the LCP as a whole instead of the currents drawn by individual tasks can reduce the cost function by greater amounts and we illustrated this benefit by using numerous task graphs of varying complexity.

To demonstrate the benefit of using the proposed DVS algorithm in extending the battery life, we have applied it to 25 random task graphs (Tgff1 - Tgff25) used by [3], and one real task graph from [6] which describes the behavior of a robot arm controller. The real task graph is executed on a single PE in [6] and here we suppose it is executed on two identical PEs. The tasks of the robot arm controller include Ohold law, Jhold law, matrix vector multiplication and several others. The detail description of the example is in [11]. We assume that all the PEs are DVS-enabled and linked by a bus. The proposed method and that of [4] were implemented using C++ on a Pentium-4/2.0GHz/512MB computer. We compute the cost function reduction after voltage scaling and compare the results of our algorithm with that of [4]. We implemented the procedure of [4] in order to compare the proposed method with it.

The results are given in Table II. The first column gives the names of the task graphs. We use T to represent Tgff, for instance, $T1$ means Tgff1, and we use RAC to represent the robot arm controller. The second column is the number of nodes and edges of each task graph. The number of allocated PEs is shown in the third column. The fourth and fifth columns show the cost function reduction and the corresponding cpu time by the algorithm of [4] and the proposed algorithm, respectively. The cost function reductions are given as a percent of the cost function value for the case where DVS is not used. It can be seen that both algorithms can reduce the cost function for every task graph. The proposed algorithm achieves greater battery cost function reduction for all task graphs. With regard to the computation time, the proposed algorithm consumes more time than the procedure of [4]. It is important to note that the percent reduction of the cost function value is determined by the slack and the amount of concurrent processing in the original schedule. In other words the percent reduction in cost function value is not directly related to the size of the task graph.

V. Conclusion

This paper addressed the problem of battery-aware dynamic voltage scaling in distributed embedded systems. Minimizing the battery cost function of a battery model was targeted. By considering task execution parallelism and the battery load current profile, the proposed DVS procedure distributes the slack in the schedule of execution of the system tasks in order to reduce the battery charge consumption. Experimental results show the better effectiveness of the proposed procedure compared with a DVS procedure for distributed embedded systems [4] that assigned slack iteratively to the latest executed tasks.

References

[1] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi, "Battery driven system design: a new frontier in low power design," in Proc. of the 15th International Conference on VLSI Design (VLSID'02), pp. 261–267, Jan. 2002.

[2] J. Luo and N. K. Jha, "Low power distributed embedded systems: Dynamic voltage scaling and synthesis," invited paper, in Proc. Int. Conf. High Performance Computing, Dec. 2002.

[3] M. T. Schmitz and B.M.Al-Hashimi, "Considering Power Variations of DVS processing Elements for Energy Minimisation in Distributed Systems, "in Proc. of International Symposium on Systems Synthesis (ISSS'01), pp. 205-255, October 1-3, 2001.

[4] P.Chowdhury and C. Chakrabarti, "Battery Aware Task Scheduling for A System-on-a-chip Using Voltage/Clock Scaling, " in Signal Processing Systems 2002 (SIPS '02). IEEE Workshop on , pp. 201-206, October 16-18, 2002.

[5] D. Rakhmatov, S. Vrudhula and C. Chakrabarti, "Battery-Conscious Task Sequencing for Portable Devices Including Voltage/Clock Scaling, " in Proc. of the 39th conference on Design automation (DAC 2002), pp. 189-194, June 10-14, 2002.

[6] D. Rakhmatov, S. Vrudhula, "Energy management for battery-powered embedded systems, " ACM Transactions on Embedded Computing Systems (TECS), Volume 2 , Issue 3, pp. 277 - 324, Aug. 2003.

[7] J. Luo and N.K.Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in Proc. of the 38th Design Automation Conference, pp. 444-449, Jun. 2001.

[8] R. Rao, S. Vrudhula and D. Rakhmatov, "Battery modeling for energy-aware system design, " IEEE Computer, Special Issue on Power-Aware and Temperature-Aware Computing, pp. 77 - 87, December 2003.

[9] D. Rakhmatov, S. Vrudhula and D. A. Wallach, "A model for battery lifetime analysis for organizing applications on a pocket computer," IEEE Trans. on, Very Large Scale Integration (VLSI) Systems, Volume: 11 , Issue: 6 , pp. 1019-1030, Dec. 2003 .

[10] K.Lahiri, A.Raghunathan and S.Dey, "Efficient Power Profiling for Battery-Driven Embedded System Design," IEEE Trans.on,Computer-Aided Design of Integrated Circuits and Systems, Volume:23, Issue:6, pp. 919-932, Jun. 2004.

[11] V. Mooney III and G. De Mecheli, "Hardware/software Codesign of Run-Time Schedulers for Real-Time Systems," ACM Journal, Design Automation of Embedded Systems, 6(1), pp. 89–144, September 2000.