Energy-Aware Fixed-Priority Multi-core Scheduling for Real-Time Systems

Junyang Lu, Yao Guo

Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
Email: {lujunyang, yaoguo}@pku.edu.cn

Abstract—**Multi-core processors are becoming the dominant choice due to energy and thermal considerations, which also applies to embedded and real-time systems. While fixed-priority scheduling with task-splitting in real-time systems are widely applied, current approaches have not taken into consideration energy-aware aspects such as dynamic voltage/frequency scheduling (DVS). In this paper, we propose two strategies to apply DVS to fixed-priority scheduling algorithms with task-splitting for periodic real-time tasks on multi-core processors. We first propose a strategy that does traditional DVS for each processor after scheduling (post-DVS), which ensures all tasks meet the timing requirements on synchronization. We then propose a new strategy, which determines the frequency of each task before scheduling (pre-DVS) according to the total utilization of task-set and number of cores available, so that the system could take full advantage of all the cores. The combination of frequency pre-allocation and task-splitting makes it possible to maximize energy savings with DVS. We perform a series of simulations to compare the performance of each algorithm with two state-of-the-art scheduling algorithms with task-splitting. Simulation results show that the pre-DVS algorithm we proposed has performed satisfactorily on both schedulability and energy consumption in comparison to the previous approaches.**

Index Terms—**Real-time systems, multi-core scheduling, dynamic voltage scaling (DVS), energy optimization**

I. INTRODUCTION

Multi-core processors have been adopted not only in high-performance servers and personal computers, but also for embedded and real-time systems. Many real-time scheduling algorithms for multi-core processors have been proposed in recent years, among which the semi-partitioned fixed-priority multi-core scheduling algorithms [3], [4] could achieve higher utilization bound. On the other hand, energy consumption is critical for many battery-operated embedded and real-time systems. However, although techniques such as dynamic voltage/frequency scaling (DVS) [7] have been available in most modern processors, these energy-aware aspects have not been considered in the recently proposed semi-partitioned fixed-priority multi-core scheduling algorithms with high utilization bound.

In order to understand the energy implications of these semi-partitioned fixed-priority multi-core scheduling algorithms, this paper explores the possibility of applying DVS to two recently proposed semi-partitioned fixed-priority multi-core scheduling algorithms: SPA2 [3] introduced by Guan *et al.* and PDMS_HPTS_DS (PHD) [4] introduced by Lakshmanan *et al.*. Among the two techniques, SPA2 could reach a utilization bound of 69.3% and PHD 65%, both considerably higher compared to the previous approaches in priority-based multi-core scheduling.

Because neither algorithm has considered energy, we introduce two different methods to apply DVS to the above two multi-core scheduling algorithms (SPA2 and PHD):

- **DVS after scheduling (post-DVS)**: We first develop a extended DVS algorithm based on the traditional DVS algorithm for fixed-priority scheduler [6], and apply it after scheduling with task-splitting to evaluate the schedulability and energy savings. To be specific, we set the maximal frequency for certain split sub-tasks, so that all the former sub-tasks could finish before the latter ones release. This ensures that the scheduling will meet the timing requirements on synchronization for scheduling with task-splitting.

- **DVS before scheduling (pre-DVS)**: We then propose a new algorithm, which determines the frequency of each task before scheduling in order to achieve better performance on both schedulability and energy consumption. Pre-DVS actually schedules the tasks with prolonged execution time based on DVS and could ensure that all the tasks meet the timing requirements after DVS. Specifically, the frequency is determined by the total utilization of task-set and number of cores available, pursuing the maximal balance of tasks in each processor and therefore getting considerably more energy saving compared to the post-DVS approach.

In order to evaluate the two approaches mentioned above, we developed a simulator to compare their schedulability and energy consumption with various total utilization and number of processors.

When considering their schedulability, PHD performs much better than SPA2 when the utilization is over 70% (Both algorithms can be one hundred percent schedulable when the utilization is below 65% due to their utilization bounds). PHD can keep a schedulability more than 90% until the utilization comes up to 90%, while SPA2 sharply decreases to zero.

In terms of energy consumption, PHD with post-DVS algorithm nearly get close to the worst case. While the other three strategies show considerable energy savings, among which PHD with pre-DVS saves most energy because of its equal distribution of tasks to every processor available. Overall, the simulation results show that PHD with the pre-

DVS algorithm has demonstrated lower energy consumption compared to the other three strategies while maintaining the good schedulability of PHD.

The main contribution of this paper is that we explored the possibility of applying DVS to semi-partitioned fixed-priority multi-core scheduling algorithms. We have presented two different approaches to apply DVS, and simulation results shows that energy-aware scheduling is achievable without affecting the schedulability of two state-of-the-art algorithms. To the best of our knowledge, this is the first work considering energy-efficient multi-core scheduling with task-splitting.

II. PRELIMINARIES

This section introduces the background information and assumptions, as well as notations used in this paper.

A. Task Model

We shall use the following notation throughout this paper. We consider a *task-set* $T_1, T_2, ..., T_n$ comprising of N periodic tasks. This task-set is assigned to M processor cores. We use the classical (C, P, D) model to represent the parameters of a task T, where C is the worst-case computation time at maximal frequency of each job of T, P is the period of T , and D is the deadline of each job of T relative to job release time. For a task without split, its deadline D equals to its period P. When it comes to a subtask of a split task, its deadline D is less than its period P, in order to set aside time for other subtasks from the same split task.

Tasks $T_i : (C_i, P_i, D_i)$ are ordered such that $i < j$ implies $D_i < D_j$. Since our proposed algorithms use deadline-monotonic scheduling as the scheduling algorithm on each processor, we can use the task indices to represent the task priorities, i.e., i has higher priority than j if and only if $i < j$. The utilization of each task i is defined as $U_i = C_i/T_i$. The total utilization U_{tot} is given by $\sum U_i$.

While scheduling, some tasks are split and assigned to different processors. We call these tasks *split task*, which are split into several *subtasks*. For a split task T_i, T_i^k denotes the kth subtask of T_i. We call the last subtask of T_i its *tail subtask*, and other subtasks its *body subtasks*.

The subtasks of a split task need to be synchronized to execute correctly. That means, T_i^{k+1} cannot start execution until T_i^k is finished. Therefore, the time for a subtask T_i^k to execute is shorter than its period, in order to share time with other subtasks from the same split task. For a subtask T_i^k split from task T_i, its deadline D_i^k satisfies the equation $D_i^k = P_i - \sum_{j=1}^{k-1} R_i^j$, where R_i^k means the actual time span of the subtask from its release to completion.

B. Energy Model

For processors based on CMOS technology, the power consumption is dominated by dynamic power dissipation P_d, which is given by: $P_d = C_{ef} \times V_{dd}^2 \times f$, where V_{dd} is the supply voltage, C_{ef} is the effective switching capacitance, and f is the processor clock frequency. For simplicity, processor frequency can be considered roughly linearly to the supply
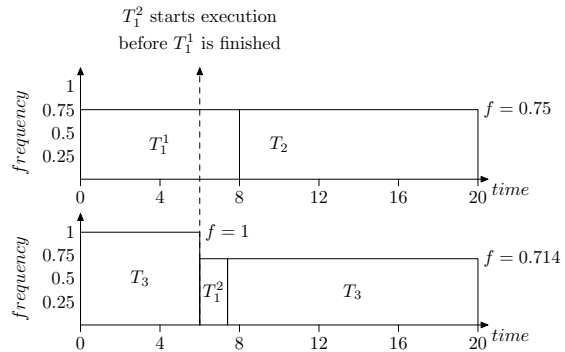

Fig. 1. Synchronous Violation with Traditional DVS

voltage: $S = k \times \frac{(V_{dd}-V_t)^2}{V_{dd}}$, where k is a constant and V_t is the threshold voltage [2]. Thus, P_d is almost cubically related to f: $P_d \approx C_{ef} \times \frac{f^3}{k^2}$. Since the time needed for a specific task is: $time = \frac{C}{f}$, where C is the number of cycles to execute the task, the energy consumption of the task, E, is $E = P_d \times time \approx C \times C_{ef} \times \frac{f^2}{k^2}$. When decreasing processor speed, we can also reduce the supply voltage. This reduces processor power cubically and energy quadratically at the expense of linearly increasing the task's latency.

For simplicity, we assume the power dissipation function $p(f) = f^3$, and the total power of a multi-core processor is simply a sum of the power dissipated in each core: $p_{tot} = \sum p_i$.

III. THE POST-DVS APPROACH

Pillai and Shin have introduced a DVS algorithm for fixed-priority schedulers [6], achieving energy savings by reducing the operating frequency and voltage when remaining tasks need less than the remaining time before next deadline. However, for scheduling with task-splitting, one cannot reduce the frequency freely, because the synchronous requirements of split-task may be violated when postponing the execution time of each subtask, as the example shown in Fig. 1.

In order to achieve energy savings for scheduling with task-splitting, we develop a new post-DVS algorithm based on the previous work, which did not take task-splitting into consideration. We reselect the frequency when any of the tasks is released or finishes, according to the earlier one between next deadline and next release time, and the remaining total cycles current tasks still need. Before selecting frequencies, we first examine whether the task/subtask is a body subtask. If it is a body subtask, we just execute it under maximal frequency and reselect the frequency when this subtask finishes, so that the synchronous requirement is satisfied.

The detailed description of the algorithm for post-DVS scheduling with task-splitting is shown in Algorithm 1. In function $available\_time\_until\_next\_time\_line()$, time line means release time or deadline. So the function returns the smaller one between available time until the next deadline and available time until the next release time.

Algorithm 1: Post-DVS for Scheduling with Task-Splitting

1: **select_frequency():**
2: s_m :=available_time_until_next_time_line()
3: $f := f_{max} * \sum d_i/s_m$
4: **upon task_release(T_i):**
5: $C\_left_i := C_i$
6: s_m :=available_time_until_next_time_line()
7: allocate_cycles(s_m)
8: **if** T_i is body subtask **then**
9:     $f := f_{max}$
10: **else**
11:     select_frequency()
12: **end if**
13: **task_completion(T_i):**
14: $C\_left_i := 0$
15: $d_i := 0$
16: select_frequency()
17: **during task_execution(T_i):**
18: decrement C_left_i and d_i
19: **allocate_cycles(k):**
20: **for** $i := 1$ to N **do**
21:     **if** $C\_left_i < k$ **then**
22:         $d_i := C\_left_i$
23:         $k := k - C\_left_i$
24:     **else**
25:         $d_i := k$
26:         $k := 0$
27:     **end if**
28: **end for**

We can prove that the above post-DVS algorithm will not violate any of the timing requirements (the proof is not included due to space limitation).

IV. THE PRE-DVS APPROACH

In this section, we first consider the potential of energy optimization, and then propose a new DVS algorithm which determines the frequency of each task before scheduling (pre-DVS) that can save more energy compared to post-DVS.

A. Energy Optimization

Given a task-set with periodic real-time tasks and a processor with M cores, we need to find a schedulable task-to-core assignment that minimizes the energy consumption under DVS. Thus, two conditions [1] must be satisfied:

1) The assignment must evenly divide the total load U_{tot} among all the processors.
2) In each processor with total utilization S, the frequency f must be constant and equal to S.

That is, each processor must manage to run under constant frequency f that satisfies $f/f_{max} = \sum U_i/M$, where f_{max} is the maximal frequency a processor's multiple supply voltages could provide. We call this frequency the "ideal frequency":

$$f_{ideal} = f_{max} \times \sum U_i/M$$

With frequency pre-allocation and task-splitting, it is possible to get very close to the minimization of energy, as long as the assignment is schedulable.

B. Algorithm Description

Even though the two conditions for energy minimization are usually hard to satisfy, we could still try to achieve a result as close as possible. Below are the detailed steps of the proposed pre-DVS algorithm.

Step 1. We try to set all the tasks' execution frequency to the same ideal level: $f = f_{ideal} = f_{max} \times \sum U_i/M$.

Step 2. Because some tasks may not execute in such low frequency due to their relatively high utilization, which might be greater than f/f_{max}, we deal with the tasks in two different ways: if a task T_i satisfies $U_i > f/f_{max}$, we set its frequency f_i as $f_{max} \times U_i$. Otherwise, we set its frequency f_i as f.

Step 3. After changing the frequency of each task, we need to extend their execution time accordingly. For convenience in the next step, we just regard the extended execution time as new execution cycles, and store the original execution cycles. Thus, for each task, $C_i = Old\_C_i$.

Step 4. We try the new task-set with the scheduling algorithm (PHD for example). If it is schedulable, we can decide that the minimal f that is schedulable for certain task-set and processors. Otherwise, we need to gradually increase the frequency f and repeat Step 2, 3, 4 until it is schedulable.

As a result, we could assign the tasks to processors as balanced as possible and achieve significant energy savings with the pre-DVS algorithm. Experiments in the next section will show that such algorithm could maintain the good schedulability of PHD while consuming much less energy.

Algorithm 2: Pre-DVS for Scheduling with Task-Splitting

1: **for** each f available among the range of $[f_{max} \times \sum U_i/M, f_{max}]$ **do**
2:     **for** each $i \in [1, N]$ **do**
3:         $Old\_C_i := C_i$
4:     **end for**
5:     **if** $U_i <= f/f_{max}$ **then**
6:         $f_i := f$
7:         $C_i := C_i \times f_{max}/f$
8:     **else**
9:         $f_i := f_{max} \times U_i$
10:        $C_i := P_i$
11:    **end if**
12:    **if** $Schedulable() = TRUE$ **then**
13:        Done
14:    **else**
15:        **for** each $i \in [1, N]$ **do**
16:            $C_i := Old\_C_i$
17:        **end for**
18:    **end if**
19: **end for**

The detailed description of the algorithm Pre-DVS Scheduling with Task-Splitting is shown in Algorithm 2.

V. SIMULATION

We have developed a simulator to evaluate the schedulability and energy savings from dynamic voltage scaling in a multi-core real-time system for both pre-DVS and post-DVS approaches on the two scheduling algorithms SPA2 and PHD.

A. Simulation Methodology

We developed a simulator for the operation of hardware capable of voltage and frequency scaling with real-time scheduling. This simulator takes input as utilization per processor and number of processor, and calculates the schedulability and power consumption for each of the algorithms we have studied: SPA2 with post-DVS, PHD with post-DVS, SPA2 with pre-DVS and PHD with post-DVS.

To achieve an ideal effect from DVS, we assume that the multiple supply voltages is continuous. That is, the processor could execute in any frequency below the max frequency. We have also simulated energy results modeling a real processor based on Intel PX270, the results show similar trends (the results for PX270 are not included due to space limitation).

Only energy consumed by the processor is computed, and variations due to different types of instructions executed are not taken into account. With this simplification, the task execution model can be reduced to counting cycles of execution, and execution traces are not needed. In particular, this does not consider preemption and task switching overheads, or the time required to switch operating frequency or voltages. There is no loss of generality from these simplifications. The preemption and task switch overheads are the same with post-DVS or pre-DVS, so they have no (or very little) effect on relative power dissipation numbers.

The real-time tasks are specified using pairs of (C, P), indicating their computation cycles and period. The task-sets are generated as follows. Each task has an equal probability of having a period among [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]. This simulates the varied mix of short and long period tasks commonly found in real-time systems. The computation time is uniformly distributed over [0, P]. Finally, the last task computation requirements are scaled by a constant chosen such that the sum of the utilizations of the tasks in the task-set reaches a desired value. We simulate each task on platforms with 2, 4, 8 or 16 processors. For a fixed number of processors M, we varied $\sum U_i$ between $M/10$ ($Utilization = 0.1$) and M ($Utilization = 1$).

B. Simulation Results

We performed simulations for the four approaches, which include the two post-DVS approaches (PHD+DVS and SPA2+DVS), and the two pre-DVS approaches (DVS+PHD and DVS+SPA2). Each utilization and processor number level in the results corresponds to simulations of 10,000 task-sets. The results shown are average numbers from 10,000 simulations.

1) Schedulability: Fig. 2 shows simulation results of schedulability for all the approaches. Schedulability stands for the possibility that a task-set with determinate total utilization is schedulable to a determinate number of processors under an algorithm.

From Fig. 2, we can see clearly that all the algorithms have a schedulability of 100% when the average utilization is under 70%, which corresponds to the utilization bound given by previous work – 65% for PHD and 69.3% for SPA2. However, the two scheduling algorithms show a remarkable difference when the average utilization exceeds 70%. SPA2 does not perform so well as its utilization bound, due to its severe restrictions set by Liu & Layland's Utilization Bound [5] during scheduling. On the contrary, PHD has a much better schedulability at high utilization because it fills every processor as full as possible without unnecessary restrictions. Of course, it is difficult for either algorithm to schedule most of task-sets when the total utilization equals to the number of processors. So almost all the algorithms' schedulability decrease to zero when the average utilization reaches 100%.

In addition, we realize that the time when we perform DVS (either pre-DVS or post-DVS) does not make much difference on schedulability. After all, pre-DVS will try all the frequencies until it is schedulable or the frequency becomes maximal as post-DVS does. So any task-set that is schedulable with post-DVS algorithms could be schedulable with pre-DVS algorithms as well. On the other hand, pre-DVS algorithms try to reduce the frequency by prolonging the time a task executes. So it is hard to find a task-set schedulable with pre-DVS while unschedulable with post-DVS, although it does exist in rare instances. As a whole, the schedulability of pre-DVS and post-DVS are at approximately the same level, as shown in the simulation results.

2) Power: Fig. 3 shows the power numbers for all the approaches. The power numbers in our results are normalized to the power under a processor's maximal frequency.

From Fig. 3, we notice that when the average utilization is close to zero, all the algorithms' power numbers reduces to zero as expected, because there are few tasks to execute and DVS just reduces the frequency to zero to save energy (of course this is the ideal case because). When the average utilization comes to 1, all the algorithms' power numbers reaches the number of processors, because all the processors have to be full and keep at maximal frequency as long as the task-set is schedulable.

When the utilization gets close to about 50%, PHD with post-DVS gradually shows much higher energy consumption compared to the other three techniques. This is because PHD greedily assigns tasks to as few processors as possible and just keeps other processors idle, while DVS could not take effect when a processor is completely full or empty. As a result, the power of PHD with post-DVS always approximately equals to the task-sets' total utilization, showing an almost linear relationship.

As to the other three algorithms, they have different ways to schedule the tasks evenly to all the processors, and therefore
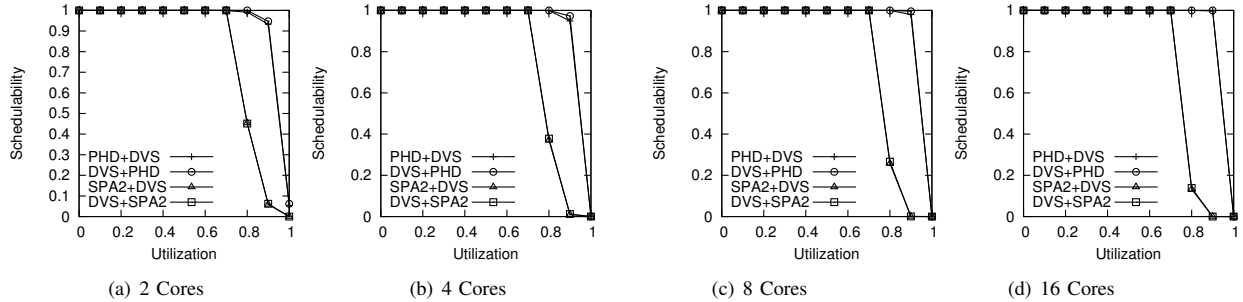
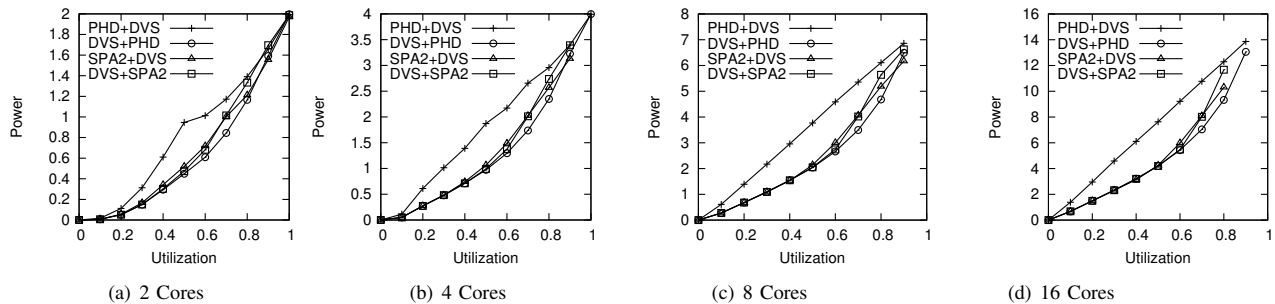Fig. 2.   Schedulability simulation results

Fig. 3.   Power simulation results

achieve a considerable energy saving. The SPA2 algorithms assign tasks in a width-first way. Pre-DVS algorithms increase all the tasks utilization as high as possible, so that they have to fill all the processors.

In all the cases, we found that PHD with pre-DVS achieves the most energy savings, while PHD with post-DVS achieves the lease energy saving. For example, for the 8-core case with 70% utlization, PHD with pre-DVS reduces energy by 59.6%, compared to only 43.0% savings for PHD with post-DVS. The benefits of the other two techniques (SPA2 with pre-DVS and post-DVS) are roughly 50%. We believe it is because pre-allocation of frequencies and a depth-first way assigning with task-splitting produce the most balancing scheduling, thus taking most advantages of all the processors.

From the simulation results, we can see that it is practical to apply energy-saving techniques such as DVS to multi-core scheduling algorithms with task-splitting. Although all the four approaches we have studied could save considerable power consumption with DVS, the PHD scheduling algorithm with pre-DVS shows both excellent schedulability and energy savings among all the approaches.

VI. CONCLUSION

In this paper, we have explored the possibility of combining dynamic voltage (frequency) scheduling with semi-partitioned fixed-priority multiprocessor scheduling with task-splitting for real-time systems. We proposed two different techniques to apply the DVS algorithm to multi-core scheduling approaches with task-splitting features. The techniques proposed include performing DVS after scheduling (post-DVS) and performing DVS before scheduling (pre-DVS).

We simulated the proposed techniques under different processor setups. Simulation results show that it is possible to achieve significant energy savings with DVS while preserving the schedulability requirements of real-time schedulers for multi-core processors.

REFERENCES

[1] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 113.2–, Washington, DC, USA, 2003. IEEE Computer Society.

[2] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low power cmos digital design. *IEEE Journal of Solid State Circuits*, 27:473–484, 1995.

[3] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with liu and layland's utilization bound. In *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '10, pages 165–174, Washington, DC, USA, 2010. IEEE Computer Society.

[4] K. Lakshmanan, R. Rajkumar, and J. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 239–248, Washington, DC, USA, 2009. IEEE Computer Society.

[5] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20:46–61, January 1973.

[6] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 89–102, New York, NY, USA, 2001. ACM.

[7] M. Weiser, B. B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *OSDI*, pages 13–23, 1994.