International Journal of u- and e-Service, Science and Technology
Vol. 2, No. 1, 2009

A practical dynamic frequency scaling scheduling algorithm for general purpose embedded operating system

Chen Tianzhou, Huang Jiangwei, Zheng Zhenwei, Xiang Liangxiang
College of computer science, ZheJiang University
ZheDa road 38#, Hangzhou, China, 310027
{tzchen|hjw|zhengzhenwei|lxxiang}@zju.edu.cn

Abstract

Dynamic frequency scaling (DFS) techniques for real-time embedded systems have been widely studied. However, most of the scheduling algorithms so far concern only special purpose real-time systems. In this paper we devise a power-aware parameter task model for time-sharing real-time systems of general purpose, as well as a scheduling algorithm that is based on the task model. We implement the model and corresponding algorithm on embedded Linux. With the energy consumption measurement method presented in this paper, the experiment shows that for real-time tasks in Linux, 31.7% power consumption can be saved compared with the circumstance under which the processor runs at highest frequency.

1. Introduction

Mobile and embedded systems have developed rapidly. Different from traditional desktop systems, embedded devices demand not only high processor performance but also low power consumption. Since these devices are usually battery-driven, a proper method in power consumption management will extend the battery life significantly. Efficiency in power consumption requires careful design for both hardware and software.

Often, embedded application should be real-time, which means a task must complete its execution before some deadline. A hard real-time system does not permit any failure in meeting the deadline, while a soft real-time system has a relatively weaker constraint. As stated above, in order to meet the critical power requirement in embedded systems, various power-aware algorithms have been proposed.

Dynamic voltage scaling (DVS) was first introduced by [1]. The idea of being power-aware in execution gave birth to DVS processors. In [1], the best scheduling principle bases on a region spanned by a given task specification. A task is scheduled only when it can be run at the lowest processor frequency.

The voltage and frequency of a processor has a linear relationship, and processor scaling is usually via frequency scaling, so we will refer to scaling frequency (DFS) as the DVS method in this paper. There are two power-saving situations: one is that no urgent real-time task exists and the other is that the processor is executing memory-bound instructions. Formula 1 illustrates the power-saving reason:

$$P_{dynamic} \propto \alpha C V^2 f \quad (1)$$

α and C are constants, V denotes the voltage and f denotes the processor frequency. It is easy to see that reducing the processor frequency has a quadratic influence of the voltage while has a cubical influence of the power, due to the linear relationship of voltage and frequency of a processor.

Various DFS algorithms have been proposed for real-time systems [2] [3] [4]. However, most of them are not designed for general purpose time-sharing real-time embedded systems. In this paper, we design a DFS algorithm for general purpose systems and implement it on embedded Linux. The measurement method is also provided. We devise a parameter task model for original real-time task model, and the parameters can be used to indicate the task's emergency and memory bound level.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 describes the task model and algorithm. Section 4 presents the experimental method. Section 5 shows the corresponding results. Section 6 is the conclusion.

2. Related work

DFS reduces the processor power consumption by scaling down the processor frequency. The scaling policy falls into two categories [5] according to frequency adjustment: task-based and interval-based. Inter-task and intra-task DFS are two usual approaches of task-based DFS. The inter-task approach determines different frequencies for each task, and it is inappropriate for a general purpose time-sharing embedded system to consider the whole workload. The same problem applies to the intra-task approach. [6] proposed an event-driven frequency scaling for every task. [7] provided a workload decomposition method for a task according to CPU-bound or memory-bound instructions to scale down the frequency when memory-bound instructions are executed. The interval-based approach evaluates the processor workload over time interval and adjusts the processor frequency accordingly. The interval-based approach is suitable for time-sharing scheduling, so we adopt this approach in our algorithm.

Usually, there are two situations where DFS can be used to save power: No urgency real-time task exists, and Memory-bound instructions are executed. In situation 1 where all tasks are not urgent, there is an opportunity to make tradeoff between performance and energy. In situation 2 where the processor is executing memory-bound instructions, the high frequency clock time is wasted when the processor is waiting for the accomplishment of memory operations. Therefore, the processor frequency can be reduced to save energy without affecting the performance. [8] measures different application performance while scaling the processor frequency, which illustrates situation 2 experimentally.

Most works implement DFS algorithms on a special purpose real-time system or simulator. [3] proposed a DFS method for multitasking real-time systems with uncertain execution time via a probability-based DFS. They assumed a period task model, which is not the common case in a general purpose embedded OS. [4] considered the system level power-aware scheduling, but adopted the period task model also. [2] provided an energy-aware kernel for hard real-time system, using a hard real-time period model. The kernel is implemented on ecos. [7] decomposed a task into CPU-bound and memory-bound parts and implemented the DFS algorithm on a general purpose OS. However, task level optimization is not suitable for time-sharing soft real-time systems.

Most of those works has the assumption that the task set is a periodical work, but it is not practical for a time-sharing system. Although [7] succeeded in a general purpose system, however it is not for wide power-aware scheduling.

We hereby devise a new task model for time-sharing soft real-time systems of general purpose. In this model, each task has a parameter to define its emergency and memory bound

level. We also propose a simple but practical and powerful scheduling algorithm based on the model. The task model and scheduling algorithm are implemented on embedded Linux.

3. Task model and algorithm

3.1 Task and System Model

The real-time model proposed in previous works is described as follows. There are a group of tasks $\{T_1, T_2, T_3, \ldots, T_n\}$. Each task has its own arrival time (b), deadline (e), and the reserved processor time for the maximum processor frequency (p). Tasks are independent and have no period. When a task T_x is added to the waiting queue, the arrival time (b) is set. It is also assumed that there is no resource constraint.

This task model, however, is appropriate only for a special purpose real-time system, which deals with single and stable task sets. When it turns to a general purpose time-sharing embedded OS, a task's arrival is uncertain and some scheduling policy should be used to switch tasks. This model thus can not deal with this situation very well. We modify this model into a suitable one so that the problem mentioned above can be solved. We call this method the parameter task model.

In the parameter model, there is no candidate task set. We consider the arrival time (b), deadline (e), and processor time (p) as a single parameter (pf). The pf is the ratio of the required frequency to the highest processor frequency. The required frequency is the minimum frequency to finish a task in an entire e-b time interval. For example, if a task T_x has a pf of 0.1 and occupies the processor from b to e, T_x can finish its deadline as long as the processor frequency is set as 1/10 of the highest frequency. Obviously, we have Formula 2:

$$pf = \frac{p}{(e-b)} \tag{2}$$

In Formula 2, the pf summarizes these three original parameters and can be used to indicate the task's emergency and memory bound level.

A task T_x can notify the OS scheduler that it is not an urgent real-time task by claiming a smaller pf, which satisfies the power saving situation 1. When a memory-bound task arises, the task can set down the pf to get a low processor frequency, which cooperates with the memory instructions for energy optimization.

3.2 Algorithm Analyzing

Consider real-time tasks T_1 and T_2. We assume the two tasks run separately and each task can spread processor time from beginning to end with the lowest frequency requirement of 1/3f, as shown in Figure 1 (a) and (b). When T_1 and T_2 arrive simultaneously, the round-robin scheduler splits the whole time from beginning to end for each task, and scales the frequency to 2/3f so that it can meet both deadlines. We show it in Figure 1 (c). This illustrates the accumulation of each pf to determine the online processor frequency workload.

According to the algorithm, we do not scale down frequency as long as there are real-time tasks in the waiting queue. It is because the pfsum is not the sum of current waiting queue tasks pf, and the sum may include the already finished tasks pf.

In Figure 2 we suppose t is the time of T1 from beginning to end, T2 requires last 1/3t, and their minimum frequency requirement is 1/3f. When T2 arrives, the scheduler re-calculates the pfsum, and scales the frequency to 2/3f if the round robin chooses T1 to continue. After T1 is finished, if the pfsum is re-calculated as the sum of waiting queue tasks pf, the frequency is scaled down to 1/3f, and T2 will fail to meet its deadline. In our algorithm, when we scale up the frequency, we take T2 into account. If round robin chooses T1, the previous frequency has to be maintained when T1 is finished.

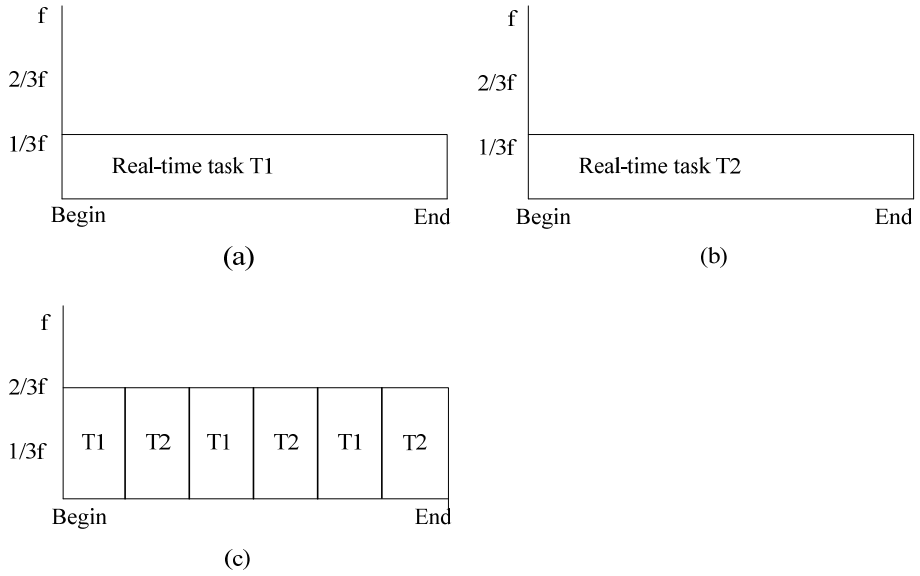

Figure 1. The relationship between pf and time-sharing scheduling

If real-time tasks continuously arrive, the value of pfsum will keep growing; in that way the scheduling is not energy optimistic. However, in the general purpose time-sharing situation, it becomes appropriate practically.

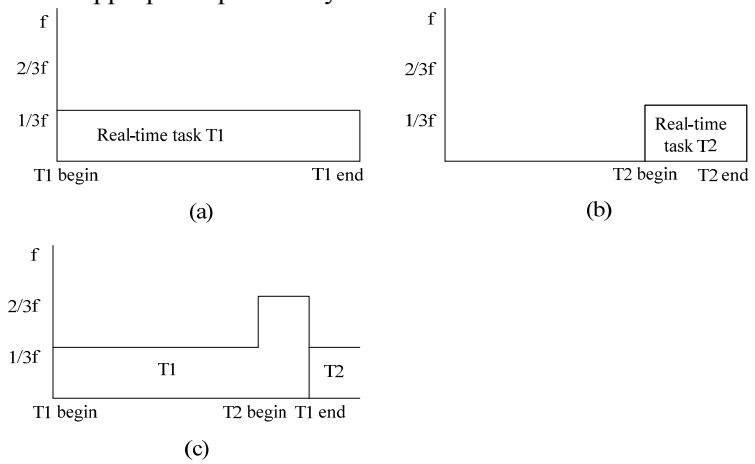

Figure 2. When T2 arrives, the frequency is scaled to 2/3f. After T1 is finished, the frequency remains 2/3f in order to finish T2 before deadline.

When a burst out of real-time task arrivals occurs, the algorithm reduces the scaling of frequency which may cause energy waste. But in this situation, system performance is guaranteed.

When a real-time task declares its deadline, usually a slack time bound is presented. Actually, it is why power saving situation 1 works. In a general case, the real-time task will not occupy the system for a long time. The processor frequency will be scaled to the lowest for power saving as long as there is no real-time task. In our algorithm, we use the accumulation pfsum. Due to the nature of time sharing systems, we can only provide a soft real-time guarantee.

In a general purpose system, we have less knowledge of the task information. In this condition, we make the practical parameter assumption to design our algorithm, which can be a basic implementation for optimization and extention. In our experimental implementation, the processor frequency is discrete. In the next section, we present our experiment.

4. Experimental methodology

4.1 Hardware

We implement the proposed algorithm on the Intel XScale PXA255 [10] platform. We modify the embedded Linux kernel (v2.4.17) to implement the parameter algorithm. The XScale PXA255 supports 7 frequency levels by writing the CCCR and CCLKCFG registers [10] that grant software the ability of scaling frequency during execution. Scaling core frequency will scale the bus, memory and SDRAM frequency at the same time. We choose 199.1M (with CCCR value 0x141), 298.6M (with CCCR value 0x1c1), 398.1M (with CCCR value 0x241) frequency, with the same bus, memory and SDRAM frequency to remove the disturbing factors. We will refer the three frequencies as 200M, 300M, and 400M in this paper.

4.2 Measurement Methodology

The power consumption measurement is based on [9]. [9] provided a power prediction method for Intel XScale processors using performance monitoring unit (PMU) events. They proposed a liner model (Formula 3) to compute power consumption.

$$\begin{aligned} Power_{cpu} = {} & \alpha_1(IFetch_{miss}) + \alpha_2(DataDep) + \\ & \alpha_3(DataTLB_{miss}) + \alpha_4(InstTLB_{miss}) + \\ & \alpha_5(InstExec) + K_{cpu} \end{aligned} \tag{3}$$

And they also provided α_1, α_2, α_3, α_4, α_5 and K_{cpu} parameters' values for different processor frequencies. We can use the PMU on the processor to record these events and predict the power consumption.

XScale can record processor events such as instruction cache miss ($IFetch_{miss}$), stall due to data dependency (DataDep), data TLB miss ($DataTLB_{miss}$), instruction TLB miss ($InstTLB_{miss}$) and instruction executed (InstExec), using PMU via setting PMNC register. But there are only two events that can be monitored simultaneously. In order to measure the five events, three same task scenes are required. The repeatability of the task scenes is unavailable in the Linux time-sharing scheduling model; a group of tasks can not be guaranteed when

having identical scheduling results each time. To overcome this drawback, we launch a group of real-time tasks and record the scheduling scene. When all tasks are finished, we use an identical task to simulate the group of tasks. At the same time we provide different pf values to simulate the parameter model scheduling scene. We can measure the simulator task for more times to monitor the events and give the result.

5. Results

We provide a group of tasks in Table 1 to test the modified scheduler. The value of pf is calculated from b, e and p by Formula 2.

Table 1. Prepared tasks group with parameters

Task	b	e	p	pf
T0	0	200	60	0.3
T1	50	100	30	0.6
T2	100	150	25	0.5
T3	250	475	22.5	0.1
T4	260	460	30	0.15

In Figure 4, there are totally 36 times when the scheduler selects a candidate to run on the processor. It shows the round robin policy. In Figure 6, processor frequency selection is presented, the 6th-15th can be considered as a real-time tasks burst. The task burst refers to not only the growth of the task count but also the real-time emergency level.

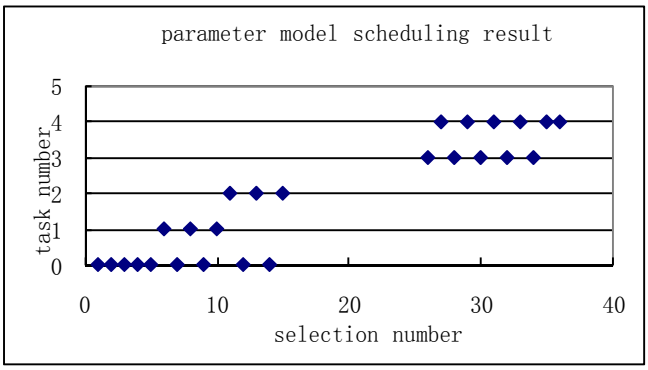

Figure 3. Scheduling sense for the given tasks group

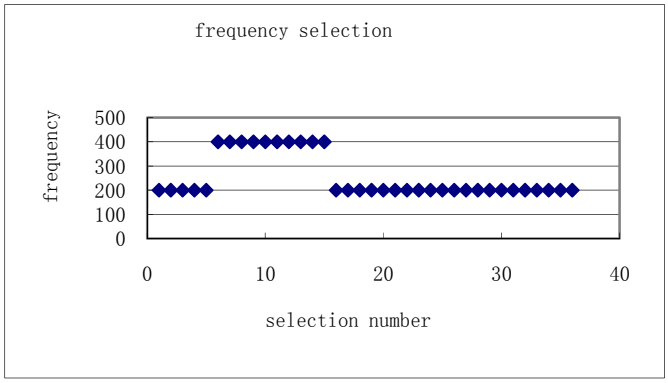

Figure 4. Processor frequency selection

Table 2 presents a real-time scheduling scene of our parameter model. The tasks require the processor frequency as the sequence 400M, 300M, 200M, 300M, 400M, 200M. We use our simulating system to generate a simulator task.

The measurement of the simulator task processor events is presented in Table 2. As a comparison, we consider the same workload simulator task which however requires the highest processor frequency. The monitored result is presented in Table 3. As a result, we acquire a (49.1065-33.5372)/ 49.1065=31.7% power saving.

Table 2. Simulator task processor events caught by PMU in different frequency

frequency	**IFetch** $_{miss}$	**Data Dep**	**Data** TLB_{miss}	**Inst** TLB_{miss}	**Inst Exec**
400	291	267	20	266	60106771
300	299	297	14	296	180325572
200	313	287	16	286	84166392
300	241	275	16	274	60111762
400	268	256	16	255	96169532
200	274	354	17	353	120235955
Power$_{cpu}$	33.5372				

Table 3. The processor events caught by PMU when the same simulator task is run without frequency scaling

frequency	**IFetch** $_{miss}$	**Data Dep**	**Data** TLB_{miss}	**Inst TLB** $_{miss}$	**Inst Exec**
400	403	373	16	372	601035523
Power$_{cpu}$	49.1065				

6. Conclusion and future work

In this paper, we present our power-aware parameter task model for a time-sharing real-time system of general purpose. In cooperation with the task model, we propose a scheduling algorithm and implement it on embedded Linux. We also propose a measurement method to evaluate the energy consumption. Future efforts can be put on the real-time parameter determination method. Time windows used to scale the pfsum value down is also needed for convenience.

References

[1]. Pering, T., and Brodersen, R. Energy efficient voltage scheduling for real-time operating systems. In Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session, Denver, CO, June 1998.

[2]. Goel, A., Krishna, C. M., and Keren, I. Energy Aware Kernel for Hard Real-time Systems. In CASES'05, September 24–27, 2005, San Francisco, California, USA.

[3]. Changjiu Xian, and Yung-Hsiang Lu. Dynamic Voltage Scaling for Multitasking Real-Time Systems with Uncertain Execution Time. In GLSVLSI'06, April 30–May 2, 2006, Philadelphia, PA, USA.

[4]. Ravindra Jejurikar, and Rajesh Gupta. Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems. In ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

[5]. Vasanth,V., and Michael, F. Power Reduction Techniques For Microprocessor Systems. ACM Computing Surveys, Vol. 37, No. 3, September 2005, pp. 195–237.

[6]. Weissel, A. and Bellosa, F. Process cruise control: event-driven clock scaling for dynamic power management. In Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems.ACM Press, 238–246.

[7]. Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Dynamic Voltage and Frequency Scaling based on Workload Decomposition. In ISLPED'04, August 9-11, 2004, Newport Beach, California, USA.

[8]. Singleton, L., Poellabauer, C., Schwan, K. Monitoring of cache miss rates for accurate dynamic voltage and frequency scaling. In: Proc. Multimedia Computing and Networking Conference, 2005.

[9]. Gilberto, C., and Margaret, M. Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events. In ISLPED'05, August 8–10, 2005, San Diego, California, USA.

[10]. Intel XScale® Core Developer's Manual Intel XScale® Core Developer's Manual January, 2004 Order Number: 273473-002

Authors

CHEN Tianzhou obtained his B.S. degree in computer software in 1994. He studied for M.S. degree in computer architecture, and received his PH.D degree in computer application from Zhejiang University in 1998. He is a professor of computer science at Zhejiang University. And he is the vice director of computer systems engineering research institute of Zhejiang University. He is a member of IEEE, and ACM. His current research interests include computer architecture, multi-core system, on chip interconnection, embedded system, power-aware computing, hardware/software co-design and security. He has authored and/or coauthored 12 textbooks, 161 research papers in major journals and international conferences in computer architecture area. He holds 27 Chinese Patents, 49 Software Copyrights.