Available online at www.sciencedirect.com

SciVerse ScienceDirect

Physics Procedia 33 (2012) 188 – 194

Physics Procedia

2012 International Conference on Medical Physics and Biomedical Engineering

Power Management and Tasks Scheduling Analysis in Power-Aware Real-Time Systems

Jingyu Xing[1,a], Zhang Feng[2,b]

[1]School of Software, Nanyang Institute of Technology, Nanyang, Henan Province, China
[2]School of Software, Nanyang Institute of Technology, Nanyang, Henan Province, China
[a]jing8089@163.com, [b]yeyeqiuyu@tom.com

Abstract

Power aware computing has become popular recently because of the vast application of the portable systems. Many mechanisms have been proposed to manage energy consumption based on the dynamic voltage scaling. In this paper, we give a detailed analysis about static, dynamic power management and scheduling techniques to reduce energy consumption of tasks that have deadlines. We also study the overhead in the system when changing processor speed. Take into consideration this overhead, we decide whether slow down the processor speed or not.

Keywords: Real-Time Systems; Power Management; Scheduling; Voltage Adjustment

1. Introduction

High power consumption has been a critical design factor in portable and hand-held computing systems in recent years. It not only decreases the lifetime of battery operated system, but also generates more heat, which causes heat dissipation to be a big problem. Because in many large systems, such as，complex satellite and data warehouses, the cost of energy consumption and cooling is substantial. Many techniques and algorithms have been proposed to resolve this problem based on dynamic voltage scaling. It is a piece of good news that processors with multiple supply voltage have become available in recent years.

DVS(Dynamic Voltage Scaling)[1] was proposed based on comparing the convex relation between the supply voltage and the CPU power consumption. It could obtain quadratic energy saving just because the DVS mechanism roughly linearly increased task' response time. However, there is one aspect that must to be carefully taken into consideration is that changing the power available to tasks may lead to violation of timing constraints and other undesirable consequences.

 doi:10.1016/j.phpro.2012.05.049

For systems under timing or energy constraints, there is a way to minimize CPU energy consumption when meeting the deadlines, by adjusting the processors' supply voltage and clock frequency. Many of embedded real-time systems operate in a cyclic basis, a set of tasks must execute within a frame whose execution is to be repeated. The frame consists of a set of tasks and all of frames have a common deadline. In this paper, assuming tasks execute in a frame and any two of tasks have no precedence constraints.

The rest of the paper is organized as follows: the system and energy model is described in Section 2. Static power management and static scheduling is addressed in Section 3 and 4, respectively. Section 5 discusses the dynamic power management and tasks scheduling. The system overhead is analyzed in Section 6. Finally, Section 7 gives a conclusion.

2. System and Energy Model

We assume that each real-time task t_i, has a deadline D_i, which is derived from the system's real-time requirements. Each task t_i has the estimated worst-case execution time W_i and the actual execution time a_i based on maximal processor speed S_{max}. In the static power management, the value of W_i for each task t_i is known before execution. The number of processor cycles required by t_i will be denoted by C_i. For a specific architecture, we assume that the number of cycles to execute a task is independent of the processor speed. Frame-based systems are special periodic real-time systems and each frame consists of a set of tasks, $\{t_1, t_2, \ldots, t_i\}$. The deadline of frame F, is the least common multiple of deadline of tasks $\{t_1, t_2, \ldots, t_i\}$. The power consumption for CMOS processor is dominated by dynamic power dissipation P_d, which is given by:

$$P_d = C_{ef} * V_{dd}^2 * S \tag{1}$$

Where C_{ef} is the effective switch capacitance, V_{dd} is the supply voltage and S is the processor clock frequency, that is, the processor speed [2]. P_d is a strictly increasing convex function of the supply voltage. The processor speed is characterized by:

$$\mathrm{S}= k * \frac{\left(V_{dd}^2 - V_t\right)^2}{V_{dd}} \tag{2}$$

Where k is a constant, V_t is the threshold voltage [2]. It is almost linearly related to the supply voltage. A specific task's execution time is C_i/S, where S is the processor speed when execute this task. So the energy consumption of this task E, will be

$$\mathrm{E}= P_d * \frac{C_i}{S} = C_{ef} * V_{dd}^2 * C_i \tag{3}$$

This formula illuminates that when decreasing process supply voltage (process speed), we reduce the energy consumption of this task.

3. Static Power Management

Static power management always assumes that each task presents its worst-case workload to the processor. In the context of power management through CPU speed adjustment, a power management point (PMP) is used to insert at the beginning of each program segment. A PMP consists of a piece of

code. PMP code decides the execution of the tasks in the system and decides about changing the CPU speed. There are two type of PMPs, one is task-level PMP, the other is system-level PMP.

In the real-time systems, we should assure the stable and effectiveness of systems even in the worst-case scenarios. If in static power management, systems estimate the worst-case execution time of tasks W_i before their execution. If executing task t_i at its worst-case does not consume the entire time allocated to it, then we will use this static slack time to reduce the speed of executing t_i in order to reduce energy consumption and still meeting the timing constraints. At this time, the static slack time is the difference between the deadline of t_i and the worst-case execution time of t_i. PMP calculates processor speed S, based on static slack time and set processor speed to S, $S=C_i/d_i$. In this paper, we discuss frame-based systems. Given a frame, F, its deadline, D, the static slack time of frame is defined as the difference between the deadline of frame and the length of static schedule, that is, the difference between the deadline of frame and the finish time of the last task in this frame. We use $Slack_{static}$ to express this slack time.

For uniprocessor real-time system, if there are task A and B, their execution is as shown in Fig.1 ($A_{wi/di}$, W_i is the worst-case execution time while d_i is the deadline of task A) . Task's initial scheduling is as shown in Fig.2a.

There are two second static slack. If D=8, then $Slack_{static}$=8-6=2. This slack static can use for processor to reduce energy consumption. The final static scheduling based on static power management is as shown in Fig.2b.

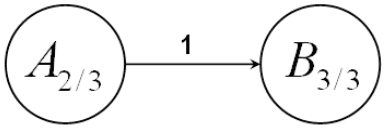

Fig.1 Task's execution for uniprocessor real-time system

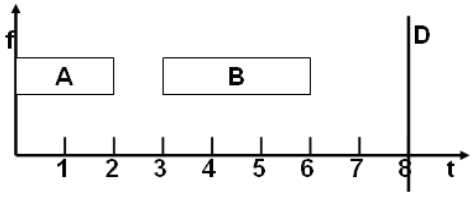

Fig.2a Task's initial scheduling for uniprocessor real-time system

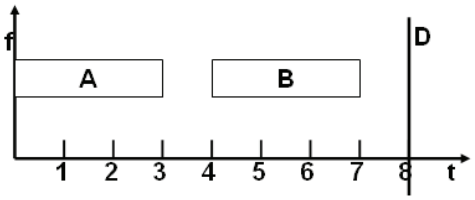

Fig.2b final static scheduling based on static power management

4. Static Scheduling

This chapter gives an example about how to use $Slack_{static}$ to perform task scheduling in a two-processor system. Assuming two of processor shares a memory and a task queue Q. We use the longest

task first heuristic (LTF) when determining task priority. There are three tasks $\{t_1(4,2), t_2(4,1), t_3(2,2)\}$ in the queue. The priority of the task determines the order of tasks in a queue. In general, the optimal solution of assigning task priority to get minimal execution time is NP-hard [3].

Assuming the deadline of the frame in Fig.3a is 8, S_0 is the initial processor speed. Fig.3a shows the initial scheduling of tasks.

In Fig.3a, the X-axis represents time, the Y-axis represents processor speed, the area of the task box represents the number of CPU cycles needed to execute the task. The static slack time in frame $Slack_{static}$ =8-6=2.

4.1 Greedy Static Algorithm:

This algorithm collects the static slack time $Slack_{static}$ and allocates the entire $Slack_{static}$ to the first task in the frame on each processor. The saving of energy just results from the speed adjustment of the first task on each processor. Applying two $Slack_{static}$ to the tasks in Fig.3a, both task t_1 and t_2 will get 2 units of slack time. After performing greedy algorithm scheduling, the speed of task t_1 and t_2 will reduced to $2/3S_0$ while the task t_3 still executes at speed S_0. The static scheduling is shown in Fig.3b.

4.2 Parallel Static Algorithm:

From above discussion, we observe that greedy static algorithm is not an optimal algorithm for multi-processor systems. The one reason is that it always allocates all the $Slack_{static}$ to the first task on each processor in the system. When many of tasks exist in system, the others tasks execute at their initial speed and have not any speed adjustment, moreover, the speed of processor has a bottom line in order to

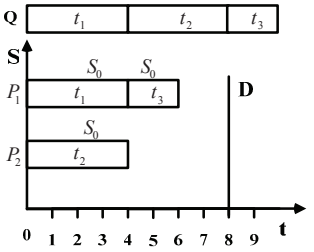

Fig.3a the initial scheduling of tasks

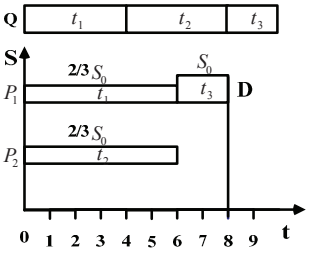

Fig.3b Greedy Static Algorithm

maintain system stable. So it is impossible to reduce the speed of the first task on each processor but without any constraints. The other problem of the greedy static algorithm is that it doesn't take into

consideration the degree of parallelism of tasks when allocating slack. Then we present another new algorithm, parallel static algorithm [4].

For an N-processor system, we define the degree of parallelism P_a is: if there are i processors in the executing scenario at time t, then P_a =i. P_a will range from 0 to N in N-processor system. In Fig.3a, P_a =2 from 0 to 4; P_a =1 from 4 to 6; P_a =0 from 6 to 8. Using L_i represents the total scheduling time length when P_a =i. In Fig.3a, L_0=2, L_1=2, L_2=4. Seg_i represents the segment when P_a =i and L_i represents the slack time allocated to Seg_i by this algorithm. In Fig.3a, $Slack_{static}$ =2. First, the algorithm calculates $Slack_{static}$ in the system based on the estimate worst-case execution time. Then divides $Slack_{static}$ into Δm common segments and each of length h= $Slack_{static}$ / Δm . The algorithm next allocates h to segments through a loop until there haven't h to allocate.

5. Dynamic Power Management and Tasks Scheduling

Defining S_s is the optimal speed obtained by static power management. Though static power management can be shown to be optimal under a worst-case workload, it is known that in many cases the tasks of real-time systems start and complete earlier than the worst-case scenario [5]. When the tasks don't execute at the worst-case scenarios, dynamic slack time will be generated in system. The first task of PMP is to calculate this slack. If a task t_i at the worst-case scenario reached at time t_{wc}, its actually reached at time t_{ac}($t_{ac} \le t_{wc} \le d_i$), then the difference between t_{wc} and t_{ac} can be considered as dynamic slack time $Slack_{dynamic}$, assuming task t_i execute at its optimal speed S_s obtained by static power management.

5.1 Greedy Dynamic Algorithm:

In this algorithm, defining t_{wc}^i and t_{ac}^i is the worst-case reach time and actual reach time of the i^{th} task, respectively. This $Slack_{dynamic}$ can be totally used to slow down the execution speed of the next task t_{i+1}.

In Fig.4, $Slack_{dynamic}=d_i-t_{ac}-C_i/S_s= d_i-t_{ac}-(d_i-t_{wc})=t_{wc}-t_{ac}$. By using $Slack_{dynamic}$, the processor execution speed of next task t_{i+1} can be adjusted in order to reduce energy consumption. However, the execution speed adjustment must take into consideration two constraints. One is the lowest speed of processor, S_{min}, and the other is that speed adjustment should guarantee all tasks meet their deadlines.

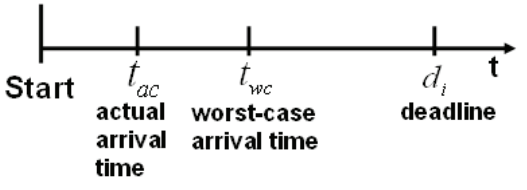

Fig.4 Greedy Dynamic Algorithm

We characterize the $Slack_{dynamic}$ of the i^{th} task as $Slack^i_{dynamic}$ and the execution time of the i^{th} task as T_i. Adding $Slack^i_{dynamic}$ to the task t_{i+1}, the execution time will be:

$$T_{i+1} = C_{i+1} / S_s + Slack^i_{dynamic} \qquad (4)$$

Subject to:

$$T_{i+1} \le d_{i+1} - t_{ac}^{i+1}$$

The execution speed will be:

$$S_{i+1} = C_{i+1} / (Slack^{i}_{dynamic} + C_{i+1} / S_s) \quad (5)$$

Subject to:

$$S_{i+1} \geq S_{\min}$$

In this paper, every task in frame has its own deadline, so we can say that frame F meets its deadline only when all of its tasks meet their deadlines. Dynamic power management is based on the assumption that every task executes at their average case scenario and greedy dynamic algorithm gives all the slack to the next task for slowing down its speed. Dynamic power management mechanism is more reasonable than the worst case behaviour.

6. System Overhead

When changing the speed/voltage of processor there are two kinds of overhead: time overhead and energy overhead. These overhead were ignored in the above work. However, time overhead affects the feasibility of scheduling. If time overhead is too long then tasks will miss their deadlines. Energy overhead is another can't be ignored part. If energy consumption after slowing down the speed of CPU is larger than the normal energy consumption, then we should decide not to run this task at a lower speed.

6.1 Time Overhead:

Some systems can continue operation while changing speed and voltage [6,7]. We assume that the processor can't execute task code during this period.

Time overhead consists of two parts: one part is for processor to compute the new speed and set up to it, suppose this part to be a constant part, H; another part is for changing speed from current speed to the new speed, it is proportional to the steps of voltage/speed adjustment. We assume that a fixed time, T_f, is needed for each speed step transition. Hence:

$$Time_{overhead} = H + T_f * g\left(S_{current}, S_{new}\right)$$

where $S_{current}$ denotes the processor's current speed and S_{new} denotes the new speed after adjustment, respectively. $g(S_{current}, S_{new})$ is a function that returns the numbers of speed steps between $S_{current}$ and S_{new}. Because we don't know S_{new} before tasks execution, we use a conservative way to estimate this $Time_{overhead}$, that is, $Time_{overhead} = H+T_f*g(S_{max}, S_{min})$.

By adding this maximum time overhead to the static scheduling:

$$Slack'_{static} = Slack_{static} - Time_{overhead}$$

To the dynamic scheduling:

$$Slack'_{dynamic} = Slack_{dynamic} - Time_{overhead}$$

6.2 Energy Overhead:

Besides the time overhead, there is energy overhead associated with speed adjustment. Energy overhead consists of two parts either: one part is for computing the new speed and another is for changing the speed. Suppose they are E_c, E_n , respectively. Hence:

$$Energy_{overhead} = E_c + E_n$$

Assuming E_i is the energy consumption before adjustment and E_i' is the energy consumption after the adjustment. If $E_i \le E_i' + Energy_{overhead}$, then it is not reasonable and necessary to adjust processor speed even if the timing constraints of tasks can be met.

7. Conclusion

In this paper, we have introduced and described the power management techniques of power aware real-time systems. The static power management is based on the worst-case workload offered to the system. Through estimating the worst-case execution time before deadlines tasks get $Slack_{static}$ (if any) and use it to slow down its execution speed. We give two scheduling algorithms based on static power management and make a comparison of their characters. Dynamic power management is based on the actual workload offered to the system. If task actually arrival time earlier than its worst-case scenario, then system generate $Slack_{dynamic}$ and slow down the processor speed to reduce energy consumption. We also give a scheduling algorithm based on dynamic power management. Finally, we discuss the system overhead.

In this paper, we don't discuss the dependencies between tasks and the recharging model of systems. When tasks have different software versions, how to schedule tasks can achieve a high QOS is another problem deserving further study. Thus, in the future, we will extend our work in these ways.

References

[1] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," In Proceedings of USENIX Symposium on Operating Systems Design and Implementation, 1994.pp203-204.

[2] Dakai Zhu, Daniel Mosse, Rami Melhem, Power Aware Scheduling for AND/OR Graphs in Real-Time Systems. Proc. of the International Conference on Parallel Processing (ICPP), Vancouver, B.C. (Aug. 2002).pp115-117.

[3] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Trans. On Software Engineering, 15(12):1497–1505, 1989.

[4] Ramesh Mishra, Namrata Rastogi, Dakai Zhu, Daniel Mosse and Rami Melhem. Energy Aware Scheduling for Distributed Real-Time Systems. Proc. of the International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France (April 2003).pp19-20.

[5] R. Ernst and W. Ye. Embedded Program Timing Analysis based on Path Clustering and Architecture Classification. In Computer-Aided Design (ICCAD)'97. pp. 598-604.

[6] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lpARM Microprocessor System. International Symposium on Low Power Electronics and Design 2000, pp.96-101, 2000.

[7] Hong, G. Qu, M. Potkonjak and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS'98), Madrid, December 1998.pp220-221.