Increasing Appliance Autonomy using Energy-Aware Scheduling of Java Multimedia Applications

Parain Frédéric[1] Cabillic Gilbert[1] Banâtre Michel[1]
Higuera Teresa[2] Issarny Valérie[2]
Lesot Jean-Philippe[1]

(1) INRIA-IRISA, Campus de Beaulieu 35032 Rennes Cédex, France
{First.Lastname}@irisa.fr

(2) INRIA-Rocquencourt, Domaine de Voluceau, BP 105, 78153 Le Chesnay Cédex, France
{First.Lastname}@inria.fr

March 24, 2000

Abstract

Nowadays the appliance market is growing faster and faster. These computers are going to support in the near future new applications like high fidelity music player, video on demand, or video conference. In this context, real-time (due to multimedia applications) and autonomy are still main concerns for these embedded computers. To solve those problems, we introduce in this position paper a new approach based on the combination of real-time and power management in the scheduler of the operating system.

1 Introduction

Following the explosion of cellular phones, the appliance market is growing faster and faster. These computers are becoming now true PDAs with diary, phone, and notebook. Moreover, the incoming of wireless Internet connections opens the use of appliances. New applications like Web browser, high fidelity music player, video on demand, or video conference appear.

Those applications require intensive computation capacities, which are not yet available and imply an evolution of appliance systems (hardware and software). With regard to the hardware, the trend is towards heterogeneous multiprocessor to provide high performance at low power. Due to this architecture, the operating system executed on the hardware will perform real-time execution using several processors to support multimedia applications. To guarantee the time constraints, it will also use a strong power management policy to increase the appliance autonomy, which remains one of the major concerns on these embedded computers.

This position paper presents a new approach based on the combination of real-time and power management in the scheduler of the operating system. Section 2 provides the project context, with applications requirements and hardware platform description. Section 3 shows related works in software power management. Section 4 presents our approach, with a description of our scheduler and the profiling requirements. We conclude with a summary of our future plans.

2 Background

This paper reports ongoing work on software power management inside a new platform built as a possible solution to new appliances' requirements. This new platform supports a JVM[1] to accept dynamic loading of applications from a wireless Internet connection. Java was designed to provide this feature, it further offers security and hardware independent executable format, which are mandatory properties to accept applications from anywhere.

[1] Java Virtual Machine

The hardware platform was designed to have an important computation ability with a low power consumption. If monoprocessor solutions are simple, they often require a frequency increase to improve computation ability, which implies a voltage increase that further causes a consumption increase. On the other hand, multiprocessing allows increasing the computation ability without frequency or voltage modification, and hence has lower power requirements than the monoprocessor approach. Moreover, some results on hardware specific usage show that a specialized processor has a lower consumption than a generic one. This is why the hardware platform evolves towards a heterogeneous multiprocessor.

If a low power hardware platform is necessary to achieve a low consumption appliance, software power management can also improve autonomy. Whereas hardware solutions are restricted to static improvements, software management is able to act on dynamic distribution of processing, which has a direct influence on consumption in a heterogeneous environment. But distributed real-time scheduling of multimedia applications over a multiprocessor platform is already complex. This is why the introduction of power consumption concerns has to be done carefully because it impacts real-time as well as distribution management.

3 Related Works

A common software technique to decrease power consumption is the detection of idle periods. When a system has nothing to do during these periods, some of its devices can be switched to a sleep mode in order to reduce their energy consumption [8]. Although this consumption mode change is achieved through a hardware mechanism, the switch is software-driven. In general, this is achieved by the operating system, which uses timeouts to detect idleness. But several policies can be implemented to detect or predict [1] idle periods.

Power consumption can also be decreased by remote processing [9] using the wireless communications to deport computation on a remote server. To be an attractive solution, the remote execution cost must be lower than the local execution cost. The cost of a remote execution is composed of the communication cost to send data and computation instructions to the remote server and to get back the computation result to the appliance. Main problems in this solution are the availability of the remote server and estimations of communication cost and computation cost.

Multimedia applications are generally able to adapt their behavior to the availability of external resources, like network bandwidth, memory or CPU capacity. The work described in [3] shows that battery duration can be considered as an adaptation criteria for this kind of applications. In this case, the operating system is in charge of evaluating power consumption of applications, and the remaining energy available. Then, it is able to negotiate with applications to get a trade-off between quality and power consumption through degradation of multimedia data.

Another technique is voltage scaling. This technique is based on adaptation of the computation capacity of the processor to the current workload; this can be achieved by a regulation of the processor frequency [10]. This frequency regulation can be done by an inhibition of the clock signal on certain periods, and it is then called clock gating [7]. It can evenly be the result of a modification of the instruction given to the PLL[2], which controls the clock signal frequency. In this case, if the voltage supply can be adapted in the same time, this is called voltage scaling. The energy consumption can then be reduced to the bare minimum needed to complete computations before their deadlines. In the case of a real-time environment, voltage scaling implies scheduling algorithms adaptation, but several solutions have been already proposed in [5] and [4]. The main drawback of this frequency regulation is the stabilization period after a frequency modification. But it is possible to limit the number of supported frequencies without a significant impact on consumption decrease [2]. And this allows to keep a limited number of well-characterized points of operation with a well known stabilization period. We can notice in the above work, which aims at combining real-time and power management, that the scheduler becomes in charge of power management. That seems to be mandatory to keep a good real-time behavior.

[2] Phase Locking Loop

4 Our approach

Related work described above is mainly based on consumption reduction when the application workload decreases. Our approach is quite different but, at the same time, complementary to those proposals. It aims at fully exploiting energy characteristics of the hardware. This is why it uses a judicious distribution over the heterogeneous multiprocessor platform, by putting each task on the processor which will have the lowest consumption to execute it. This distribution alters execution times of tasks, so the power management has to be included inside the real-time scheduler of the operating system. The following sections describe the scheduling granularity, the working rules, and the profiling requirements for the time and energy consumption.

4.1 Granularity

We believe that the thread model proposed by Java is too weak for real-time applications. So we propose a new model that must be used by all applications loaded on the appliance. This model, only based on absolute temporal specifications, allows to describe real-time applications as if they were alone. In this way, several applications developed separately can be loaded and executed simultaneously. In practice, the scheduler manages elementary execution units, called tasks, defined in the model. A task has not exactly the same definition as a Java thread, but the transition from the Java thread model to the task model should not need a lot of source modifications.

A task is a code segment which is periodically executed, so it has an entry point in the bytecode[3] and a period. A task cannot migrate during one period, but its placement is not static; it can be different from one period to another period. A task assignment can be dynamically decided by the scheduler or be fixed once at the beginning because of the use of a processor-dependent native method.

4.2 Rules

The scheduler is in charge of providing a correct execution for all the applications running on the appliance, and has to do this at the lowest energy cost. An execution is correct if it completes without errors and before its deadline. Those constraints are modeled as rules that the scheduler has to respect when it schedules tasks. The efficiency of the scheduler depends on these rules. Next paragraphs describe mutual exclusion rules, real time and energy rules, and finally the complexity of the relationship between those rules.

4.2.1 Resources

To prevent corruption of shared objects, Java already provides monitors, which are a way to implement mutual exclusion between threads. But monitors do not have a predictable behavior, which is a serious problem to provide a real-time execution to applications. To avoid unspecified latencies in task executions, the scheduler is able to take into account mutual exclusions between tasks. The only condition is that each task declares objects it shares with other tasks. Those objects are called resources and the set of resources of a task is one of its attributes.

4.2.2 Real-time and energy

The application programmer has to specify a deadline and a period for each task of the application. Execution times are computed from profiles as described in Section 4.3. Real-time rules rely on this information to build a temporal distribution with the intention to complete all tasks before their deadlines. A best-effort policy has been chosen because multimedia applications can tolerate some deadline misses without serious damages. Nevertheless, the scheduler can detect applications whose tasks systematically complete after their deadlines, and takes action to fix this problem. The scheduling is made on-line to adapt to execution times variations.

[3] Java's executable code

The energy management is based on a spatial distribution of tasks; each task should be executed on the most adapted processor according to its type of computation. An adapted processor is more efficient, and thus has a lower consumption than a less adapted processor. The processor selection is done by comparison between energy consumption evaluations, as discussed in Section 4.3 .

The introduction of energy concerns can seriously impact the real-time scheduling. In most of on-line scheduling techniques, the task with the highest priority is executed as soon as a processor is idle. Nevertheless, a power management policy can prefer to delay this task execution, in order to place it on a more efficient processor, which is currently busy. This delay might be useful for energy preservation, but it might be dangerous for the real-time behavior too.

4.2.3 Principles and problems

The real-time scheduling of tasks with arbitrary execution times on a multiprocessor platform is a NP-complete problem. Our scheduler does not aim at being an optimal solution. It is elaborated from a simple non-optimal solution: the EDF[4] [6] algorithm, which we are gradually extending with a set of rules. Those rules improve the scheduler behavior on a heterogeneous multiprocessor and introduce energy concerns. Some of those rules, like the mutual exclusion rule, are simple to apply, but most of them are complex: the real-time rule is, in fact, a sorting by increasing deadlines, and the energy rule is a global optimization. This last rule is really a problem, because we want to decrease the overall consumption of the appliance, and local decisions taken for one task or one application are not necessarily judicious for a global optimization. Heterogeneity produces different execution times and energy consumption for each task on each processor. Then the scheduler has to perform a multi-criteria optimization. The complexity of solving algorithms of such an optimization are usually high, so it can cause a non-negligible cost, which must be taken into account. To be an attractive solution, our algorithm cost must be lower than the energy saving obtained through the computed distribution. A way to achieve this is to design an adaptive algorithm with an optimization degree, which depends on the consumption differences between processors.

4.3 Profiles

As said previously, the scheduler needs an evaluation of execution time and energy consumption to perform its job correctly. Java is an interpreted language; execution times as well as energy consumption depend on both the hardware and the JVM implementation. Time and consumption information included in the bytecode has to be platform-independent for two reasons. The first one is that it is against the Java philosophy, which advocates hardware-independence. The second one is that platform-dependent information implies one set of values for each existing platform. We believe that a profiling technique might solve this problem, which we are currently investigating.

5 Status

This position paper has reported on our current work. We are convinced that the scheduler is the key system component of power management on heterogeneous multiprocessor platforms. The application model described briefly in this paper is almost implemented and our current work focuses on the design of the profiling technique and scheduler rules. Regarding the latter issue, mutual exclusion rule is already completed, whereas real-time and power management rules are in good progress. First evaluations reflecting the effects of the scheduler rules, obtained through simulation, are promising. About the profiling technique, it is an ongoing work.

During the next step of this work we are going to implement the scheduler in a clean room multiprocessor Java virtual machine designed by our research team[5]. This prototype running on a heterogeneous multiprocessor will be useful to evaluate the actual impact of scheduling on the appliance autonomy, and to validate the real-time concerns of the scheduler.

Further research could go on to merge this approach with other power management mechanisms like energy quality of service negotiation or remote processing.

[4] Earliest Deadline First

[5] `http://www.irisa.fr/solidor/work/scratchy.html`

References

[1] Luca Benini and Giovanni De Micheli. *Dynamic Power Management, Design Techniques and CAD Tools*. Kluwer Academics Publishers, 1998.

[2] Anantha Chandrakasan, Vadim Gutnik, and Thucydides Xanthopoulos. Data driven signal processing: An approach for energy efficient computing. In *Proceedings of the 1996 international symposium on Low-Power Electronics and Design*, pages 347–352, 1996.

[3] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of th* 17^{th} ACM *Symposium on Operating Systems Principles*, 1999.

[4] Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Power optimization of variable voltage core-based systems. In *Proceedings of the* 35^{th} *annual conference on Design Automation Conference*, 1998.

[5] Inki Hong, Miodrag Potkonjak, and Mani B. Srivastava. On-line scheduling of hard real-time tasks on variable voltage processor. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 653–656, 1998.

[6] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), January 1973.

[7] Hidekiyo Ozawa and Seiya Kitagawa. Power management technology. *Fujitsu Scientific and Technical Journal*, 34(1):68–77, September 1998.

[8] Lorch Jacob R. and Alan Jay Smith. Software strategies for portable computer energy management. IEEE *Personal Communications Magazine*, 5(3):60–73, June 1998.

[9] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, January 1998.

[10] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *Proceedings of the First Symposium on Operating System Design and Implementation*, November 1994.