IEICE TRANS. INF. & SYST., VOL.E98–D, NO.8 AUGUST 2015

LETTER

T-L Plane Based Real-Time Scheduling Using Dynamic Power Management

Youngmin KIM[†], **Ki-Seong LEE**[†], **Byunghak KWAK**[†], ***Nonmembers***, ***and*** **Chan-Gun LEE**[†a)], ***Member***

SUMMARY We propose an energy-efficient real-time scheduling algorithm based on T-L Plane abstraction. The algorithm is designed to exploit Dynamic Power Management and generates a new event called event-s to render longer idle intervals, which increases the chances of switching a processor to the sleep mode. We compare the proposed algorithm with previous work and show that it is effective for energy management.

key words: *real-time scheduling, dynamic power management, energy management*

1. Introduction

There has been increasing interest in energy-efficient real-time/embedded systems equipped with multi-processors. The most popular techniques to save energy of a processor include Dynamic Power Management (DPM) and Voltage Frequency Scaling (VFS). DPM [1] switches a processor to lower power consumption states such as idle or sleep modes from the active mode. VFS scales the frequency of a processor since the frequency directly relates to its power consumption. There are recent scheduling approaches based on Dynamic Voltage Frequency Scaling (DVFS) [2]–[5] and Static Voltage Frequency Scaling (SVFS) [6], [7].

The T-L plane scheme is actively adopted to develop scheduling algorithms for real-time tasks on multi-core systems. LLREF [8] and LRE-TL [9] algorithms are efficient real-time scheduling algorithms for periodic and sporadic tasks, respectively. An extension to LLREF was made so that it can schedule tasks with synchronization [10]. Later Funaoka et al. advanced the algorithm to allow tasks to be work conserving [11] and help the system to reduce wasteful idle times. It was further extended to deal with energy consumption with static voltage and frequency scaling [7]. T-R plane was proposed along with NNLF scheduling algorithm [12] to handle dynamic events such as aperiodic services. Cho et al. proposed work conserving and non-work conserving algorithms to reduce migrations of tasks [13]. Alhussian et al. proposed a technique for sporadic tasks to reduce the task migrations [14]. A few approaches [2]–[5] made the energy-efficient extension to the T-L plane based algorithms using DVFS recently. RT-DVFS [2] scales the frequency of processors with respect to the fluctuation of task execution time. LRE-DVFS-EACH [3] scales the frequency of processor when sporadic tasks are released on multiprocessor. GMF [4] is a non-uniform frequency scaling algorithm for uniform multiprocessors.

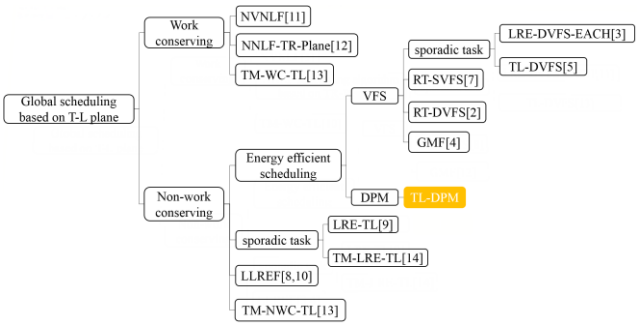

Fig. 1 Taxonomy of T-L plane based global scheduling algorithms.

However, to the best of our knowledge, most energy-efficient scheduling algorithms based on the T-L plane abstraction studied so far did not consider DPM but only utilized VFS. Zhang et al. briefly mentioned the utilization of idle intervals [5], however, they mainly focused on DVFS and did not fully exploit DPM.

In this paper, we make an extension of the T-L plane based real-time scheduling algorithm LLREF [8] to exploit the DPM technique. Figure 1 shows the taxonomy of global scheduling algorithms based on T-L plane abstraction. In the figure, TM represents the technique for minimizing task migrations. WC and NWC represent work conserving and non-work conserving scheduling algorithms, respectively. The proposed algorithm (TL-DPM in Fig. 1) introduces a new event called event-s which is designed to let the system steal tasks scheduled for the upcoming T-L plane. The event renders longer idle intervals in the next plane in hoping that they become long enough to accommodate the sleep mode.

2. T-L Plane Algorithm Review

Figure 2 shows the basic concept of T-L plane. For each task T_i with periodical arrivals at a_{i1}, a_{i2}, ..., and the cost c_i, its deadline is assumed to be the same as its period. In the figure, horizontal and vertical axes represent time (T) and local remaining execution (L), respectively. Each dotted slope from (a_{ij}, c_i) to $(a_{ij+1}, 0)$ represents the fluid schedule of a task. The absolute deadline of every task divides time as the vertical dotted lines and a triangle containing its fluid schedule is placed between every two adjacent deadlines.

Manuscript received September 18, 2014.
Manuscript revised March 5, 2015.
Manuscript publicized May 12, 2015.

[†]The authors are with School of Computer Science and Engineering, Chung-Ang University, 221 Heukseok, Dongjak, Seoul 156–756, Korea.
a) E-mail: cglee@cau.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2014EDL8184

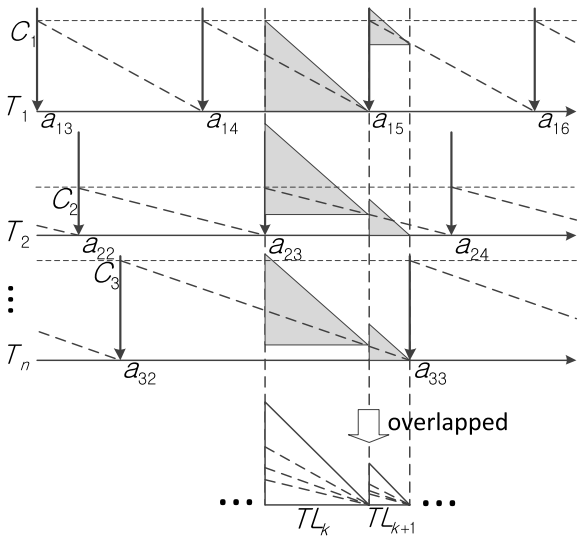

Fig. 2 Constructing T-L plane [3].

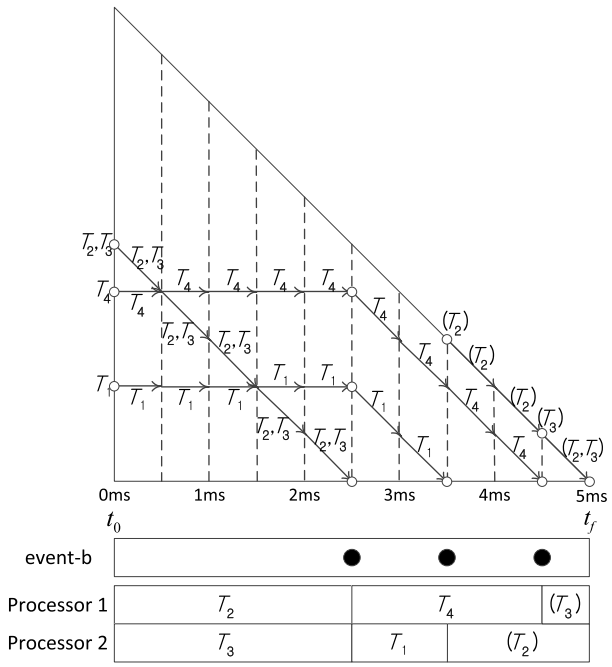

Fig. 3 Scheduling example for LLREF and T-L DPM.

By overlapping all triangles with the same time intervals, we can construct a T-L plane as shown in the bottom of the figure. Since it is impractical to realize the fluid scheduling, scheduling decisions are made based on local laxity times of the tasks and events such as event-c and event-b which will be explained in the following.

For illustration purpose, assume that we are given a task set as shown in Table 1. Figure 3 shows how LLREF [3] schedules the tasks using T-L plane abstraction. The tasks are shown as tokens moving from t_0 to t_f, the start and end time of the current plane, respectively. For this example, ignore the tokens enclosed by parenthesis; they are only valid in our proposed algorithm explained in the next section. The

Table 1 Task property.

Task	Period	WCET	Utilization
T_1	5ms	1ms	0.2
T_2	10ms	5ms	0.5
T_3	10ms	5ms	0.5
T_4	15ms	6ms	0.4

tasks assigned to the processors move diagonally down like T_2 and T_3 at t_0. Otherwise, they move horizontally like T_4. When a token hits the bottom, an event-b occurs and the processor that has executed the task becomes available. Note that every processor is available at t_0 in each T-L plane. When a processor is available, the task with the highest local utilization is assigned to it. The local utilization of a task T_i is $l_{i,j}/(t_f - t_{cur})$ where $l_{i,j}$ is remaining time at current time t_{cur}. The local utilization indicates the processor capacity required for executing T_i from t_{cur} to t_f. When a token hits the slope of T-L plane, an event-c occurs and the corresponding task must be assigned to one of the processors immediately.

3. Extension for DPM

For the processors supporting multiple operating modes, we can reduce the power consumption by utilizing the sleep and idle mode of the processors. LLREF can be incorporated with DPM by the following simple extension to the b-event handler and we shall refer to it as a naive approach in this paper:

- If the wait queue is not empty, then the handler resumes its normal behavior, i.e., pick a task and let the processor execute it.
- If the wait queue is empty, then check if $t_f - t_{cur} > c_{sleep}$. If it is true, then the processor is switched to the sleep mode till the end of the current plane. Otherwise, we switch the processor to the idle mode.

In the above c_{sleep} is break-even time for the sleep mode; in order to put a processor to the sleep mode, the idle interval must be long enough to compensate the overhead for the transitions, which occur when the processor is switched from active to sleep and from sleep to active.

Now we extend the naïve approach so that we can render longer idle intervals in the next plane. The idea is to allow the scheduler to steal the tasks originally scheduled to be executed in the next plane when a transition to the sleep mode is not permissible.

Figure 4 illustrates this idea in detail. A new event named event-s occurs instead of event-b when the execution of a task is just completed by a processor, the wait queue is empty, and $t_f - t_{cur}$ is smaller than c_{sleep}. These conditions imply that the processor has nothing to do and the remaining time is not long enough to utilize the sleep mode.

At initialization (else part), it accepts a set of ready tasks and returns an array of running tasks. When an event-c is detected, the corresponding task is moved to Q_{run} from ready queue ζ_{cur}. The victim is the one with the smallest remaining time in the execution queue.

In case of an event-b, a task in ζ_{cur} moves to Q_{run}. How-

Algorithm 1 The TL-DPM algorithm

Input : Tasks in ready state
Output : Array of dispatched tasks to processors
T-# of tasks
M-# of processors
ζ_{cur} - ready queue of current plane
ζ_{done} - finished queue of current plane
ζ_{next} - ready queue of next plane
Q_{run} - Queue for running tasks, Maximal size of Q_{run} is M
l_i - local remaining execution time of T_i
t_1, t_2 - Temporary variables for tasks
p_1 - Temporary variables for processors
if event-c **then**
  t_1 = getTaskOfThisEvent();
  erase(t_1, ζ_{cur});
  t_2 = pop-back(Q_{run});
  push-front(t_2, ζ_{cur});
  push-front(t_1, Q_{run});
else if event-b **then**
  t_1 = getTaskOfThisEvent();
  p_1 = getProcessorOfThisEvent();
  push-back(t_1, ζ_{done});
  erase(t_1, Q_{run});
  t_2 = pop-front(ζ_{cur});
  if t_2 is not Null **then**
    push-back(t_2, Q_{run});
  else
    putProcessorToSleep(p_1);
  end if
else if event-s **then**
  t_1 = getTaskOfThisEvent();
  t_2=pop-front(ζ_{next});
  p_1 = getProcessorOfThisEvent();
  push-back(t_1, ζ_{done});
  for $i \leftarrow 1, sizeof(\zeta_{next})$ **do**
    if t_2 is in the ζ_{done} **then**
      if deadline of t_2 is not end time of current plane **then**
        push-back(t_2, Q_{run});
        return Q_{run};
      end if
    else
      push-back(t_2, ζ_{next});
    end if
    t_2=pop-front(ζ_{next});
  end for
  putProcessorToIdle(p_1)
else
  putEveryProcessorToActive();
  clear(ζ_{cur});
  t_2=pop-front(Q_{run});
  while t_2 is not Null **do**
    if remaining time of t_2 is not zero **then**
      push-back(t_2, ζ_{cur});
    end if
    t_2=pop-front(Q_{run});
  end while
  append($\zeta_{cur}, \zeta_{next}$);
  decideLocalParameters(ζ_{cur});
  initNextReadyQueue(ζ_{next});
  clear(Q_{run});
  for $i \leftarrow 1, M$ **do**
    t_1=Pop-front(ζ_{cur});
    push-back(t_1, Q_{run});
  end for
end if
return Q_{run}

Fig. 4 T-L plane DPM algorithm.

Table 2 Processor property.

Processor mode	Power	Upward transition time
Active	30mW	n/a
Idle	6mW	0.006ms
Sleep	0.005mW	5ms

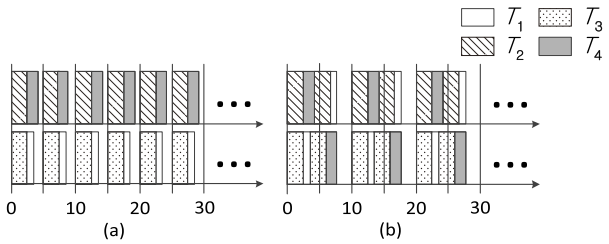

Fig. 5 Scheduling examples for naive (a) and proposed algorithms (b).

ever, if ζ_{cur} is empty, then the processor executing the task caused the event-b enters the sleep mode and the task is removed from Q_{run}. Upon the detection of event-s, the scheduler selects a task originally scheduled for the next plane and its corresponding instance has been completed for current T-L plane. If there is no such task, we let the processor go to the idle mode. In the algorithm, the method *decideLocalParameters* sorts the tasks in current queue in the order of local laxities and adjusts the local parameters. The method *initNextReadyParameters* initializes the tasks to be executed in the next plane.

Let us run the algorithm for the same task set used for illustrating LLREF before and compare their energy consumptions. Table 2 shows the processor characteristics, which are typical in many modern processors. In Fig. 3, the tokens and tasks surrounded by parenthesis are scheduled by our algorithm only. In the naïve approach, however, those intervals would have been filled with the idle mode. Figure 5 (a) shows the scheduling results in the naïve approach for longer durations; the processors P_1 and P_2 are switched to the idle mode for 1ms and 1.5ms every 5ms, respectively. Because the idle intervals are shorter than c_{sleep} = 2ms, none of the processors can exploit the sleep mode. In contrast, Fig. 5 (b) shows that all the processors can enter the sleep mode every 10ms in our proposed algorithm. From the given processor characteristics, it turns out that our proposed algorithm reduces the energy by 323mW every 30ms compared with the naive approach.

Theorem 1. The proposed algorithm is never worse than the naïve approach in terms of energy management.

Proof: The total energy consumptions for a hyper-period under the naïve and our proposed algorithms are calculated as shown in the following, respectively:

$$E_{Naive} = E_A A_{Naive} + E_I I_{Naive} + E_s S_{Naive}$$
$$E_{Proposed} = E_A A_{Proposed} + E_I I_{Proposed} + E_S S_{Proposed}$$

where E_{Naive} means the total energy consumption under the naïve approach. A_{Naive}, I_{Naive}, and S_{Naive} represent the total durations of the active, idle, and sleep intervals, respectively under the naïve approach. E_A, E_I, and E_S mean the

unit energy consumptions for active, idle, and sleep mode, respectively. $A_{Proposed}$, $I_{Proposed}$, $S_{Proposed}$, and $E_{Proposed}$ can be understood similarly.

Note that A_{Naive} and $A_{Proposed}$ are the same since the two schedules are given the same tasks and their total utilizations are the same. Because the total durations of both approaches are the same, i.e., $A_{Naive} + I_{Naive} + S_{Naive} = A_{Proposed} + I_{Proposed} + S_{Proposed}$, the equation $I_{Naive} + S_{Naive} = I_{Proposed} + S_{Proposed}$ holds.

From the algorithm description, it is obvious that whenever the condition for sleep mode is satisfied under the naïve approach, the corresponding processor is switched into the sleep mode. So is true for our proposed algorithm. In addition, the proposed algorithm may steal upcoming tasks when the naïve one forces the processor(s) into the idle mode. Those unused idle intervals are accumulated in the next plane and increase the chances of making idle intervals long enough to exploit the sleep mode. This means that $I_{Naive} \geq I_{Proposed}$ and $S_{Naive} \leq S_{Proposed}$. Because $E_I > E_S$, $E_{Naive} \geq E_{Proposed}$ holds.

4. Conclusions

We proposed an energy-efficient real-time scheduling algorithm based on T-L Plane abstraction for multi-processor systems. In order to exploit DPM, we introduced a new event that can render longer idle intervals by stealing tasks scheduled for the next plane. We showed that the proposed algorithm enables better energy management than a naïve extension of LLREF in typical processor environments.

Acknowledgments

This research was supported by the Chung-Ang University Research Scholarship Grants in 2014 and the National Research Foundation (NRF-2014R1A2A2A01005519).

References

[1] V. Devadas and H. Aydin. "Real-time dynamic power management through device forbidden regions," Proc. IEEE Real-Time and Embedded Technology and Applications Symposium, pp.34–44, St. Louis, MO, USA, April 2008.

[2] K. Funaoka, A. Takeda, S. Kato, and N. Yamasaki, "Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors," Proc. International Symposium on Industrial Embedded Systems, pp.27–33, Le Grande Motte, France, June 2008.

[3] D.-S. Zhang, F.-Y. Chen, H.-H. Li, S.-Y. Jin, and D.-K. Guo, "An energy-efficient scheduling algorithm for sporadic real-time tasks in multiprocessor systems," Proc. IEEE 14th International Conference on High Performance Computing and Communications, pp.187–194, Banff, Canada, Spet. 2011.

[4] G.A. Moreno and D. de Niz, "An optimal real-time voltage and frequency scaling for uniform multiprocessors," Proc. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.21–30, Seoul, Korea, Aug. 2012.

[5] D. Zhang, D. Guo, F. Chen, F. Wu, T. Wu, T. Cao, and S. Jin, "TL-plane-based multi-core energy-efficient real-time scheduling algorithm for sporadic tasks," ACM Trans. Architecture and Code Optimization, vol.8, issue 4, no.47, Jan. 2012.

[6] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," Proc. 1st USENIX Symposium on Operating Systems Design and Implementation, no.2, CA, USA, Nov. 1994.

[7] K. Funaoka, S. Kato, and N. Yamasaki, "Energy-efficient optimal real-time scheduling on multiprocessors," Proc. IEEE 11th International Symposium on Object-Oriented Real-Time Distributed Computing, pp.23–30, Orlando, FL, USA, May 2008.

[8] H. Cho, B. Ravindran, and E.D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," Proc. IEEE 27th Real-Time Systems Symposium, pp.101–110, Rio de Janeiro, Brazil, Dec. 2006.

[9] S. Funk and V. Nadadur, "LRE-TL: An optimal multiprocessor algorithm for sporadic task sets," Proc. Symposium on Real-Time and Network Systems, vol.46, issue 3, pp.332–359, Paris, France, Oct. 2009.

[10] H. Cho, B. Ravindran, and E.D. Jensen, "Synchronization for an optimal real-time scheduling algorithm on multiprocessors," Proc. IEEE 2nd International Symposium on Industrial Embedded Systems, pp.9–16, Lisbon, Portugal, July, 2007.

[11] K. Funaoka, S. Kato, and N. Yamasaki, "Work-conserving optimal real-time scheduling on multiprocessors," Proc. IEEE 20th Euromicro Conference on Real-Time Systems, pp.13–22, Prague, Czech, July 2008.

[12] K. Funaoka, S. Kato, and N. Yamasaki, "New abstraction for optimal real-time scheduling on multiprocessors," Proc. IEEE 14th International Conference on Embedded and Real-Time Computing Systems and Applications, pp.357–364, Kaohsiung, China, Aug. 2008.

[13] H. Cho, B. Ravindran, and E.D. Jensen, "T-L plane based real-time scheduling for homogeneous multiprocessors," Journal of Parallel and Distributed Computing, vol.70, issue 3, pp.225–236, June 2010.

[14] H. Alhussian, N. Zakaria, F.A. Hussin, and H.T. Bahboug, "Reducing tasks migration in LRE-TL real-time multiprocessor scheduling algorithm," Proc. 4th International Conference on Electrical Engineering and Informatics, vol.11, pp.235–242, Selangor, Malaysia, June 2013.