Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 2746 – 2754
© TÜBİTAK
doi:10.3906/elk-1806-170

Energy-efficient scheduling for real-time tasks using dynamic slack reclamation

Vasanthamani KANNAIAN[*], **Visalakshi PALANISAMY**
Department of Electronics and Communication Engineering, PSG College of Technology, Coimbatore, India

Received: 23.06.2018 • **Accepted/Published Online:** 15.04.2019 • **Final Version:** 26.07.2019

Abstract: The application domains of portable embedded real-time systems range from safety-critical applications like electronic control units in the automotive sector to entertainment applications like digital cameras and setup boxes in the consumer electronics sector. These systems attract users with multiple and sophisticated functionalities in a single system. The developments in VLSI technology have enabled IC manufacturers to incorporate more features in a single die, which facilitates these multipurpose systems. The increase in features and functionalities has a direct impact on power consumption and has to be strictly accounted for in the design of these embedded systems. The possibility of achieving low power consumption is obtained by using only effective software techniques along with the hardware power-saving methods. The dynamic power management techniques with the support of real-time operating systems (RTOSs) provided in recent microcontrollers enable better power management using software control. The scheduling of tasks in real-time systems is based on the worst-case execution time (WCET), whereas tasks based on their environmental conditions may complete their execution before the WCET. This leads to increased laxity (slack time) and reduced utilization of the CPU. This paper proposes Fixed Window DynaClam, a dynamic reclamation algorithm, to effectively utilize slack time. The clock frequency of the processor is slowed down by reclaiming the available slack time to reduce the energy consumption. The Fixed Window DynaClam algorithm is implemented for nonpreemptive periodic tasks in the controllers with dynamic voltage and frequency scaling power management features. The novelty of the algorithm is that the clock frequency is not only adjusted based on the real-time demands of the tasks but also does not overload the kernel of the RTOS as the complexity is on the lower side.

Key words: Real-time task scheduling, nonpreemptive tasks, energy-optimized, dynamic voltage and frequency scaling, dynamic slack reclamation

1. Introduction

The performance of embedded real-time systems is characterized by power consumption for two reasons: one, for extending the battery life, and two, for handling the heat dissipation [1]. Numerous static and dynamic energy-efficient techniques are proposed by experts and researchers since conservation of energy is the need of the day. The slack time is reclaimed in static algorithms using worst-case execution time (WCET) and actual execution time (AET) in dynamic algorithms. Many researchers have proposed tools and techniques for the exact estimation of WCET as the slack reclamation is based on the predefined nature of the tasks in static algorithms. The upper bound for utilization of the processor in static algorithms has to be definitely kept at a reduced level since they do not have any control during the runtime. Dynamic techniques overcome these drawbacks and have precise control of the system at the cost of additional system resources. Thus, a dynamic

[*]Correspondence: kvm.ece@psgtech.ac.in

technique with less computational complexity and less usage of system resources can be an effective solution. DPM and DVFS are the two power management techniques popularly used in real-time systems. In DPM, the processor is either switched to low power modes with essential peripherals working at low frequency or the processor is switched to sleep mode. The drawback in this technique is that the processor requires additional time and energy to move between low power and normal mode. In DVFS, the operating frequency of the processor is reduced by reducing either the supply voltage or the operating frequency. The DVFS technique has time and energy overhead due to frequency switching. This paper deals with the dynamic technique to reclaim the available slack time and applies DVFS to reduce the clock frequency of the processor without missing the deadline of the tasks. The paper is organized as follows: existing research using the dynamic reclamation method is discussed in Section 2, system models are characterized in Section 3, and a motivational example is illustrated in Section 4. The proposed technique is explained in Section 5 and the results and discussions are given in Section 6. Finally, the conclusion of this paper is presented in Section 7.

2. Related work

The energy-efficient scheduling of real-time tasks employs either static or dynamic algorithms. Researchers have also proposed a few algorithms that initially calculate the task frequency using static slack and further refine it using dynamic slack. Each task in the task set is operated at a different frequency depending on the available slack time. Static techniques are always preferred to avoid failure in hard real-time systems. The concepts and tools used for the better estimation of WCET and few algorithms for slack reclamation are presented in this section.

2.1. Analysis of worst-case execution time

Authors [2] have discussed the following three approaches for the estimation of WCET: measurement-based, static, and hybrid. Furthermore, they have elaborated on the concepts of the measurements and WCET-aware source code optimization techniques. Authors [3] have integrated the WCET analyzer with the compiler infrastructure and used the following process:

1. Low-level intermediate representation (LLIR) of the compiler was translated into the WCET analyzer aiT's exchange format, CRL2.
2. Timing analysis was performed.
3. WCET data produced by the WCET analyzer aiT were imported into LLIR for further compiler optimizations.

aiT is available for ARM7, Power-PC, TI TMS320C33, and the Infineon Tricore c1.3. Authors [4] have discussed WCET analysis tools for commercial (BounD-T, AbsInt) and research (Heptane) purposes. They have analyzed the industrial code using the aiT WCET analysis tool for the ARM7TDMI processor. They have conducted the experiments on system calls, interrupt regions, and other functions. METAMOC [5] was developed to estimate the WCET for programs that run on platforms with caching and pipelining. Analysis was performed for ARM7, ARM9, and ATMEL AVR 8-bit platforms. The doctoral dissertation of Timon Kelter [6] dealt with the WCET analysis of single-core and multicore processors. In PROARTIS [7], a probabilistic timing analysis method for analyzing the timing behavior of the next generation of real-time embedded systems was elaborated. Authors have devised two methods, static probabilistic timing analysis (SPTA), a static method that derives a priori probabilities from the processor or software model, and measurement-based probabilistic timing analysis (MBPTA), a method that derives probabilities from the complete run of the program under study.

2.2. Slack reclamation techniques

Authors [8] have considered four speed variation models: continuous, discrete, Vdd-hopping, and incremental. The continuous model has been formulated using geometric programming and the Vdd-hopping model using a linear program. Authors [9] have used the DVFS feature to optimize the power consumption in multiprocessor systems-on-chip (MPSoCs). The offline method explores the static slack using a greedy and integer linear programming approach and the online algorithm calculates the dynamic slack time by subtracting the AET from the WCET. The new processor speed is computed using Eq. (1) only if the calculated dynamic slack is greater than the worst-case voltage switching time (WCVST).

$$f' = f \frac{WCET}{WCET + WCRST + slack}, \tag{1}$$

where $WCVST$ is the voltage switching overhead.

Authors [10] have applied guided-search heuristics in a JPEG2000 decoder application to find the right frequency level to utilize the slack. Authors [11] have proposed a low-power fluid slack analysis (lpFSA) algorithm to estimate slack online. After exploring the slack, a rate monotonic algorithm was used to schedule the tasks. In [12] the authors used deterministic stretch to fit (DSF) in ARM processors; the slack thus obtained from the current task was utilized by the next task in the queue. They applied the algorithm to ARM1176JZF-S and Cortex-A9 processors. In [13], the authors proposed multiple voltage frequency selections for the DVFS algorithm. Two voltage frequencies were calculated and applied to the processors with discrete frequency levels. Authors [14] have proposed two dynamic approaches: greedy procrastination and look-ahead procrastination. In [15], the dynamic slack reclamation method was applied to a multimedia application based on SoCLib. The authors in [16] proposed a dynamic slow-down factor computation algorithm called energy-aware proportionate slack management (EAPSM) for multiprocessors.

3. System model

3.1. Task models

A periodic task is represented by the tuple (T_i, D_i, C_i). T_i is the period of task i, D_i is the relative deadline of task i, and C_i is the computation time of task i. The laxity L_i of the task is defined as the difference between the deadline and the computation time as given in Eq. (2):

$$L_i = D_i - C_i. \tag{2}$$

3.2. Processor model

The power consumption of the processors is directly proportional to the clock frequency as shown in Eq. (3) [17]:

$$P_{active} = S * f^3. \tag{3}$$

Two processor models, continuous and discrete, are taken into consideration in this work. In the continuous model, the processor clock frequency is continuously varied from minimum to maximum whereas in the discrete model it is varied with fixed steps from minimum to maximum frequency. The ratio between calculated processor frequency and maximum frequency is called the slow-down factor, which is denoted as η.

4. Motivational example

Consider the task set with three tasks: (15, 15, 3), (30, 30, 4), and (45, 45, 3). The hyper period ($\mathrm{lcm}(\sum_{i=0}^{n} T_i)$) is 90 and the total utilization is 0.4. Uniform slow-down with frequency inheritance (USFI) [18] and the hyperperiod-based method (HPBM) [19] are the two static methods taken for consideration and applied for the given task set. After applying the USFI algorithm to the above task set, the total utilization was increased to 0.918 with the slow-down factors of 0.4667 (η_1), 0.4083 (η_2), and 0.4083 (η_3) with an available slack time of 7.34 time units in the hyperperiod. Total utilization of the tasks using two static reclamation algorithms is shown in Figure 1. The results were taken by varying the number of tasks in the task set with different initial total utilization. The processor is still underutilized even for WCET using static algorithms and certainly the utilization will further reduce during the execution since AETs of tasks will be lesser than WCETs. The objective of this paper is to devise an algorithm to dynamically adjust the slow-down factor based on the online task requirements. Furthermore, the complexity of the algorithm should be the lowest possible as it works dynamically.

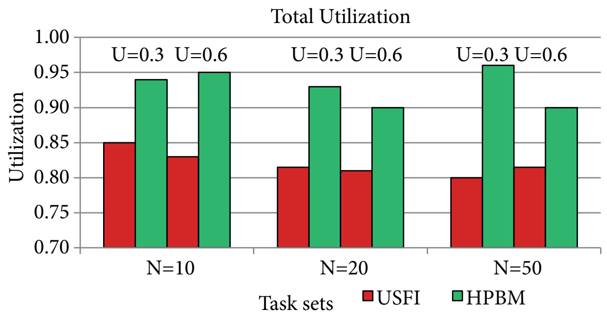

Figure 1. Total utilization of the tasks using static reclamation algorithms.

5. Fixed Window DynaClam: a dynamic reclamation technique

Initially, the slow-down factor is calculated based on the WCET and then it is further refined using the slack time reclaimed based on AET. The slow-down factor is recalculated for the pending tasks after the completion of the current task. The process of computing the slow-down factors employing the Fixed Window DynaClam algorithm is shown in Figure 2. The algorithm explores the slack for a slot of size that is twice the period of a high priority task (η_{hp}) (Eq. (4)):

$$\eta = \frac{\sum_{i=0}^{m} C_i}{D_m - t}, \tag{4}$$

where D_m is twice the deadline of the highest priority task and t is the instantaneous scheduling time.

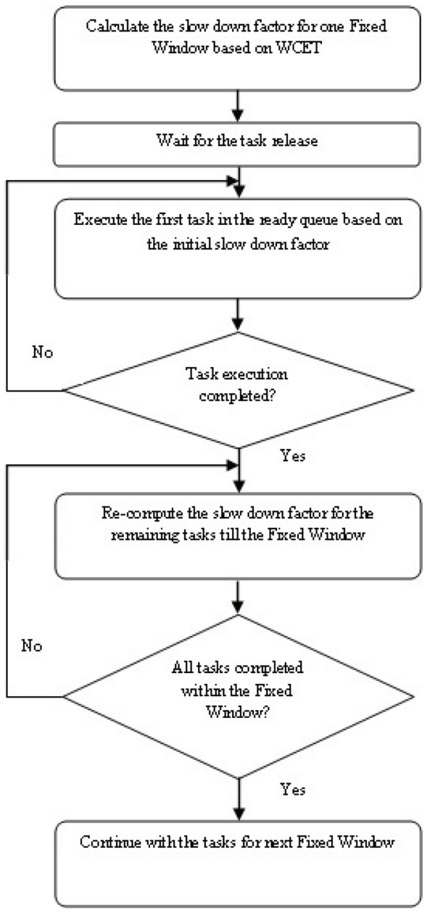
Calculate the slow down factor for one Fixed Window based on WCET
Wait for the task release
Execute the first task in the ready queue based on the initial slow down factor
No
Task execution completed?
Yes
Re-compute the slow down factor for the remaining tasks till the Fixed Window
No
All tasks completed within the Fixed Window?
Yes
Continue with the tasks for next Fixed Window

Figure 2. Fixed Window DynaClam algorithm: computation procedure.

5.1. Illustrative example

The task set shown in Table 1 is taken for illustration and the steps for computation of the slow-down factor are as follows;

1. Calculate the sum of the execution time for all tasks in ready queue and the next instance of task 1 (τ_1^1 for the first time).
2. Initialize t with zero.
3. Assign the deadline of τ_1^1 to D_m.
4. Calculate the slow-down factor using Eq. (4) and assign to all the pending tasks.
5. Recalculate the slack time and slow-down factor after the completion of current task.
6. Repeat the process for a fixed window (D_m).

The results at the completion of each step for the example task set in Table 1 with a clock frequency of 20 MHz are shown in Table 2. At time $t = 0$, the sum of the execution time of the two instances (τ_1^0 and τ_1^1) of task 1 and one instance of tasks 2 and 3 (τ_2^0 and τ_3^0) are computed and the resulting frequency is 18

MHz. The frequency of operation becomes 17.8 MHz if the 0th instance of task 1 completes at time $t = 1$. The recalculated clock frequency for tasks 3 and 1 is 15.4 MHz, if task 2 completes pdf-latex-vat time $t = 3.5$. The calculation for the first instance of task 1 is performed at time $t = 8$, which is also the completion time of task 3. Now the execution of tasks within the first fixed window slot (10 time units) is completed and the algorithm continues with the second window. The deadline for the second fixed window is 20 time units and the total number of tasks to be executed is 3 (τ_2^2, τ_1^3 and τ_1^4).

Table 1. Example task set.

Task	T_i	D_i	C_i
τ_1	5	5	1
τ_2	10	10	3
τ_3	20	20	4

Table 2. Slow-down factor calculation using DynaClam algorithm.

Fixed window = 1		t = 0	t = 1	t = 3.5	t = 8
Deadline = 10 f = 20 MHz	No. of tasks in D_m	4	3	2	1
	Total C_i	9	8.888	5.624	1.3
	η	0.9	0.988	0.865	0.65
	f_{clock}	18	17.8	15.4	10
Fixed window = 2		t = 0	t = 3.5	t = 8.5	
Deadline = 20 f = 8.71 MHz	No. of tasks in D_N	3	2	1	
	Total C_i	10	8.0	2.46	
	η	1.0	1.23	1.64	
	f_{clock}	10	12.3	20.0	

Theorem 1 *The reclaimed slack time after execution of the current task is used for the pending tasks in the ready queue.*

Proof The algorithm calculates slack time for the tasks in the ready queue based on WCET and for the completed tasks based on AET. The slack time is recalculated at the end of the execution of the current task and the slow-down factor is calculated to spread the tasks within the fixed window time.

$$W_n(t) = \sum_{i=0}^{n} \left\lceil \frac{t}{P_i} \right\rceil C_i \tag{5}$$

$$L_n(t) = 2P_h - W_n(t) \tag{6}$$

The execution time of tasks in the ready queue is given by Eq. (5) at the time instance t and the remaining laxity in the fixed window is as per Eq. (6). □

Theorem 2 *All the tasks in the ready queue within the fixed window are operated with the same clock speed, which reduces the overhead in frequency switching.*

Proof The slow-down factor for all the tasks in the ready queue within the fixed window is calculated based on the WCET. All the tasks will operate with the same clock frequency if the AET of all the tasks in the fixed window is equal to WCET. The transition delay and energy overhead for an Intel Core2 Duo E6850 processor for moving from 3 MHz to 2.67 MHz is 52,688 cycles and 344.1 μJ. To move between 3 MHz and 2.00 MHz, it takes 113,409 cycles with energy overhead of 1161 μJ [20]. The change in clock frequency is only marginal and hence the energy and time overhead involved in frequency switching is on the lower side. □

6. Results and discussion

The performance of the algorithm is analyzed with the testing tool SPARK [21], developed in MATLAB. The task set input to the scheduling algorithm can be selected as either user-given task sets or automatic task sets generated by the tool. SPARK has a facility to automatically reduce the AET for the task set to perform the testing of dynamic algorithms. The task set in Table 1 is given as the input to SPARK manually and the schedule created is shown in Figure 3. The laxity has reduced from 6 to 1.22 time units and the energy has decreased to 50% for one hyperperiod of the schedule using regular RMS for nonpreemptive tasks along with DynaClam.

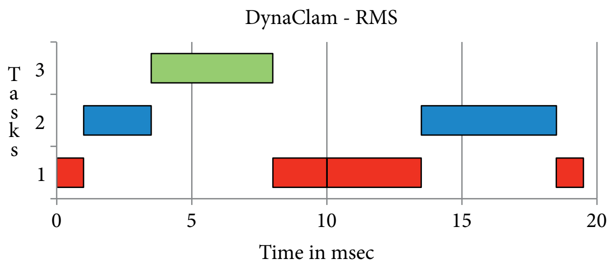

Figure 3. DynaClam algorithm: RMS schedule.

The power consumption is calculated using Eq. (3), where f is the frequency of tasks after slow-down. The energy is calculated by multiplying the task frequency by its AET. The maximum clock speed for the CPU is set as 20 MHz and a task set with 10 tasks with the periods ranging from 10 to 200 time units with initial total utilization (U_i) of 0.3 and 0.6 is used for analysis. The normalized energy for the task set with two different initial utilizations is shown in Figure 4 and the corresponding normalized laxity is shown in Figure 5. Using the DynaClam algorithm, the energy consumption of the task set is reduced by 50% and 20% when compared with the static algorithms USFI and HPBM, respectively. Normalized laxity also decreases significantly by 40% for the same task sets. The reason for the significant decrease in energy consumption is due to reduced clock frequency and effective utilization of the slack time.

7. Conclusion and future work

Algorithms employing static reclamation of the laxity can help to reduce the energy consumption of tasks only to a certain extent. The processor is underutilized and the laxity is still available even after using the slow-down because of the difference in the AET and WCET. This laxity can be effectively utilized only by dynamic reclamation algorithms. The Fixed Window DynaClam algorithm has been proposed to reclaim the slack time dynamically and the results depict a significant decrease in energy consumption. Moreover, the overhead due to frequency switching is also less because the laxity recovered is utilized immediately for the tasks to be executed in the fixed window. The greatest advantage of this algorithm is that it can either decrease or increase the

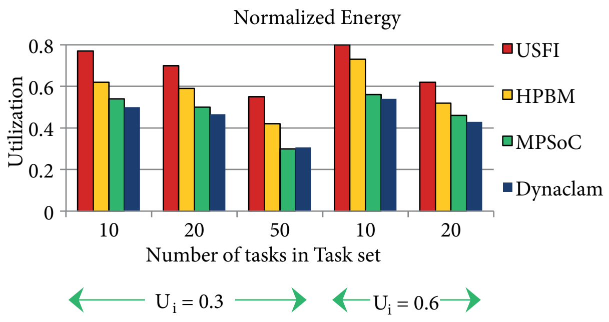

Figure 4. Normalized energy: DynaClam algorithm.

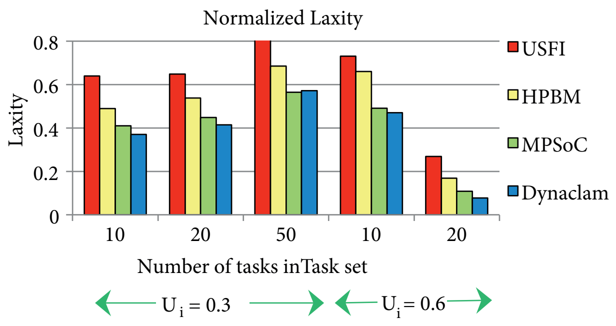

Figure 5. Normalized laxity: DynaClam algorithm.

operating frequency depending on the demand of tasks within the fixed window. The time taken to execute the algorithm is 5 μs, which is 10 times lower than HPBM and so the algorithm does not overload the kernel of the RTOS. The algorithm is applied and tested using periodic nonpreemptive tasks, which can be further extended to preemptive and sporadic tasks.

References

[1] Li YT, Malik S. Performance Analysis of Real-Time Embedded Software. 1st ed. New York, NY, USA: Springer Science & Business Media, 2012.

[2] Lokuciejewski P, Marwedel P. Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems. 1st ed. Amsterdam, the Netherlands: Springer Science & Business Media, 2010.

[3] Falk H, Lokuciejewski P, Theiling H. Design of a wcet-aware c compiler. In: IEEE 2006 Workshop on Embedded Systems for Real Time Multimedia; Seoul, Korea; 2006. pp. 121-126.

[4] Sandell D. Evaluating static worst case execution time analysis for a commercial real-time operating system. MSc, Mälardalen University, Västerås, Sweden, 2004.

[5] Dalsgaard AE, Olesen MC, Toft M, Hansen RR, Larsen KG. METAMOC: Modular execution time analysis using model checking. In: 2010 International Workshop on Worst-Case Execution Time Analysis. Brussels, Belgium: DROPS; 2010. pp. 113-123.

[6] Kelter T, Borghorst H, Marwedel P. WCET-aware scheduling optimizations for multi-core real-time systems. In: 2014 Embedded Computer Systems: Architectures, Modeling, and Simulation; Agios Konstantinos, Greece; 2014. pp. 67-74.

[7] Cazorla FJ, Quiñones E, Vardanega T, Cucu L, Triquet B. Probabilistically analyzable real-time systems. ACM T Embed Comput S 2013; 12: 94.

[8] Aupy G, Benoit A, Dufossé F, Robert Y. Reclaiming the energy of a schedule: models and algorithms. Concurr Comp-Pract E 2013; 25: 1505-1523.

[9] Singh AK, Das A, Kumar A. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In: ACM 2013 Design Automation Conference; New York, USA; 2010. p. 115.

[10] Yu H, Veeravalli B, Ha Y. Leakage-aware dynamic scheduling for real-time adaptive applications on multiprocessor systems. In: IEEE 2010 Design Automation Conference; Anaheim, CA, USA; 2010. pp. 493-498.

[11] Chen DR. Slack computation for DVS algorithms in fixed-priority real-time systems using fluid slack analysis. J Syst Architect 2011; 57: 850-865.

[12] Khan J, Bilavarn S, Belleudy C. Energy analysis of a DVFS based power strategy on ARM platforms. In: IEEE 2012 Faible Tension Faible Consommation; Paris, France; 2012. pp. 1-4.

[13] Rizvandi NB, Taheri J, Zomaya AY. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. J Parallel Distr Com 2011; 71: 1154-1164.

[14] Niu L, Li W. Energy-efficient fixed-priority scheduling for real-time systems based on threshold work-demand analysis. In: IEEE 2011 International Conference on Hardware/Software Codesign and System Synthesis; Taipei, Taiwan; 2011. pp. 159-168.

[15] Ramesh P, Ramachandraiah U. Energy aware proportionate slack management scheduling for multiprocessor systems. Procedia Comput Sci 2018; 133: 855-863

[16] Mhedhbi I, Atitallah RB, Jemai A. Dynamic slack reclamation strategy for multiprocessor systems. In: IEEE 2012 Mediterranean Electrotechnical Conference; Yasmine Hammamet, Tunisia; 2012. pp. 979-984.

[17] Saha S, Ravindran B. An experimental evaluation of real-time DVFS scheduling algorithms. In: ACM 2012 International Systems and Storage Conference; New York, NY, USA; 2012. p. 7.

[18] Jejurikar R, Gupta R. Energy aware task scheduling with task synchronization for embedded real time systems. IEEE T Comp Aid D 2006; 25: 1024–1037.

[19] Vasanthamani K, Visalakshi P. Energy optimized scheduling for non-preemptive real-time systems. Turk J Elec Eng & Comp Sci 2017; 25: 3085-3096.

[20] Park S, Park J, Shin D, Wang Y, Xie Q, Pedram M, Chang N. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. IEEE T Comput Aid D 2013; 32: 695-708

[21] Vasanthamani K, Visalakshi P, Shankar SR. SPARK-A tool for synthesis and analysis for scheduling of real-time tasks. Asian J Res Soc Sci Hum 2017; 7: 990-999.