6H-09

Reducing Jitter and Energy in Hard Real-Time Systems using Intra-task DVFS Technique

Boyu Tseng
School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology

Kiyofumi Tanaka
School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology

I. Introduction

In real-time embedded systems, energy consumption is one of significant issues. To reduce energy dissipation without violating timing constraint, **Dynamic Voltage and Frequency Scaling (DVFS)** has been widely applied in many researches. Moreover, DVFS enables the system to control the actual execution/response times of periodic tasks, thus it is also applicable to reduce finish time jitter. Especially predictability of response times is another high demand in particular applications like control system or data acquisition.

With above concerns, we propose a jitter-aware intra-task DVFS scheme for mitigating finish time jitter whilst keeping energy efficiency in hard real-time systems. In this paper, we mainly extend the implementation of intra-task DVFS from [3]–[5] which exploits program control flow and data flow analysis. Furthermore, we select benchmark programs [2] as a target set of periodic tasks to simulate multitasking and evaluate jitter/energy reduction.

II. Preliminaries

A. Related Work

During the execution of a task, actual execution time may vary from time to time, consequently generating some slack time. Those slack times give a system opportunity of slowing down the execution speed whilst still meeting deadline. To identify/predict presence of slack time during runtime, the previous works [3], [4] proposed worst-case execution path evaluation to trace required execution cycles through different execution paths of every single task's control flow graph (CFG). These assess a feasible frequency-updated ratio to make task complete as late as possible without deadline miss.

B. Causes of Finish Time Jitter

In periodic task scheduling, finish time jitter may arise from runtime variation in execution time and preemption/interference. Regarding task's running behaviour in the form of CFG, every feasible execution path may contain different workload which ranges from cost of best-case execution path (BCEP) to cost of worst-case execution path (WCEP). BCEP and WCEP are the certain paths of task's CFG resulting in best-case execution cycles (BCECs) and worst-case execution cycles (WCECs), respectively. Consequently execution time variation among task's instances would occur. Additionally, interference time variation is equivalent to execution time variation of higher priority tasks.

III. Jitter-aware Intra-task DVFS scheme

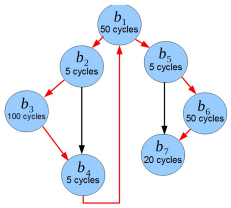
To avoid large finish time jitter, we propose a usage of DVFS technique which changes the execution speed to adapt to the actual execution path of task's CFG and actual interference time, i.e., controls the actual response time by proactive approach. The procedures of the proposed intra-task DVFS scheme is as follows:

A. Execution Cycle Estimation

Target task's C source code is converted into CFG. Then the number of cycles needed for executing each basic block is calculated on a WCET basis [2] regardless of cache/pipelining behavior, as an example shown in figure. 1.

```
1  cond_1 = func1();
2  cond_2 = func2();
3  while(cond_1 == func3()) {
4      if(cond_2)
5          func4();
6      func5();
7  }
8  if(cond_3)
9      func6();
10 func7();
```

(a) Source code of target task



(b) CFG of target task

Fig. 1: **Execution cycle estimation of target task**

B. Control and Data Flow Analysis

This step aims at identifying specific instruction points where the runtime variation on execution time will arise. In the CFG's point of view, searching for branch and loop are our target. This is because branches decide the actual execution paths in CFG and different loop iteration counts lead to different execution cycles. In addition, we also identify the loop dependency [5] using data flow analysis. The loop dependency is the variable which determines the actual number of iterations through one loop.

C. Scaling Point Placement

In the work of [4], they inserted checkpoints into task's program right after branch instructions as an additional sequence of instructions. Those checkpoints evaluate the

remaining execution cycles (REC) of successive execution paths by referring to the **Mining Table**[1]. Accordingly, we classify all checkpoints into three different types: B-type (for branch), L-type (for loop), and P-type (for instruction of value assignment of loop dependency). When a checkpoint is executed in a task's program, the system compares REC with a given timing constraint (target response time described in the next step) and conducts voltage frequency scaling if the current frequency cannot meet the constraint.

D. *Frequency-Updated Ratio*

For each task, a target response time is determined. All frequency scaling points inside a task try to adjust its frequency so that actual response time of every instance is close to the target response time, which shortens the range of response time variation. In this paper, we propose two different directions:

*1) **Static Target Response**:*

$$R_i^{target} = BCRT_i + \alpha_i \times (WCRT_i - BCRT_i) \quad (1)$$

$$f_{new} = \frac{REC_i}{R_i^{target} - time_{executed} - I_{worst(i)}} \quad (2)$$

For task i, target response time (R_i^{target}) is calculated with response time analysis [1], task i's best-case response time ($BCRT_i$), and worst-case response time ($WCRT_i$). α is a user-defined ratio. $I_{worst(i)}$ is worst-case interference time and $time_{executed}$ is time for which task i has been executed.

*2) **Profile-based Target Response**:*

$$R_i^{expect} = time_{executed} + \frac{REC}{f_{current} + I_{average(i)}} \quad (3)$$

$$R_i^{target} = \begin{cases} R_i^{min}, & R_i^{expect} < R_i^{min} \\ R_i^{max}, & R_i^{expect} > R_i^{max} \end{cases}$$

$$f_{new} = \frac{REC_i}{R_i^{target} - time_{executed} - I_{average(i)}} \quad (4)$$

In the profiling approach, the target response time is determined by the average-case response time and average-case interference time ($I_{average(i)}$).

IV. Evaluation

A. *Experimental Setup*

For experiment, we built a CFG-based multitasking simulator in C++11. For target tasks, we selected four different CFGs of benchmark programs [2] and made another simple CFG. The target tasks were simulated with Rate-Monotonic scheduling for their hyperperiod. The frequency-power combinations used in the simulation are 300MHz-114.38mW, 600MHz-303.15mW, 720MHz-437.49mW, 800MHz-542.73mW, and 1000MHz-736.08mW. In the evaluation, we used the same scheduling patterns for environments with DVFS and without DVFS to see how much jitter and energy consumption can be reduced.

[1] Mining Table records addresses of branch instructions, RECs of successive execution paths, etc.

B. *Experimental Results*

We simulated the same scheduling patterns with different execution paths in each task 10 times. Table 1 presents the mean values of tasks' absolute finish time jitter. In this experiment, we defined bs.c and ludcmp.c as jitter-sensitive tasks. The other ones (compress.c, cfg.1, and matmult.c) were defined as lower-energy demand tasks.

Task	WCEC (cycle)	Period (ns)	$Jitter_{NonDVFS}$ (ns)	$Jitter_{DVFS}$ (ns)
bs.c	9750	286765	14784	11061
compress.c	11950	62895	N/A	N/A
cfg.1	1810	36200	N/A	N/A
matmult.c	1890395	42963523	N/A	N/A
ludcmp.c	27546	272733	9967	4895

TABLE I: **Reduction of finish time jitter**

Task	$Energy_{NonDVFS}$ (uJ)	$Energy_{DVFS}$ (uJ)	Energy Saving (%)
bs.c	345860	202110	41.2
compress.c	1697631	1156270	24.8
cfg.1	276316	227091	17.8
matmult.c	5768	4957	15.6
ludcmp.c	113686	71937	36.7

TABLE II: **Reduction of energy consumption**

V. Conclusions

In this paper, we proposed a jitter-aware intra-task DVFS scheme which takes both execution and interference time variations into account. According to the experimental results, our proposed scheme can reduce finish time jitter for two jitter-sensitive tasks by 24.8% and 47.8%, respectively. In addition, even for aiming at lower jitter performance, the total energy consumption is reduced as a side effect. However, there are some limitations on our approach in that jitter reduction is feasible only for certain tasks. In future work, we aim at addressing effective boundary of jitter reduction on any task (any kind of task's CFG structure) and add practical transition overhead of DVFS to simulation.

References

[1] Anton Cervin, Bo Lincoln, Johan Eker, Karl-Erik Arzén, and Giorgio Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 1–9. Gothenburg, Sweden, 2004.

[2] Diego Pinheiro, Rawlinson Gonçalves, Eduardo Valentin, Horácio de Oliveira, and Raimundo Barreto. Inserting dvfs code in hard real-time system tasks. In *Computing Systems Engineering (SBESC), 2017 VII Brazilian Symposium on*, pages 23–30. IEEE, 2017.

[3] Dongkun Shin and Jihong Kim. Optimizing intratask voltage scheduling using profile and data-flow information. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):369–385, 2007.

[4] Tomohiro Tatematsu, Hideki Takase, Gang Zeng, Hiroyuki Tomiyama, and Hiroaki Takada. Checkpoint extraction using execution traces for intra-task dvfs in embedded systems. In *Electronic Design, Test and Application (DELTA), 2011 Sixth IEEE International Symposium on*, pages 19–24. IEEE, 2011.

[5] Burt Walsh, Robert Van Engelen, Kyle Gallivan, Johnnie Birch, and Yixin Shou. Parametric intra-task dynamic voltage scheduling. In *Proceedings of the Workshop on Compilers and Operating Systems for Lower Power (COLP 2003)*, 2003.