Proc of Globecom 2001

A Middleware Framework Coordinating Processor/Power Resource Management for Multimedia Applications*

Wanghong Yuan, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urban-Champaign
Urbana, IL 61801

Abstract – **It is challenging to reduce the processor power consumption while meeting the processor resource requirement of distributed multimedia applications in portable computers. This paper presents a middleware framework coordinating the Processor/Power Resource Management (*PPRM*) in mobile computing environment. Our framework has four major contributions: (1) providing a power-aware resource reservation mechanism, where admission control is based on the processor utilization and power availability; (2) adjusting the speed and corresponding power consumption of the processor upon events, triggered by the change of the system workload or power availability; (3) updating reservation contracts of multimedia applications to maintain their resource requirements while adjusting the processor speed; and (4) notifying applications about the change of resource status to enable them to adapt their behavior and complete tasks before the power drains. Our experimental results show the effectiveness of the coordinating PPRM framework to save energy and maintain the resource requirements of multimedia applications.**

I. INTRODUCTION

Distributed multimedia applications are becoming increasingly an integral part of the ubiquitous computing environment, using battery-operated portable computers. They need to consume system-level resources, such as processor time and power, to satisfy a user-level quality of service (QoS) requirement. On the other hand, reducing power consumption, thereby extending the battery lifetime, of portable computers is a major concern in mobile computing. Therefore, it is challenging to reduce power consumption while meeting the resource requirement of multimedia applications.

Resource reservation is a common mechanism to separate soft real-time multimedia applications from best-effort applications and provide resource availability to multimedia applications with reservation contracts. However, most proposed approaches [9][10][11][12] focus on the processor resource only and consider the processor speed as a constant. On the other hand, previous power management work [6][7][8] saves the processor energy consumption by adjusting the processor speed and voltage in intervals. The processor speed is determined by predicting future workload, based on the workload over the proceeding interval. These approaches have three common limitations. First, they globally adjust the processor speed without differentiating applications with different performance and resource requirements. Second, they provide no mechanism to notify the resource status change to adaptive applications. Finally, the interval-based prediction and adjustment may incur unnecessary overhead when the workload does not change.

In this paper, we present a middleware framework coordinating the Processor/Power Resource Management (*PPRM*) for portable computers. The goals of the framework are: (1) providing soft real-time scheduling to multimedia applications under different processor speeds and power levels, (2) achieving minimum energy wasted, and (3) differentiating applications with different performance requirements under low power availability. The coordinating PPRM framework makes four major contributions to meet the above goals. First, it allows multimedia applications to make a power-aware processor resource reservation, and makes admission control based on both processor utilization and power availability. Second, it dynamically adjusts the speed and power consumption of the processor, when the system workload changes or the power availability is not sufficient. Third, it distinguishes soft real-time multimedia applications from best-effort applications when adjusting the processor speed, so that multimedia applications maintain their performance and best-effort applications tolerate more performance degrading. Finally, it notifies applications when the processor is overloaded and the battery cannot last long enough for all applications. Upon such a notification, applications can adapt their behavior to complete their tasks at a lower level resource requirement before the battery drains.

The rest of the paper is organized as follows. First, section II introduces the system architecture. Section III describes the PPRM coordinating algorithm. Section IV presents experimental evaluation. Finally, section V provides some concluding remarks.

* This work is supported by the Air Force grant F30602-97-2-0121, the NSF CCR 96-23867, the NSF CISE CDA 96-24396 and the NSF CISE EIA 99-72884. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, or the U.S. government.

II. SYSTEM ARCHITECTURE

Recently, Compaq, Intel, Microsoft, Phoenix and Toshiba proposed the Advanced Configuration and Power Interface (ACPI) to provide an open standard for power management [5]. The ACPI specification defines an interface between operating system and hardware for power management, as shown in Fig.1. The OS-based Power Management (OSPM) system code in the operating system implements the power management policies.

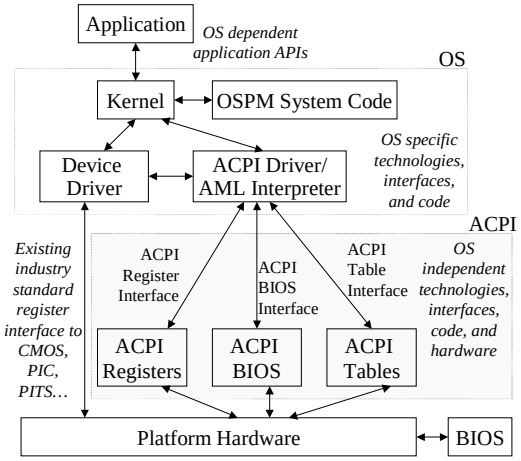

Fig. 1. ACPI and power management

We have designed an ACPI-based middleware framework, which coordinates the Processor/Power Resource Management (*PPRM*) to achieve: (1) soft real-time scheduling guarantees under different power levels, (2) minimum energy wasted, and (3) differentiation among applications with different performance requirements. The system architecture consists of four layers: resource layer, OS layer, middleware layer, and application layer, as shown in Fig. 2.

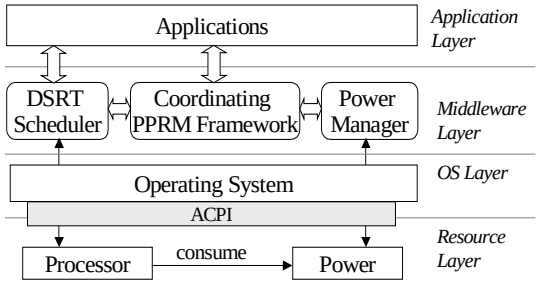

Fig. 2. PPRM architecture

The operating system exports hardware resource (such as processor and power) status to the middleware layer, which receives resource request from applications. The middleware layer consists of three major components: the *Dynamic Soft Real-Time* (*DSRT*) processor scheduler [9], the power manager, and the coordinating PPRM framework. The DSRT scheduler allows multimedia applications to reserve processor resource and corresponding power resource, and monitors the system workload. The power manager monitors the power availability (i.e., how long the battery life remaining) and the processor power consumption. The coordinating PPRM framework (1) determines polices how to adjust reservations according to power availability, (2) uses the corresponding polices to differentiate applications in case of low power availability, (3) adjusts the processor speed to achieve a minimum energy wasted, and (4) notifies applications, if it cannot extend the battery life and meet the processor resource requirements of applications under low power availability.

III. PPRM COORDINATING ALGORITHM

To coordinate the processor and power resource management, the coordinating PPRM framework must have knowledge about processor's operating modes, the future power availability, and the future workload. This section first introduces the resource model, which describes the relationship between processor operating modes and the future power availability, and then presents the workload model, followed by the coordinating algorithm.

A. Resource Model

Power consumption P of a processor is essentially proportional to the switching capacitance C, the square of the voltage V, and the clock frequency f, i.e., $P \propto CV^2f$. Some modern processors, such as Mobile Intel Pentium III[1], Crusoe TM5600[2], PowerPC 8260[3], and AMD-K6-III+[4], support multiple operating modes with tradeoff between performance and power consumption. For example, a low voltage Mobile Intel Pentium III processor has two operating modes: *battery optimized* with frequency 500MHz, voltage 1.1V and power 1.5watt(W), and *performance optimized* with frequency 600MHz, voltage 1.35V and power 2.0W[1]. In follows, we assume a futuristic (simulated) processor as an example, whose operating modes are shown in Table I.

TABLE I
OPERATING MODES OF AN IMAGINED PROCESSOR

Mode	*Speed*	*Power Consumption*
Mode 1	600MHz	2.0W
Mode 2	500MHz	1.5W
Mode 3	400MHz	1.2W
Mode 4	300MHz	0.9W

The power availability is determined by two factors: the battery energy and the power consumption, which is, in turn, determined by the processor speed. In this paper, we use the current actual power availability (PA_{Actual}), measured with function *GetSystemPowerStatus*() as explained in section IV, to predict the future power availability ($PA_{Predicted}$) when adjusting the processor speed. The prediction is based on the equation PA_{Actual}* (*current*

power consumption) = $PA_{Predicted}$(adjusted power consumption)*. Fig.3 shows the relationship between the processor operating modes and the predicted power availability for the imagined processor in Table I. The actual power availability is 90 minutes when the processor currently runs at speed 600MHz(power 2.0W), and the predicted power availability will be 120 minutes if the processor is switched to run at speed 500MHz (power 1.5W). Note that the predicted power availability is used for the processor speed adjustment, as explained in subsection C; and the power manager monitors the actual power availability and reports the low power availability.

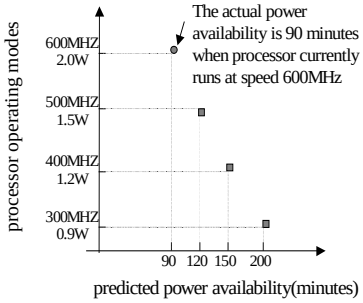

Fig. 3. Relationship between processor operating modes and predicted power availability

B. Workload Model

The coordinating PPRM framework relies on the DSRT scheduler, which provides a power-aware processor resource reservation mechanism to separate soft real-time multimedia applications from best-effort applications, and statistically multiplexes the processor resource between real-time and best-effort applications. The system workload consists of two parts: real-time workload and best-effort workload, as shown in Fig.4. Each real-time application reserves a certain amount of processor resource, *required capacity*, C_{RE} (as explained below), and the real-time workload is the sum of the *required capacity* of all admitted real-time applications in the system. The best-effort workload is limited by the available unreserved processor resource.

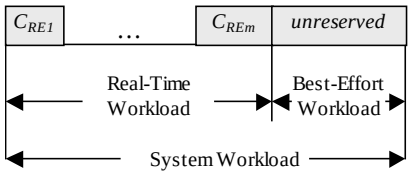

Fig. 4. Workload model

A multimedia application makes processor resource reservation through the power-aware reservation specification (*class, period, utilization, speed, duration, weight*), extended from the reservation specification in [14], as shown in Table II.

TABLE II
POWER-AWARE RESERVATION SPECIFICATION

Parameter	*Meaning*
class	Periodic constant processing time (*PCPT*) or periodic variable processing time (*PVPT*)[14].
period	Inform the scheduler when to release a new job, which must be finished before a deadline.
utilization	How much percentage of processor resource to reserve.
speed	Context for *utilization* (request *utilization* percentage when the processor runs at *speed*)
duration	How long the application lasts[1].
weight	Importance of the application under low power availability.

Required capacity, C_{RE}, of an application is defined as C_{RE}*=utilization*speed/highest processor speed*. The DSRT scheduler allocates the *required capacity* processor resource to the application when power is enough, and reduces the processor resource according to the parameter *weight*$\in$[0,1] under low power availability, i.e. C_{RE}*(low power availability)=weight**C_{RE}. For example, a video decoder requests *utilization 25%, speed 600*MHz, and *weight 0.8* on a processor with the highest speed *600*MHz (C_{RE} *=25%*600MHz/600MHz = 0.25)*. The DSRT scheduler allocates *utilization 30%* to the decoder, when the processor runs at speed *500*MHz (*30%*500MHz /600MHz = 0.25*). It allocates *utilization 40%* to the decoder, when the processor runs at speed *300*MHz *(0.25*0.8= 40%*300MHz/600MHz*) under low power availability. Fig. 5 shows that the decoder needs different utilization to decode a video frame before a deadline.

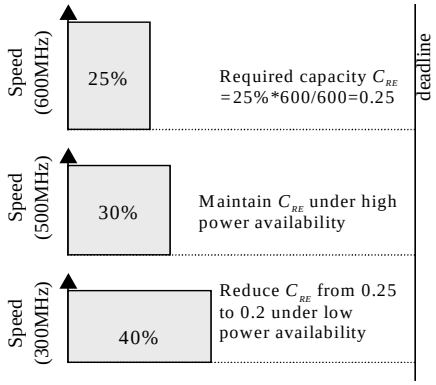

Fig. 5. Different utilization for frame decoding

C. Coordinating Algorithm

The coordinating algorithm dynamically adjusts the speed and corresponding power consumption of the processor to meet the following goals[2]:

[1] Parameter *duration* is used to determine if the power is enough for the application to finish its task, as explained in next subsection.

[2] Here, we focus on the coordinating resource management when the system is active. A simple *timeout* scheme is employed to turn off the process when there is no activity for a specified time.

(1) Ensuring enough power availability for all admitted multimedia applications (that is, the battery can last for the maximum duration of all multimedia applications).
(2) Allocating the *required capacity* of processor resource to each multimedia application under high power availability.
(3) Reducing the *required capacity* C_{RE} to $weight*C_{RE}$ for each multimedia application, and notifying applications under low power availability.
(4) The processor runs as slowly as possible to save energy while meeting the above goals.

Fig. 6 shows the coordinating algorithm. Meeting the processor/power resource requirement of a soft real-time application means that (a) the *required capacity* of processor resource can be allocated to the application; and (2) the power availability is enough for the duration of the application. Note that a certain amount of processor resource is shared among all best-effort applications to protect starvation for best-effort applications [14].

1. *Event of resource reservation request*
   - 1.1 If current processor speed can meet the requested processor/power resource requirement, then admit the reservation.
   - 1.2 Otherwise, speed up the processor until meeting processor/power resource requirements of already admitted and the requesting applications.
   - 1.3 If unable to achieve 1.2, reject the request and recover the processor speed.
   - 1.4 Otherwise admit the request and update reservation contracts
     $Util_{NEW} = Util_{OLD}*Speed_{OLD} / Speed_{NEW}$
2. *Event of resource reservation release*
   - 2.1 Slow down the processor while meeting processor/power resource requirements of admitted applications.
   - 2.2 Update reservation contracts
     $Util_{NEW} = Util_{OLD}*Speed_{OLD} / Speed_{NEW}$
3. *Event of low power availability*
   - 3.1 Reduce the *required capacity* C_{RE} to $weight*C_{RE}$ for each admitted application.
   - 3.2 Slow down the processor until meeting processor/power resource requirement for all admitted applications.
   - 3.3 If unable to achieve 3.2, notify the user and applications to enable them adapt, thereby reducing resource requirements.
   - 3.4 Update the reservation contracts
     $Util_{NEW} = weight*Util_{OLD}*Speed_{OLD} / Speed_{NEW}$

Fig. 6. Coordinating algorithm

IV. Implementation and Experiments

We have implemented the coordinating PPRM framework prototype on Windows NT platform. The DSRT scheduler, ported from our previous work on the Unix platforms [9], monitors the real-time workload through the admission control, and the actual processor utilization with Performance Data Helper (PDH) functions: *PdhOpenQuery*(), *PdhAddCounter*() and *PdhCollectQueryData*(). The power manager monitors the actual power availability with Microsoft *OnNow* [13] power management function *GetSystemPowerStatus*().

The experiments described here are performed on a battery-operated IBM T20 laptop with a single Pentium III 700Mhz processor and 128MB RAM. The operating system is Microsoft Windows 2000 Professional. The experimental applications include *math*, a mathematic computing program, and *mpegplay*, the Berkeley MPEG 2 player. In our experiments, *math* is a PCPT soft real-time application; *mpegplay* is a PVPT soft real-time application. In the experiments, we assume the processor has the operating modes in Table I, and simulate the processor speed adjustment[3] through a program, which consumes a certain percentage of processor time. For example, when we want to slow down the processor to 500MHz, the program consumes 1/6 processor resource. Therefore, there is only 5/6 processor resource available to all other applications, that is, the processor seems to run at speed 500MHz.

Our experiments have the following steps:
(1) For the first 16 seconds, there is no real-time workload, so the processor runs at the slowest speed *300*MHz, and best-effort applications can use all processor resource.
(2) At time 17, the *mpegplay* program requests to reserve (*utilization* 60%, *speed* 300MHz, *weight* 0.9, *duration* 600 seconds)[4] and the *required capacity* is 60%*300MHZ/600MHZ=0.3. This reservation request can be admitted without adjusting the processor speed.
(3) At time 41, the *math* program requests to reserve(*utilization* 80%, *speed* 300MHz, *weight* 0.6, *duration* 100 seconds) and the *required capacity* is 80%*300MHZ/600MHZ=0.4. The coordinating policy adjusts the processor speed to *500MHz* to admit this reservation request with new utilization 48%. It also updates the reservation contract of the *mpegplay* with new utilization 36% to maintain its required capacity.

[3] We are currently seeking ACPI-compliant programming interfaces to control the processor speed by a user-level program.
[4] The *period* parameter in specification is used for soft real-time scheduling [9][14] and not focused in this paper.

(4) At time 89, the power manager finds that the actual power availability is 400 seconds and not enough for the *mpegplay* program. Therefore, the coordinating policy adjusts the *required capacity* of *mpegplay* to 0.3*0.9=0.27, and the required capacity of *math* to 0.4*0.6=0.24. Then it slows down the processor to 400MHz, and allocates new utilization 40.5% and 36% to *mpegplay* and *math*, respectively. The predicted power availability is 400 seconds*1.5W/0.9W=666.6 seconds, and enough for both programs.

Fig. 7 shows the required capacity, actual utilization of the *mpegplay* program, and the processor speed over time. The processor runs with speed *300MHz* between time *1* and *40*, *500MHz* between *41* and *88*, and *400MHz* between *88* and *152*. Therefore, the saved energy percentage is *((2.0-0.9)*(40-1+1) + (2.0-1.5)*(88-41+1) + (2.0-1.2)*(152-88+1))/(2.0*152) =39.5%*.

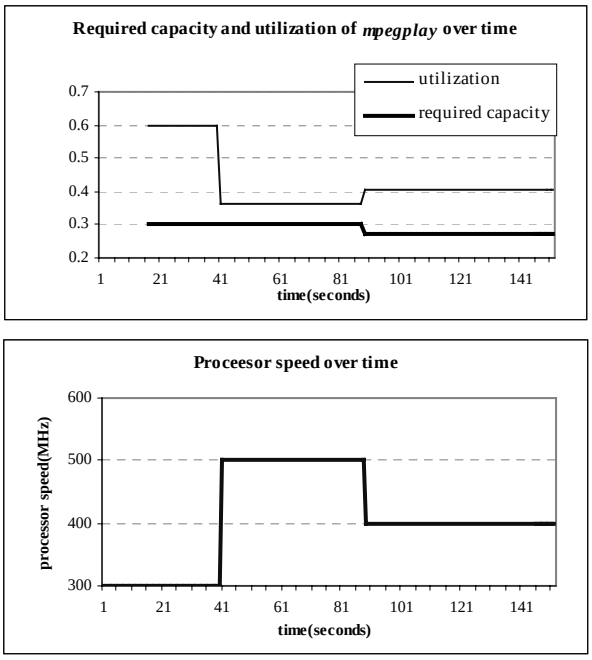

Fig. 7. Required capacity, utilization and processor speed over time

V. CONCLUSION

Multimedia applications are becoming ubiquitous in battery-operated portable computers, which have constraints in power consumption. In this paper, we have presented a middleware framework coordinating the processor and power resource management (*PPRM*). PPRM dynamically adjusts the processor speed and power consumption based on the observation of the system workload, the processor status, and the power availability. It maintains the resource requirement of multimedia applications through a power-ware processor resource reservation mechanism. When power is not enough, PPRM reduces the resource requirements of applications and also notifies user and applications, so that applications can adapt behavior to finish tasks with a lower user-level QoS before the battery drains. Our experimental results demonstrate the capability and effectiveness of PPRM to save energy while meet the processor resource requirements of multimedia applications.

REFERENCE

[1] Intel, *Mobile Intel Pentium III Processor featuring Intel SpeedStep Technology*, [online] http://www.intel.com/design /mobile/perfbref/24956001.pdf, March 2001.

[2] Transmeta, *Crusoe Processor Model TM5600 Features*, [online] http://www.transmeta.com/crusoe/download/pdf/ TM5600_ProductBrief_8-2-00.pdf, August 2000.

[3] Motorola, *MPC8260 HiP3 Hardware Specifications*, [online] http://e-www.motorola.com/brdata/PDFDB/ MICROPROCESSORS, February 2001.

[4] AMD, *Mobile AMD-K6-III+ Processor Data Sheet*, [online] http://www.amd.com/K6/k6docs/pdf/23535.pdf, May 2000.

[5] Compaq, Intel, Microsoft, Phoenix and Toshiba, *Advanced Configuration and Power Interface Specification*, [online] http://www.teleport.com/~acpi/ spec.htm, July 2000.

[6] M. Weiser, B. Welch, A. Demers, S. Shenker, "*Scheduling for Reduced CPU Energy*," USENIX Symposium on Operating Systems Design and Implementation (OSDI), 13-23, Nov. 1994.

[7] K. Govil, E. Chan, and H. Wasserman, "*Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU*," Proc. ACM Int'l Conf. on Mobile Computing and Networking, 13-25, Nov. 1995.

[8] Pering, T. Burd, and R. Brodersen, "*The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms,*" Proc. Int'l Symposium on Low Power Electronics and Design, 76-81, Aug. 1998.

[9] Hao-hua Chu, Klara Nahrstedt, "*CPU Service Classes for Multimedia Applications*," Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'99), Florence, Italy, June 1999.

[10] C.W. Mercer, S. Savage and H. Tokuda, "*Processor Capacity Reserves: Operating System Support for Multimedia applications*," Proc. of the Int. Conf. on Multimedia Computing and Systems, 90-99, May 1994.

[11] M.B. Jones, D. Rosu, M.-C. Rosu, "*CPU Reservations & Time Constraints: Efficient, Predictable Scheduling of Independent Activities*," Proc. of the 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, 198-211, October 1997.

[12] Chen Lee, Raj Rajkumar and Cliff Mercer, "*Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach,*" Proc. of Multimedia Japan, Match 1996.

[13] Microsoft, *OnNow Power Management Architecture for Applications*, [online] http://www.microsoft.com/hwdev/desinit/ onnowapp.HTM, March 2000.

[14] Wanghong Yuan, Klara Nahrstedt, Kihun Kim, "*R-EDF: A Reservation-Based EDF Scheduling Algorithm for Multiple Multimedia Task Classes*," Proc. of the 7th IEEE Real-Time Technology and Applications Symposium, Taiwan, May 2001.