Dynamic Scheduling of Imprecise-Computation Tasks in Maximizing QoS under Energy Constraints for Embedded Systems

Heng Yu, Bharadwaj Veeravalli*, and Yajun Ha
Department of Electrical and Computer Engineering
*Computer Networks and Distributed Systems (CNDS) Laboratory
National University of Singapore, Singapore, 119243
{h.yu27, elebv, elehy}@nus.edu.sg

***Abstract*— In designing energy-aware CPU scheduling algorithms for real-time embedded systems, dynamic slack reclamation techniques significantly improve system Quality-of-Service (QoS) and energy efficiency. However, the limited schemes in this domain either demand high complexity or can only achieve limited QoS. In this paper, we present a novel low complexity runtime scheduling algorithm for the Imprecise Computation (IC) modeled tasks. The target is to maximize system QoS under energy constraints. Our proposed algorithm, named Gradient Curve Shifting (GCS), is able to decide the best allocation of slack cycles arising at runtime, with very low complexity. We study both linear and concave QoS functions associated with IC modelde tasks, on non-DVS and DVS processors. Furthermore, we apply the intra-task DVS technique to tasks and achieve as large as 18% more of the system QoS compared to the conventional "optimal" solution which is inter-task DVS based.**

I. Introduction

Real-time CPU task scheduling needs to strike a trade-off between system capacity such as system energy budget, and task execution quality. In application areas such as multimedia [6] and track tracing [5], approximate results are acceptable as long as they fulfill basic execution requirements in a timely fashion. With baseline quality satisfied, the system Quality-of-Service (QoS) can thus be measured by task cycles executed beyond the basic. More often than not, this type of tasks are modeled using the Imprecise Computation (IC) model [1] [5], in which a task is decomposed into a mandatory part and an optional part. The mandatory part must be completed to produce an acceptable result, while the optional part refines and improves the result. The QoS function is represented as a linear or concave function of CPU cycles allocated to the optional part [5] [6]. The more cycles the optional task executes, the more QoS it generates.

Besides QoS requirements, energy is a primary concern for embedded systems with tight energy budget. Most of the related works, such as [7], aim to minimize energy consumption. They normally apply Dynamic Voltage Scaling (DVS) so that system energy is reduced by dynamic voltage/speed adjustment. The DVS problems can be in general divided into two categories: inter-task DVS and intra-task DVS. Being extensively studied, inter-task DVS problems focus on voltage switching among different tasks [7]. On the other hand, several studies (e.g. [8] [9]) have exploited the voltage switching opportunities within a task, which is called intra-task DVS, so that one more dimension of freedom is added to the energy optimization problems, thus better optimization results can be achieved.

Another energy-aware approach, rather than minimizing energy, aims to maximize QoS whenever energy constraints can be met. For example, in IC modeled applications, optional tasks are scheduled to execute as much as possible, before the energy constraints are violated. Aydin *et al* [2] presented an optimal solution for IC optional task scheduling problems using convex programming, which is named an OPT-LU problem with the complexity of $O(N^2 logN)$, N being the number of tasks to be executed. Rusu *et al* [3] extended [2] by taking into account energy constraints, which brings even more complexity and is not applicable to run-time circumstances. Recently, Cortés *et al* [4] proposed a quasi-static approach for dynamic slack scheduling, by providing candidate voltage/optional-cycle pairs. The runtime complexity is extremely decreased, but the discrete nature of the V/O pairs may result in inadequate QoS exploration. Also, the preparation of V/O pairs leads to an excessively intensive offline computation phase, making it less applicable to real applications who have largely enumerated numbers of task combinations.

In this paper, we propose a low complexity IC based dynamic scheduling algorithm named GCS, which maximizes QoS without violating energy constraints. The OS/processor is assumed to have an integrated inter/intra-task DVS capability. We first study GCS for optional tasks with linear QoS functions on a non-DVS processor. The method is then extended to concave QoS functions. After that, we further extend the non-DVS approach to processors with inter/intra-task DVS capability. The remainder of the paper is organized as follows. Section II gives the model definitions. Section III introduces the static scheduling phase, followed by Section IV and V, which describe the dynamic scheduling method with and without DVS, respectively. Section IV discusses the experimental results. Finally, Section VII concludes the paper.

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

II. Model Definitions

In this paper, we consider a task set N containing n periodical and independent tasks. We assume N has a hyper-period equivalent to the deadline T_d units. Under the IC model, each task i must execute mandatorily M_i cycles to guarantee an acceptable level of quality, and optionally o_i cycles to process the optional part. The optional task should not exceed its upper limit cycles O_i, i.e., $0 \leq o_i \leq O_i$. M_i and O_i are measured in Worst Case Execution Cycles (WCECs).

Each task i is associated with a linear or concave QoS function $f_i(o_i)$, which is a quantified measure of the task's QoS generated. A *linear* function of an optional task increases its value uniformly with optional cycles. A *concave* function means the increase of the task QoS exhibits a continuously decreasing rate as execution goes on. The cumulative QoS of tasks in N can thus be formulated as $QoS_N = \sum_{i \in N} f_i(o_i)$.

We assume that the processor comes with DVS capability [9], operating in a voltage range $[V_{min}, V_{max}]$. The power consumption of the processor for task i is dominated by the dynamic power $P_i = C \times V_i^2 \times f_i$, where C is the effective switching capacitance, V_i is the operating voltage, and f_i denotes the clock frequency. Suppose task i runs under V_i for $M_i + o_i$ cycles, the total energy consumption of executing all tasks in the task set N is presented as:

$$E_N = \sum_{i \in N} (C \times V_i^2 \times (M_i + o_i)). \tag{1}$$

As for the relationship between V_i and execution time t_i,

$$t_i = k \times \frac{V_i}{(V_i - V_{th})^\alpha} \times (M_i + o_i), \tag{2}$$

where k represents a constant dependent on the CMOS processing technology, α the saturation velocity index between 1.4 and 2, and V_{th} the threshold voltage. Moreover, for clarity of elaboration of our work, we ignore the time and energy consumed during voltage switching of DVS.

III. Static Scheduling

The static scheduling phase sets the starting point of designing a dynamic scheduling scheme. In our work, the assignment of optional execution cycles should be in the way that maximizes the QoS. For a task set N containing task $i \in N$, the static optimization problem can be formulated as,

maximize:

$$\sum_{i \in N} f_i(o_i) \tag{3}$$

subject to:

$$\begin{cases} 0 \leq o_i \leq O_i \\ V_{min} \leq V_i \leq V_{max} \\ \sum\limits_{i \in N} t_i \leq T_d \\ E_N \leq E_{budget} \end{cases} \tag{4}$$

where E_N and t_i are from Eqns. (1) and (2) respectively. We assume the starting time of the first task to be 0. The objective of the optional cycle decision is to maximize the total QoS value, subject to constraints in (4).

IV. Dynamic Slack Reclamation without DVS

Static scheduling being an imperative step, naturally assumes tasks running at their WCETs. However, at runtime dynamic slack is usually generated and can be reclaimed and redistributed to other tasks for additional QoS. In this section, we firstly describe a method that strives to obtain the maximal QoS for a set of tasks with linear QoS functions. The GCS algorithm dealing with concave QoS functions is presented in the second part.

A. Slack allocation for linear QoS functions

For a set of tasks with their QoS functions being linear, suppose one of them finishes its scheduled execution S cycles ahead, we claim:

Theorem 1: Under timing and energy constraints, the maximal QoS gained from dynamic slack S is achieved by allocating them to the optional task whose linear QoS function has the largest gradient.

The proof for *Theorem 1* is intuitive and skipped due to space limitation. Note that this allocation scheme is immune to timing and energy violations, the reason being their linear relationship between the optional cycles and the deadline/energy budget. This is observable from equations (1) and (2), treating V_i as a constant. In runtime situations, if S cycles of slack are generated by task i, they are reclaimed by other tasks, meaning the total number of cycles remains unchanged. Thus, the total time and energy remain unchanged.

A special situation is that the remaining optional cycles could be less than the slack allocated to it. In this case, we allot the remaining slack in a greedy fashion. That is, if the task with the largest gradient has no extra optional cycles, we allocate the slack to the task with the second largest gradient, and so on.

B. Slack allocation for concave QoS functions

It is not straight forward for slack allocation on tasks with concave QoS functions. Unlike linear QoS functions with fixed slopes of change, gradients of concave functions change at any instance. However, we can still apply *Theorem 1* in concave QoS scenarios.

For linear QoS functions, if the gradient of function $f_i(o_i)$ is larger than that of function $f_j(o_j)$, then slack S should be given to task i at any instance. On the other hand, for concave QoS functions, if slack S is given to i within an interval when any value of gradient i is larger than any value of gradient j (see Figure 1(a), S' being the interval), then according to *Theorem 1* again, the resultant extra QoS is the largest. Allocating slack S to i can be viewed as left shifting curve i by S (see Figure 1(b)). On the other hand, if S exceeds S', as shown in Figure 2(a), then after left shifting i by S' as explained above (see Figure 2(b)), the remaining slack $(S - S')$ has to be distributed between tasks i and j, and we shift curves i and j together. The resultant curves also have to intercept at the same point on the $y-$axis, as shown in Figure 2(c).

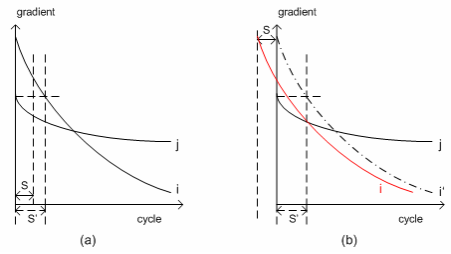

Fig. 1. (a) S within S'. Allocating S to i gives the maximal QoS. (b) Left shifting i by S cycles.

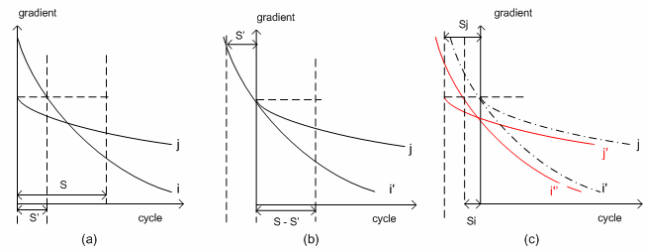

Fig. 2. (a) S larger than S'. S cannot be fully allocated to i. (b) shifting i by S'. (c) Shifting j by S_j, and i' by S_i, simultaneously.

Suppose curve i is shifted by S_i, and curve j by S_j. Their relationship can be expressed as an equation set:

$$\begin{cases} S_i + S_j = S_{remain} = S - S' \\ f_i'(S_i) = f_j'(S_j) \end{cases} . \tag{5}$$

Moreover, with more tasks shifting together, thus more gradient curves, this method is still applicable:

$$\begin{cases} \sum_{1}^{k} S_i = S_{remain} \\ f_i'(S_i) = f_j'(S_j), \forall i, j \in [1, k] \end{cases} . \tag{6}$$

Note that functions f_i' and f_j' are first derivatives of the QoS functions updated after every previous shifting. The algorithm is shown in Figure 3. The GCS function is invoked every time a task has finished execution and generated some slack. If it returns some slack after the slack allocation, it is invoked again. The worst case complexity of the GCS algorithm is $O(N^2)$. This happens when the slack is given to the the largest gradient curve until it falls to intercept the second highest curve; then the two curves shift together until they reach the third, and so on. Assuming each such shift takes $O(1)$ time, the complexity of the whole process is $O(1+2+...+N) = O(N^2)$.

```
1     FUNCTION: GCS
2     input  => slack cycles to be allotted
3     output <= remaining slack
4
5     if curve i is highest
6           shift i until intercepting second highest curve on y-axis;
7           return the remaining slack;
8     else, if i is as high as some other curves
9           shift curves together until intercepting second highest on y-axis;
10          return the remaining slack;
11    end if
12    END of GCS
13
14
15    FUNCTION: MAIN
16    for each task i in N
17          slack = slack generated by i;
18          while slack not 0
19                slack = GCS(slack);
20          end while
21    end for
22    END of MAIN
```

Fig. 3. The GCS algorithm

V. Dynamic Slack Reclamation under DVS

In the previous section we explored the slack allocation methods assuming the processor operates at a constant speed. In this section, we take DVS into the picture. This will add another dimension into the search space, in which both optional cycles and operating voltage have to be determined. However, rather than considering them comprehensively as in [3] or [4], our approach tackles the problem in two phases. Firstly, given the slack energy and slack time available, we decide the largest possible slack cycles o_i by properly choosing its operating voltage. Secondly, using the GCS algorithm, we distribute the o_i cycles to the most appropriate tasks.

A. Deciding maximal optional cycles

Before the onset of the slack, a task i works under optimally offline scheduled voltage, denoted by V_i^{sta}. If there is any dynamic slack generated from i, we define T_i as the execution time saved (the slack time), and E_i as the energy saved (the slack energy). Now, given constraints on E_i and T_i, we need to determine the voltage V_i such that the extra cycles o_i is maximized. Since QoS is represented as non-decreasing functions of cycles, it can be only maximized with maximum o_i. In order to find o_i, we claim:

Theorem 2: To obtain the largest o_i under energy and timing constraints E_i and T_i, its operation voltage V_i should remain at V_i^{sta}, i.e. $V_i = V_i^{sta}$.

Proof: We simplify Eqn. (2) by setting $V_{th} = 0$ and $\alpha = 2$, as in [4]. By manipulating Eqns. (1) and simplified (2), we obtain the $E_i - T_i$ relationship,

$$E_i = \frac{CV_i^3}{k} \times T_i \tag{7}$$

$$E_i = Ck^2 o_i^3 \times \frac{1}{T_i^2}, \tag{8}$$

where o_i replaces $M_i + o_i$ since only dynamic slack o_i is considered. Because V_i, E_i, and T_i are intrinsically bounded, there exists a bounded $E_i - T_i$ space as shown in the shaded region in Figure 4, where the two lines have gradients of CV_{max}^3/k and CV_{min}^3/k respectively. The hyperbola curve represents function $E = Ck^2o_i^3 \times \frac{1}{T_i^2}$, for which o_i holds the value that makes this curve contain the point $P(E_i, T_i)$. By Eqns. (7) and (8), every (E_i, T_i) pair determines a (V_i, o_i) pair. Notice in Eqn. (8), o_i can be maximized when both E_i and T_i are maximum. It means in the shaded region, point P represents the voltage/cycle pair in which o_i is maximum whilst E_i and T_i are both fully utilized. On the other hand, if E_i and T_i are fully used, the statically assigned energy E and time T for task i is then used up. Therefore, according to (7),

$$E = \frac{C(V_i^{sta})^3}{k} \times T. \tag{9}$$

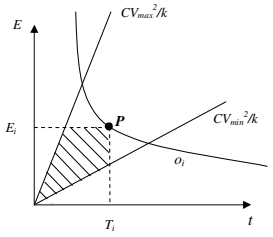

Fig. 4. The $Energy - Time$ space

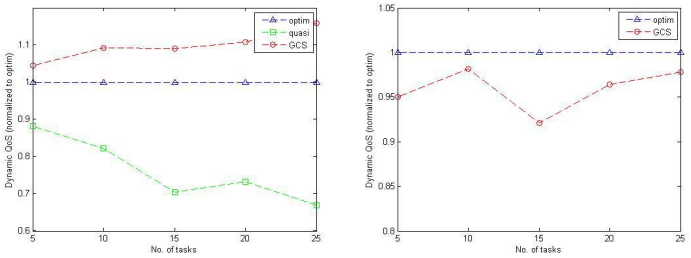

Fig. 5. (a) Normalized dynamic QoS v.s. no. of tasks. (b) Effects of no DVS applicable to GCS and optimal solutions.

Also, the cycles (both mandatory and optional ones) already executed for task i are under voltage V_i^{sta}, take $T - T_i$ units time and use $E - E_i$ units energy, so

$$E - E_i = \frac{C(V_i^{sta})^3}{k} \times (T - T_i). \tag{10}$$

From (9) and (10), it is immediately apparent that we can obtain $V_i = V_i^{sta}$. In total, we can conclude that if E_i and T_i are maximally used, V_i must be set to V_i^{sta} and simultaneously o_i is maximized. Hence the proof. ■

B. Allotting cycles o_i

With the maximum o_i determined, we allocate the slack to tasks to achieve the largest dynamic QoS. The operating voltage of the o_i slack cycles remains unchanged. Note that the operating voltage (predetermined at static scheduling phase) of the tasks probably differs from V_i, so intra-task DVS can be applied for voltage switching within the task being allotted cycles. This provides more optimization opportunity than inter-task only DVS approaches. Moreover, because o_i operates in conformation to energy and timing constraints, wherever these cycles are allocated, the total system energy and timing constraints remain inviolated.

VI. Results and Discussions

To evaluate our GCS algorithm, we have performed extensive simulation studies with over 700 task sets. Each task set is attributed with a deadline T_d in the magnitude of several milliseconds. The number of cycles of each task is derived based on given T_d and E_{budget} so as to manipulate a constrained optimization. We choose the voltage level in the interval of $[1.2V, 1.6V]$. The QoS functions for tasks are adopted from [4] as $\alpha_i o_i + \beta_i \sqrt{o_i} + \sqrt{3}\theta_i o_i$, where α_i, β_i, and θ_i are randomly generated between 0 and 1.

We have compared our GCS algorithm with two other methods, one being the quasi-static approach with 4 points per task and deadline slack between 0.1 and 0.6 [4], the other being the inter-task based optimal approach, which runs the static optimization every time a task finishes execution and generates some slack. Figure 5(a) shows the dynamic QoS gained, in which every point shows the average result of evaluating 50 task sets. The data for comparison are additional QoS generated in the dynamic phase, represented by $QoS_{dynamic} = QoS_{actual} - QoS_{static}$. Results of the GCS algorithm and quasi-static approach are normalized by the inter-task DVS optimal results. We observe from Figure 5(a) that GCS algorithm generates extra QoS of as large as 18% more than the inter-task DVS based optimal approach for different number of tasks in a task set.

In order to make it evidenced that the above superiority of GCS to the inter-task DVS optimal solution is subject to intra-task DVS, we conducted the second set of experiment, by manipulating T_d sufficiently long so that only energy is the scarce resource. Under this condition, the static optimization process results in every task running at the lowest voltage, $1.2V$. Hence in both the inter-task DVS based optimal approach and our GCS algorithm, no DVS are required. Results in Figure 5(b) show that the GCS algorithm is very near to the optimal but never exceeding it. Since DVS is the only factor of variety, we conclude that the GCS algorithm gives better performance because of the intra-task DVS capability.

VII. Conclusions

In this paper, the GCS algorithm is proposed for dynamic QoS generation under energy constraints. It achieves better QoS gain with low complexity.

References

[1] K.J. Lin, S.Natarajan, and J.W.S. Liu, Imprecise Results: Utilizing Partial Computations in Real- Time Systems, Proc. IEEE RTS, 1987.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Optimal Reward-Based Scheduling for Periodic RealTime Tasks. IEEE Transactions on Computers. Vol 50, Issue 2, February 2001, pp. 111-130.

[3] C. Rusu, R. Melhem, and D. Mosse. Maximizing rewards for real-time applications with energy constraints. ACM Transactions on Embedded Computing Systems. Vol. 2, Issue 4, November 2003, pp. 537-559.

[4] L. A. Cortés, P. Eles, and Z. Peng. Quasi-Static Assignment of Voltages and Optional Cycles in Imprecise-Computation Systems with Energy Considerations. IEEE Trans. on Very Large Scale Integration Systems, Vol. 14, Issue 10, Oct 2006, pp. 1117-1129.

[5] J.-Y. Chung, J.W.S. Liu, and K.-J. Lin, Scheduling Periodic Jobs that Allow Imprecise Results. IEEE Transactions on Computers, vol. 39, no. 9, September 1990, pp. 1156-1174.

[6] S. V. Vrbsky, J. W. S. Liu. Producing monotonically improving approximate answers to relational algebra queries. In Proceedings of IEEE Workshop on Imprecise and Approximate Computation, December 1992.

[7] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In Proceedings of IEEE Symposium on Foundations of Computer Science, 1995.

[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] J. Seo, T. Kim, and N. D. Dutt. Optimal integration of inter-task and intra-task dynamic voltage scaling techniques for hard real-time applications. International Conference on Computer-Aided Design (ICCAD), 2005, pp. 450-455.