Cooperative Voltage Scaling (CVS) between OS and Applications for Low-Power Real-Time Systems

Youngsoo Shin, Hiroshi Kawaguchi, and Takayasu Sakurai
Center for Collaborative Research,
University of Tokyo, Tokyo 106-8558, Japan

Abstract

Power efficient design of real-time embedded systems based on programmable processors becomes more important as system functionality is increasingly realized through software. This paper presents a cooperative power optimization method among OS, applications, and hardware platform. The hardware platform that consists of off-the-shelf microprocessor and custom-designed LSI is designed to support power-down and discrete levels of frequencies and voltages. In order to exploit these features, cooperative voltage scaling method is proposed, which is realized through design of power conscious OS and development of applications with concept of application slicing.

1 Introduction

Recently, power consumption has been a critical design constraint in the design of digital systems due to widely used portable systems such as cellular phones and PDAs, which require low power consumption with high speed and complex functionality. The design of such systems frequently involves reprogrammable processors such as microprocessors, microcontrollers, and DSPs in the form of off-the-shelf components or cores. Furthermore, an increasing amount of system functionality tends to be realized through software, which is leveraged by the high performance of modern processors. As a consequence, *power conscious design of software*, including operating system (OS) and application programs, as well as *supporting hardware platform* is important for the power-efficient design of such systems.

To reduce the power consumption of processors, two kinds of features are widely used: one is to bring a processor into a power-down mode when the processor is in an idle state, where only certain parts of the processor such as the clock generation and timer circuits are kept running; the other is to dynamically change the speed of a processor by varying the clock frequency along with the supply voltage when the required performance on the processor is lower than the maximum. Given a processor with these features, we are confronted with problems to *design application programs* and to *manage the mix of applications* in such a way that the power consumption is minimized while timing constraints imposed on the system is guaranteed.

In this paper, we propose a methodology for power conscious application design and OS development together with underlying hardware platform, with emphasis put on practical issues including implementation and feasibility of model. Specifically, as shown in Figure 1, we have a hardware platform consisting of off-the-shelf microprocessor and custom-designed LSI, which together provide power-down mode and discrete levels of speed (frequency and voltage). We are given a set of applications (or called tasks) with timing constraint imposed on each application, and OS that controls the execution flow of applications. Each application is modified in a way that it consists of a sequence of *slices* and some additional code fragment is inserted at the head of each slice. OS is also modified in a way that it maintains and provides timing information to applications, which is then used by each application to reduce the power consumption of the processor. The efficiency of the cooperation between OS and applications, which we call *CVS*, is verified through simulation with several examples.

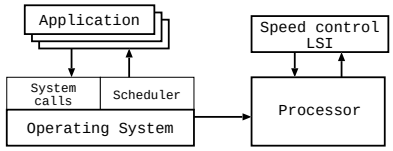

Figure 1. Structural view of a system.

The remainder of the paper is organized as follows. In the next section, we present our methodology to design power conscious applications and OS, together with issues encountered during implementation. In section 3, a hardware architecture to support our methodology is addressed. In section 4, experimental results are presented to evaluate the proposed method. Finally, a conclusion follows in section 5.

2 Cooperative Voltage Scaling

2.1 Model of CVS

As shown in Figure 1, software architecture consists of applications and OS. While each application (or application designer) has better knowledge of its own behavior, global information such as dynamic task interaction is only known to OS. This means that applications and OS should cooperate each other in order to exploit information toward reducing power consumption of processors. In our model of CVS, a set of applications[1] consists of real-time tasks, with each task associated with its period (the minimum inter-arrival time between successive requests in case of a sporadic task), deadline, and worst case execution time (WCET). The real-time tasks are scheduled according to *fixed priority preemptive scheduling algorithm* such as rate-monotonic scheduling (RMS) [1], although other scheduling algorithms can be used in our method.

2.2 Application Slicing

Power conscious application design, which we call *application slicing*, can be best explained with the help of Figure 2. An application is subdivided into a sequence of slices, with each

[1] *Application* and *task* are used interchangeably in this paper.

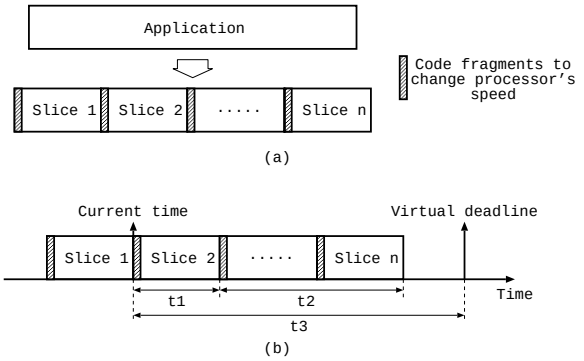

Figure 2. Application slicing. (a) Slicing structure. (b) Scheduling of slices.

slice having potentially different length, and small size of additional code fragment, which selects for each slice most appropriate speed in view of power consumption, is inserted at the start of each slice. The rationale for application slicing lies in the fact that we have no prior knowledge about future execution time of applications and the execution time of each application frequently deviates from its WCET, sometimes by a large amount [2]. Thus we gain more control over applications if they are sliced into many pieces. Although application slicing incurs a lot of engineering efforts, this can be done by application designers or middleware providers once and for all. Furthermore, recognizing the fact that processors are occupied mostly by highly demanding applications such as MPEG and the number of such applications is small in nature, the system designers can compose systems with their own custom applications, which can be designed either by application slicing or not, and sliced applications provided by middleware providers.

During run-time, the code fragment at the head of each slice computes the lowest processor speed to be used by the slice based on timing parameters. For example of `slice 2` in Figure 2(b), $t1$ denotes WCET of `slice 2` and $t2$ represents the sum of WCETs of the remaining slices. This means that $t1$ and $t2$ are static parameters obtained in the design stage, thus can be embedded in the code fragment itself. The most important parameter, $t3$, is the time difference between current time and *virtual deadline*, which is obtained from OS and to be explained in the next subsection. Thus, it is a dynamic parameter and obtained via system call. An example of the pseudo code fragment for `slice 2` is shown in Figure 3[2]. Because, time interval of $t3-t2$ is allowed to execute `slice 2`, the code fragment can compute the necessary speed taking various effects into account such as transition delay (T_d) to change speed and the speed of the previous slice[3]. It should be noted that $t3$ is determined in run-time due to two factors: although the speed used for `slice 2` is determined based on assumption that it takes WCET to execute `slice 2` ($t1$), `slice 2` may complete its execution earlier than its WCET meaning that start of `slice 3` may occur earlier than worst-case; virtual deadline itself varies even for slices in the same application due to

[2]In our implementation, there are two kinds of modes, which are to be explained in more detail in the next subsection. A task is either allowed to use virtual deadline or forced to complete its execution within its WCET.

[3]If the speed determined for the current slice is equal to that of the previous slice, there is no overhead of transition delay.

```
if virtual deadline is set then adjust-fq(t3-t2,t1);
else /* if forced to complete within WCET */
    Tav = WCET of slice 1 + t1 − executed time of slice 1;
    adjust-fq(Tav,t1);
end if

adjust-fq(t_i, t_j)
begin
    f_cur ← current frequency of processor;
    if f_cur == f_max/3 then
        if t_i ≤ 3t_j then set-fq(f_max/3);
        else if t_i ≤ 2t_j + T_d then set-fq(f_max/2);
        else set-fq(f_max);
        end if
    else if f_cur == f_max/2 then
        ...
    end if
end
```

Figure 3. Pseudo code of the code fragment at the head of `slice 2`.

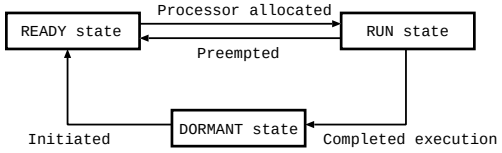

Figure 4. Task state transition.

preemptive scheduling.

2.3 Power Conscious OS

Although there have been extensive studies for OS-level control of the power consumption of processors [3], [4], [5], [2], [6], they have limitations in that dynamic nature of applications such as execution time variation is not fully exploited. There is also a method [7] to exploit application-level variation of execution time. However, it is limited to a single application. Our method, CVS, overcomes these limitations through interaction between power conscious OS and applications designed with the concept of application slicing.

Tasks are scheduled by OS based on their states as shown in Figure 4. If a task is in RUN state (called a run task for brevity), it currently occupies a processor. If it is in READY state, it is waiting to run meaning that some other task with higher priority is in RUN state. A queue, called *ready queue*, holds tasks in READY state in order of priority. If it is in DORMANT state, it has already run in its period and is waiting for its next period to start again. A queue, called *dormant queue*, holds tasks in DORMANT state in order of the time at which their release is due. When the scheduler (or dispatcher) is invoked, it searches the dormant queue to see if any tasks should be moved to the ready queue. If some of the tasks in the dormant queue are moved to the ready queue, the scheduler compares the run task to the task at the head of the ready queue. If the priority of the run task is lower, a context switch occurs.

The main function of power conscious OS other than basic operations described above consists of: providing virtual deadline to each task in such a way that *deadlines of all tasks are always*

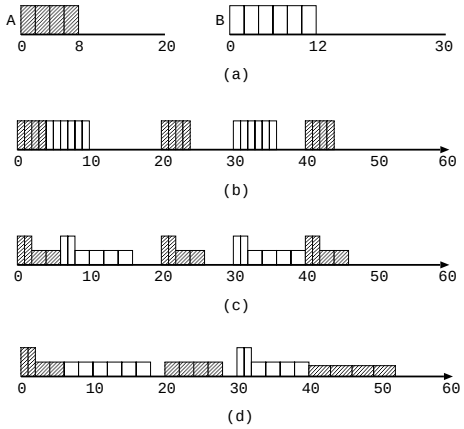

Figure 5. A schedule for the example of task set. (a) An example of task set. (b) Conventional rate-monotonic scheduling. (c) Scheduling with slice-level control of speed without interaction with OS. (d) CVS.

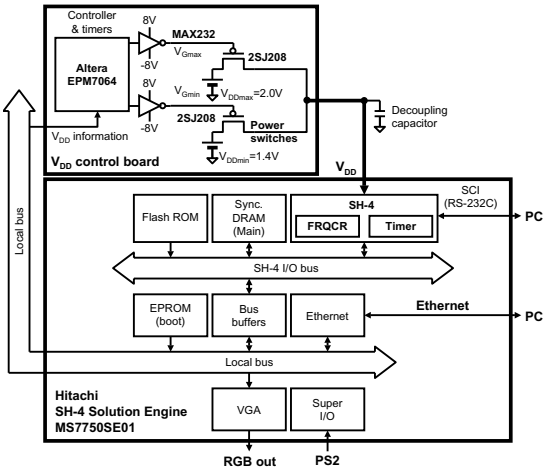

Figure 6. A block diagram of the hardware platform.

guaranteed; predicting the exact time interval during which there is no activity on the processor and bring the processor into power-down. This is done based on status of queues (ready queue and dormant queue). First, if the ready queue is empty but there is a run task (say task A) meaning that there is only one task that needs a processor, we set the virtual deadline of A to the minimum of deadline of A and release time of the task at the head of the dormant queue (recall that dormant queue is ordered by the next release time). Note that deadlines of all tasks are guaranteed with this approach. Second, if the ready queue is not empty meaning that there is a contention among tasks to take a processor, A is forced to complete its execution within its WCET. This is again conservative with respect to timing constraints. Note that there are still possibilities to lower the speed because some slices may complete their execution earlier than their WCETs thereby providing time margin to the subsequent slices. Third, if all tasks are in the dormant queue meaning that there is no task that needs a processor, we set a timer to expire at release time of the task at the head of the dormant queue and then put the processor into power-down mode. All these processes together with application slicing are illustrated in the following example.

Example 1 Consider the two tasks shown in Figure 5(a). Suppose that they consist of 4 and 6 slices, respectively, with each slice requesting 2 time units for its WCET. If we assume that period is equal to deadline, rate monotonic priority assignment is a natural choice meaning that A gets higher priority. A typical schedule, when each slice runs at half of its WCET, is shown in Figure 5(b). Suppose that there are three speed levels: 1, $1/2$, and $1/3$. CVS results in the schedule shown in Figure 5(d). At time 0, A is forced to complete its execution within its WCET at 8 because B is in RUN state. This is similar to having virtual deadline at 8. At time 6, A goes to DORMANT state. Thus, the virtual deadline of B is set to 20, which is the minimum of its deadline at 30 and the next arrival time of A at 20. The remaining schedule can be verified similarly. For comparison, Figure 5(c) shows a schedule when the method in [7] is extended to multitasking environment if proper support from OS is possible. □

2.4 Implementation of CVS

To realize CVS, we implement our method through customizing HI7750 [8] industrial real-time OS (RTOS). First, task control block (TCB), which is a data structure to contain task-specific information such as priority and start address, is modified to contain additional parameters used in CVS including period, virtual deadline, frequency used in the task, and so on. Second, several new system calls are added to support CVS. These include system calls to get virtual deadline, set or get the frequency of the processor, and so on. Third, scheduler, which resides in the kernel of RTOS, is modified such that it performs necessary action during task state transition. These include managing timing information in TCB, computing virtual deadline, bring a processor into power-down, and so on.

3 System Architecture

A block diagram of the hardware platform supporting the concept of CVS is shown in Figure 6. SH-4 [9] is used as a microprocessor. The average power consumed by a NOP instruction is about 70% of that consumed by a typical instruction, and the average power consumed by the processor when it is in power-down is 10% of the full power mode. In our current implementation, three speed levels are supported: 200 MHz with 2.0 V, 100 MHz with 1.4 V, and 66 MHz with 1.4 V. Thus, the speed can be changed by specifying clock frequency as the input parameter of a system call as shown in Figure 3, because voltage can be changed correspondingly within the system call.

The processor has a clock frequency control register, called FRQCR. The internal clock frequency, which is synchronized with external clock of 33 MHz, can be changed by accessing FRQCR. Because we use frequencies that are divisible with each other (200 MHz, 100 MHz, and 66 MHz), there is no synchronization problem with the external systems at the interface of the processor.

The V_{DD} information determined by the system call is sent via a local bus of the processor to FPGA (Altera EPM7064) residing on a *V_{DD} control board*. The function of the V_{DD} control board is also implemented in LSI and fabricated with 0.6-μm CMOS

technology. On the V_{DD} control board, V_{DD} is switched between V_{DDmax} (2.0 V) and V_{DDmin} (1.4 V) by power switch MOSFETs (2SJ208 x 2). Although the threshold voltage of the MOSFETs is one of the lowest available on the market, it is still too high (2.8 V) compared to the maximum supply voltage (2.0 V) of the processor meaning that they never turn on. To alleviate the problem, we use RS-232C driver (MAX232) as a voltage swing amplifier that amplifies the signal to ± 8 V. Although, the number of power switches is limited to two in our implementation, it can be increased at the cost of area. The measured fall and rise time of V_{DD} are less than 200 μs and 100 μs, respectively, with 30 μF decoupling capacitance at the node V_{DD}.

Because we use two switches to switch between two levels of V_{DD}, we ideally want to turn one MOSFET on and turn the other MOSFET off exactly at the same time, which is impossible. Thus, we have two cases with regard to two signals: V_{DDmax} enable signal (V_{Gmax}) and V_{DDmin} enable signal (V_{Gmin}). First, if there is an overlap between two signals, large current may flow from V_{DDmax} to V_{DDmin} thereby causing a problem. However, neither spike-like noise nor voltage drop is observed due to the decoupling capacitance. Second, if there is no overlap meaning that there is a potential period while V_{DD} line is completely cut off from both V_{DDmax} and V_{DDmin}, malfunction may occur in the processor. Thus, in reality, the switching between V_{DDmax} and V_{DDmin} should be carried out in a way that there exists a short period of overlap during which both V_{DDmax} and V_{DDmin} are connected to V_{DD} line. In our implementation, the period of overlap is set to 2 μs, and a programmable timer is put at the gate of the power switch in order to adjust the period of overlap. Another implementation issue is to assure the processor to be connected to V_{DDmax} when it is powered on. Thus, V_{Gmax} should be asserted to connect V_{DDmax} to V_{DD} line at boot up process.

4 Experiments

To evaluate CVS, we perform simulations with two examples and compare the average power consumption with CVS against that with RMS. In RMS, the processor executes NOP instructions when it is not being occupied by any task. The first example, called `vpda`, is constructed based on a task set in [10]. It consists of 4 tasks with two multimedia and two protocol applications. They are subdivided into 10, 15, 10, and 10 slices respectively. The second example [11], called `cnc`, consists of 8 tasks with each task subdivided into 20 slices.

Figure 7 shows the simulation results when we vary the execution time of each slice from 0.1 to 1.0 of its WCET. Power consumption is normalized with respect to the case when tasks are scheduled with RMS and they always execute in their WCETs. Power saving with CVS depends on the combination of periods of tasks, which in turn determines how much we can benefit from virtual deadline. It is also dependent on the extent of variation of execution time. Considering the fact that multimedia applications frequently have a lot of variations due to data-dependent execution, over-estimation of WCET, and so on, we can expect significant saving with CVS.

5 Conclusion

In this paper, we introduce a cooperative power optimization method involving design of power conscious OS, development of applications with concept of application slicing, and supporting hardware platform. Our method of voltage scaling, CVS, obtains a power reduction for a processor by exploiting the slack times inherent in the system and those arising from variation of execution times of task instances. We present a run-time mechanism to use these slack times efficiently for power reduction for a processor that supports a power-down mode and can change the clock frequency and the supply voltage dynamically. Experimental results show that CVS obtains a significant power reduction across several applications.

We are currently integrating all components: customized RTOS, applications developed with concept of application slicing, and supporting hardware platform developed for CVS.

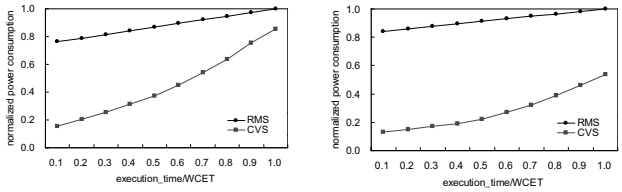

Figure 7. Simulation results of (a) `vpda`. (b) `cnc`.

Acknowledgement

This work was supported by Grants from Hitachi, Ltd. and the Japan Society for the Promotion of Science.

References

[1] C. L. Liu and James W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[2] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. Design Automat. Conf.*, June 1999, pp. 134–139.

[3] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. USENIX Symposium on Operating Systems Design and Implementation*, 1994, pp. 13–23.

[4] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE Annual Foundations of Computer Science*, 1995, pp. 374–382.

[5] C. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. Int'l Conf. on Computer Aided Design*, Nov. 1997, pp. 28–32.

[6] Y. Lee and C. Krishna, "Voltage-clock scaling for low energy consumption in real-time embedded systems," in *Proc. Int'l Workshop on Real-Time Computing Systems and Applications*, 1999.

[7] S. Lee and T. Sakurai, "Run-time power control scheme using software feedback loop for low-power real-time applications," in *Proc. Asia South Pacific Design Automat. Conf.*, Jan. 2000, pp. 381–386.

[8] Hitachi, Ltd., *HI7750 Hitachi Industrial Realtime Operating System for SH: User's Manual*, 1998.

[9] Hitachi, Ltd., *SuperH homepage http://www.superh.com/*.

[10] Y. Shin, D. Kim, and K. Choi, "Schedulability-driven performance analysis of multiple mode embedded real-time systems," in *Proc. Design Automat. Conf.*, June 2000, pp. 495–500.

[11] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin, "Visual assessment of a real-time system design: A case study on a CNC controller," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1996.