Constrained Power Management: Application to a Multimedia Mobile Platform

Patrick Bellasi*, Stefano Bosisio*, Matteo Carnevali* William Fornaciari*, and David Siorpaes†
* Dipartimento di Elettronica e Informazione
Politecnico di Milano, P.zza Leonardo da Vinci, 32. 20133 - Milano, Italy
Email: {bellasi,fornacia}@elet.polimi.it {stefano.bosisio,matteo.carnevali}@mail.polimi.it
†Advanced System Technology
STMicroelectronics, Via C. Olivetti, 2. 20041 - Agrate Brianza, Italy
Email: david.siorpaes@st.com

***Abstract*—In this paper we provide an overview of CPM, a cross-layer framework for *Constrained Power Management*, and we present its application on a real use case. This framework involves different layers of a typical embedded system, ranging from device drivers to applications. The main goals of CPM are (i) to aggregate applications' QoS requirements and (ii) to exploit them to support an efficient coordination between different drivers' local optimization policies. This role is supported by a system-wide and multi-objective optimization policy which could be also changed at run-time.**
In this paper we mostly focus on a real use case to show the very low overhead of CPM both on the management of QoS requirements and on the tracking of hardware crossdependencies, which cannot be directly considered by local optimization policies.

I. Introduction

Resources management, and thus also energy saving, has become a high priority design goal for embedded multimedia mobile devices such as smartphones. Such devices are usually based on platforms using a System-on-Chip (SoC), which embeds a number of peripherals sharing some resources and competing on their usage. These many-processor embedded systems are also usually characterized by a set of working modes, each one corresponding to different profiles of power and performances.

At the same time, modern consumer electronics products provide multiple functionalities, ranging from classic phone calls to more complex use-cases which involve network access and audio-video playback. These multiple application scenarios are usually characterized by the competition on the limited resources available and could also involve conflicting requirements on the underlaying hardware.

Therefore, nowadays the resource and power management of such new generation platforms has become a more and more complex added value. This requires the search of the optimal trade-off between power saving and performances perceived by the user. Thus, it is worth to treat it as an optimization problem targeted to the definition of solutions that are easily portable among different products.

In the light of these considerations, to effectively support the optimization of an entire platform, it is necessary (i) to collect and aggregate applications' QoS requirements and (ii) to exploit them within a dynamic system-wide optimization policy. This global optimization policy, in our view, should be targeted to provide a coarse-grained tuning for already existing low-level and device-specific optimization policies.

The rest of this paper is organized as follow. In the next section we briefly present an overview of the state of the art. Section II presents our system-wide optimization framework. A representative application is illustrated in Section III and the corresponding experimental results are presented in Section IV. Conclusions are drawn in Section V.

A. Related research works

Techniques to reduce power consumption in computing systems range from physical layers design up to higher software abstraction levels [1]. Cost-effective solutions require to tackle the problem at all the abstraction levels simultaneously, thus the development of holistic approaches, that aggregate data from multiple layers into PM decisions, is a popular research topic. Indeed, a number of approaches based on cross-layer adaptations have already been proposed [2], [3], [4]. Unfortunately none of these approaches has been really implemented within a commercially available product.

The Linux Operating System (OS), a common choice in many new generation mobile devices, is already featured by different frameworks providing good support to exploit hardware power saving capabilities of single devices (e.g., CPU) or subsystems (e.g., clock tree). However, this OS still lacks of a well settled overall coordination and global system-wide optimization strategy. This lack has been the subject of several extension proposals. However, either these approaches have not been effectively integrated within Linux [5], [6], or are based on complex models to be build off-line and thus not easily portable across different platforms [7].

II. Constrained Power Management Overview

CPM is an approach to manage both system resources and power of an entire platform. It allows to collect QoS requirements from the applications and to coordinate device drivers in a distributed manner to support the requested QoS level w.r.t a system-wide dynamic optimization policy that can be directed either to optimize energy or performance.

978-3-9810801-6-2/DATE10 © 2010 EDAA

A. Basic concepts

The CPM model is based on few theoretical concepts:
- The *System-Wide Metrics* (SWMs) are parameters that describe behaviors of a running system and represent QoS requirements. They could be either "abstract" (ASMs) or platform dependent (PSMs). The formers are exposed to user-space and can be used by the applications to assert QoS requirements. The latters, instead, are defined in the platform code and are used to keep track of hardware inter-dependencies.
- The *Device Working Regions* (DWRs) define the mapping between the operating modes of a devices and the SWMs that define the QoS level supported by each mode.
- The *Feasible System Configurations* (FSCs) are the *n*-dimensional intersection of at least a DWR for each device (where *n* is the number of SWMs defined). They identify the system-wide working points of the target platform where certain QoS levels are granted.
- The *Constraints* on SWMs are defined at run-time according to the QoS requirements of applications or drivers on these parameters. All the QoS requirements on the same SWM are translated on a constraint using an aggregation function which depends on the type of the parameter; this function could be addition/subtraction or min/max.
- The *Optimization Policy* is based on Linear Programming concepts. It supports a multi-objective optimization which could consider different performance parameters, by assigning a weight to each SWM, and energy consumptions, by assigning a power consumption measure to each FSC.

B. How CPM works

We could distinguish three main phases, with decreasing computational complexity and increasing execution frequency:
- *FSC Identification*: at boot time all the drivers register to CPM by exposing their DWRs. All FSCs can be automatically identified by performing the intersection of DWRs.
- *FSC ordering*: every time the optimization goals change, the FSC are sorted according to the global optimization policy. This happens usually when the device usage scenarios change.
- *FSC selection*: at run-time applications can assert QoS requirements on a specific SWM. These requirements are aggregated to produce a new constraint for each SWM. These constraints could invalidate some FSC, if also the current FSC is invalidated, then a new candidate is selected according to the ordering defined in the ordering phase.

Finally, all drivers are notified about the new FSC and required to update their operating mode accordingly.

The CPM model has been implemented as a Linux kernel framework (version 2.6.30) and tested under some use-cases to evaluate its overheads. In the next section we describe one of these scenarios to show the benefits of this approach.

III. A Meaningful Use-Case

The use case presented in this paper aims at showing the benefits of using CPM to manage resources according to the actual demand of applications while correctly keeping track of dependencies among different subsystems.

The considered platform is the STn8815 SoC by STMicroelectronics, which has an ARM host CPU and multimedia operations powered by an audio DSP (DSP_A) and a video DSP (DSP_V). The host CPU is clocked by CPU_CLK, while both the DSPs are clocked by DSP_CLK. The internal architecture of this SoC enforces a strict dependency between CPU_CLK and DSP_CLK, as shown in Fig. 1a. When one or both the DSPs are active and clocked at a certain frequency, also the CPU is constrained to work at a compatible frequency. Thus the processor optimization policy must be opportunely notified at run-time about this constraint.

A. Description of the use case

A quite common scenario for modern smartphones is the download of some data from the web while playing some audio-video content streamed by Internet. This scenario involves two applications: one that controls the download and playback of stream contents, and another that downloads other data, for instance to sync a mail account. Both these applications share a common resource, the connection bandwidth, and could require a minimum QoS level on it. For instance different video qualities require different connection speeds.

The devices involved in the use case are a 3G modem, both the DSPs and the CPU. The modem supports several connection modes to the mobile data network. Each mode is capable of a maximum bandwidth capacity and usually corresponds to different levels of energy consumption. The audio and video DSPs support different coding standards, which correspond to different processing work loads, and thus could require proper operating frequencies. The CPU is controlled by an optimization policy which independently select the operating frequency according to its past workload.

B. Integration in CPM

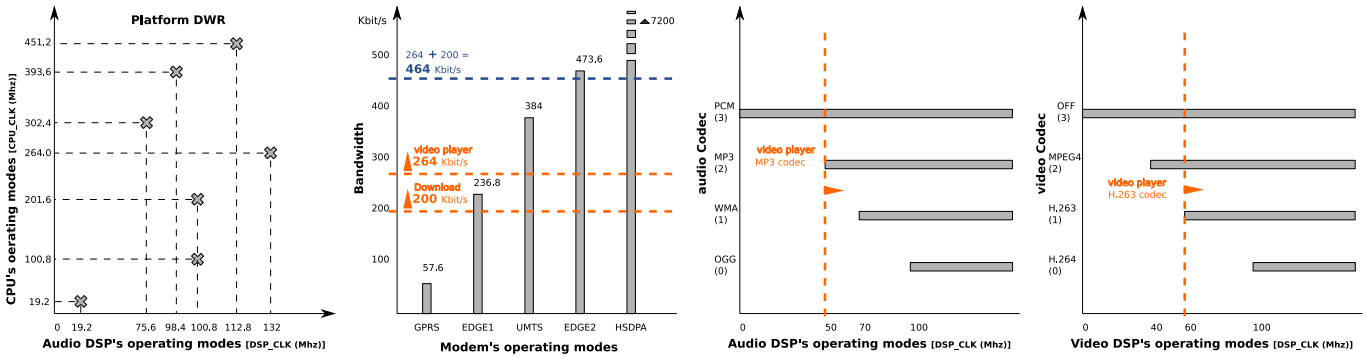
The ASMs considered are:

- *connection bandwidth*: a resource on which applications compete. An additive function is used to aggregate the requirements, asserted by applications, and identify a constraint on the QoS level for this resource.
- *audio and video codecs*: an information that is related to multimedia content that the application has to play, thus it directly impacts on the operating mode of the DSPs.

The PSMs (*DSP_CLK and CPU_CLK*) are platform specific informations defined in the platform code to keep track of hardware inter-dependencies described so far. Finally, the driver of each involved device exploits the ASMs to define its own DWRs as depicted in Fig. 1b, 1c and 1d.

C. Dynamics of the use case

The use case begins with the user selecting a video stream content to be played. As soon as the download of audio and video data starts, the player application collects informations



(a) (b) (c) (d)

Fig. 1. a) Modem's DWRs with constraints asserted by applications and the aggregated constraint. b/c) Audio/Video DSP's DWRs related to the audio/video codec ASM and DSP_CLK. d) Platform DWR to track CPU_CLK and DSP_CLK dependencies

about the connection bandwidth required to have good playback (264 Kbit/s)[1], the audio codec (mp3) and video codec (h.263) of the encoded content and set a QoS requirement on the corresponding ASMs. These requirements are expressed as a lower bound value on the ASM *bandwidth* and as a single value on the ASMs *audio codec* and *video codec*; they are represented by the orange constraints in Fig. 1b, 1c and 1d, respectively.

The assertion of these constraints invalidates the current FSC. Thus, CPM coordinates the selection of a new candidate FSC and the corresponding DWRs are notified to the modem and DSPs for a distributed agreement. The required codecs are bound to a specific DSP_CLK frequency: 50MHz for DSP_A and 60MHz for DSP_V. The platform DWRs (Fig. 1a) allow to manage the dependency with the CPU_CLK which is setup to support the desired frequency. Therefore the CPU frequency optimization policy will be able to scale the processor frequency according to the imposed constraint ($\geq$100.8MHz). After the agreement, the candidate FSC is activated and all involved subsystems move to the new working mode, e.g., the modem switches from GPRS to EDGE1.

The use-case continues and during the playback, an application that downloads data from the web starts, e.g., an email update application. This new download application asserts a requirement on the bandwidth for an amount of 200 kb/s. Since the ASM *bandwidth* is of type *additive*, the aggregation function takes into account all the previously asserted requirements and aggregates with a sum. Thus, 464 kbit/s becomes the new active constraint on the bandwidth and thus a new FSC is selected, which brings the modem device to move to the EDGE2 working mode in order to satisfy the requirements. The aggregation of the requirements is represented by the blue constraint in Fig. 1b.

Finally, the video stream playback ends and the corresponding requirements on bandwidth and codecs are de-asserted. This leads to a new aggregation on the bandwidth, which results in a subtraction of the value on the ASM *bandwidth*.

[1]i.e., no jitters and buffer underruns

Declaration of the BANDWIDTH *ASM*

```
struct cpm_asm cpm_platform_asm[] = {
 [0] =
    CPM_PLATFORM_ASM("BANDWIDTH", CPM_TYPE_GIB,
      CPM_USER_RW, CPM_COMPOSITION_ADDITIVE, 0, 8000),
}
```

ASM id
ASM name
ASM type
writable?
aggregation function
min max value

Declaration of the Modem *Device*

```
static struct platform_device modem = {
 .name = "modem",
 .id = 1,
 .dev = {
 .release = cpm_modem_dev_release,
  }, };
```

Declaration of the Modem's *DWRs*

```
struct cpm_asm_range modem_dwr0_ranges[] = {
 DEV_DWR_ASM(0, 0, 57, CPM_ASM_TYPE_RANGE),
};
[...]
struct cpm_asm_range modem_dwr4_ranges[] = {
 DEV_DWR_ASM(0, 0, 7200, CPM_ASM_TYPE_RANGE),
}; };
```

ASM id
ASM range: min, max values
ASM type of values

Declaration of the Modem's *DWR list*

```
struct cpm_dev_dwr modem_dwrs_list[] = {
 DEV_DWR(&modem.dev, 0, "GPRS", modem_dwr0_ranges,
  ARRAY_SIZE(modem_dwr0_ranges)),
[...]
 DEV_DWR(&modem.dev, 4, "HSDPA", modem_dwr4_ranges,
  ARRAY_SIZE(modem_dwr4_ranges)), };
```

device reference
DWR id
DWR name
DWR values

Fig. 2. Example declaration of an ASM and of the modem's DWRs

At this point only the download application is still active and a new FSC is selected and activated.

IV. Experimental Results

Each driver has been updated to provide the necessary informations to interface with CPM, i.e., the definition of its DWRs. The modifications of the driver code to support CPM are easy and have low impact in the development and maintenance of device drivers. For example, in Fig. 2 is shown how ASMs and modem's DWRs have been coded in the driver.

The modem device maps its working modes (i.e., GPRS, EDGE1, UMTS, EDGE2, and HSDPA) on certain values of bandwidth. The mapping is expressed through definition of DWRs, setting a range of feasible values on the ASM BANDWIDTH. In the same way also the other drivers involved in the use case have been patched.

The described use-case has been executed on the modified platform and the representative results are reported in the

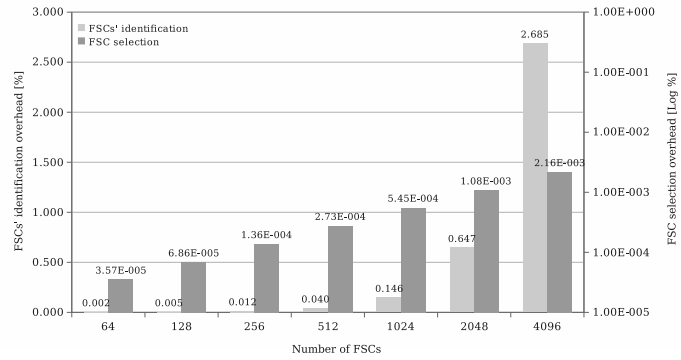


Fig. 3. CPM: overhead of FSC identification and selection

following plot.

A. Resources Management

The requirements declaration and aggregation allows to keep a correct and precise view of used and still available system resources. This could be exploited to configure the hardware devices on the best *feasible* operating mode that supports the resources demand. A positive effect of this method is the energy saving that could be achieved by selecting, for each QoS demand, the optimal working mode not only with respect to a multi-objective performances optimization policy but also considering the system-wide power consumption, which can be associated to every FSC.

B. Dependency Tracking

CPM allows to track hardware dependencies among different subsystems of a SoC that may prevent a correct operation of a system. Instead of patching each device driver to adapt to the platform, developers declare platform DWRs to solve dependencies issues. In that way code portability is improved.

C. Identification of Feasible System-wide Configurations

The automatic computation of the FSCs allows to identify all the feasible working points of an entire platform. This is done by exploiting the information defined, independently, in each device driver code. Other approaches to PM require to code all the working points by hand. Considering that in the presented use case the total number of FSCs was 415, we understand how interesting is the ability to automatically compute these point. Thus, this is a relevant result by itself. Moreover, it improves the portability of the solution across different platforms because allows to reuse drivers defined informations.

D. Overheads and Time Domains

We have measured the execution time of the algorithms for the identification and the selection of FSCs obtaining the results in Fig. 3. These overhead measurements refer to a 60s execution of the use-case and focus on the worst-case.

This measures prove the negligible impact of the framework with respect to a system not using it. Indeed, the identification algorithm shows a maximum of 2.5% overhead for a quite complex system with 4096 feasible configurations, which is much more than the 415 of the considered use-case. This means that over the 60s of use-case execution, around 1.5s are devoted to the framework execution. However it should be considered that this algorithm runs just one time at system boot and can be easily replaced by a look-up table. Indeed, especially in embedded systems, where the platform's configuration for a final product does not change, all the FSC can be pre-computed and then just loaded at boot time.

While the identification algorithm has a complexity which is exponential in the number of the FSCs, the selection algorithm not only has a better (linear) complexity but is also three orders of magnitude better in absolute values. This is also another important result, because the identification algorithm is the one executed more frequently, i.e. each time a new requirement is asserted by an application. The experimental setup considered one run every 10 seconds and the measurements show a really negligible overhead which is always less than 0.01%.

V. Conclusions

In this paper we have presented CPM, a Linux kernel framework, applied to a real use case. The proposed method allows to coordinate the operation of different subsystems of a SoC-based platform exploiting hardware power saving capabilities. CPM collects and aggregates QoS requirements from the applications and coordinates the reconfiguration of devices' working modes in order to support the expected QoS. In particular, CPM's mechanisms permit to track hardware dependencies and to obtain a correct and precise view of used and available system resources, while supporting global system-wide optimization of QoS and energy thanks to the definition of dynamic policies.

References

[1] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 195–237, 2005.

[2] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Integrated power management for video streaming to mobile handheld devices," in *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*. New York, NY, USA: ACM, 2003, pp. 582–591.

[3] Y. Fei, L. Zhong, and N. K. Jha, "An energy-aware framework for dynamic software management in mobile computing systems," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 1–31, 2008.

[4] N. AbouGhazaleh, B. Childers, D. Mosse, R. Melhem, and M. Craven, "Energy management for real-time embedded applications with compiler support," in *LCTES '03: Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*. New York, NY, USA: ACM, 2003, pp. 284–293.

[5] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "Ecosystem: managing energy as a first class operating system resource," *SIGPLAN Not.*, vol. 37, no. 10, pp. 123–132, 2002.

[6] M. Anand, E. B. Nightingale, and J. Flinn, "Ghosts in the machine: interfaces for better power management," in *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*. New York, NY, USA: ACM, 2004, pp. 23–35.

[7] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser, "Koala: a platform for os-level power management," in *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*. New York, NY, USA: ACM, 2009, pp. 289–302.