CPM: A Cross-Layer Framework to Efficiently Support Distributed Resources Management

Patrick Bellasi, Stefano Bosisio, Matteo Carnevali, William Fornaciari
Politecnico di Milano, P.zza Leonardo da Vinci, 32. 20133 - Milano, Italy

David Siorpaes
STMicroelectronics, Via C. Olivetti, 2. 20041 - Agrate Brianza, Italy

Abstract

Resource management, and especially power management, is a key aspect for the success of modern battery supplied multimedia devices. This class of devices are usually based on SoCs with a wide range of heterogeneous subsystems, competing for shared resources while offering several power control mechanisms. Many of these mechanisms require suitable software support to be exploited effectively. Unfortunately, real commercial systems focusing on mobile multimedia exposes a software layer composed by a collection of fairly independent local optimization policies, each focusing on a specific device or subsystem. This paper presents CPM, a cross-layer model and framework to support system wide resource management. The main goals of this framework are to efficiently support the aggregation of applications' QoS requirements and to provide a dynamic, system-wide, and multi-objective optimization to coordinate devices' local policies. A real solution, which is well integrated within the existing Linux kernels, has been developed and evaluated to asses its negligible overhead.

1 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 devices 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 usage 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 which 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 QoS requirements and (ii) to exploit them within a dynamic system-wide optimization policy. This global optimization policy 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. The rest of this section briefly presents an overview of the state of the art. Section 2 presents our system-wide optimization framework. Experimental results are discussed in Section 3. Finally, Section 4 draws some conclusions.

1.1 Related 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].

2 Constrained Power Management

Supporting system-wide power and resource management is crucial to allow multiple functionalities with different competing requirements to peacefully coexist on the same device. Cost-effective solutions for this kind of optimizations require to tackle the problem at different abstraction levels simultaneously. From one side, optimization policies focusing on specific devices and subsystems are already available. Thus, they should be considered in order to exploit their detailed and device specific knowledge on available resource and working modes. On the other side, user-space applications define the system requirements that should be meet in order to get certain QoS levels. Thus, to effectively solve the optimization problem we cannot disregard any of these levels. Indeed, while multiple applications could assert requirements on the same system resources, these requirements must be properly aggregated and translated into constraints for the multiple optimization policies available. An effective solution should also grant that inter-dependencies among different devices are properly tracked. Otherwise, a local optimization policy could take optimization decisions which are suboptimal or even worst with dangerous side-effects on either performances or system-wide power consumption.

The solution proposed in this work is based on the concept of *constrained power management* and provide a framework, implemented at OS level and named CPM, which supports the identification of an optimal trade-off between expected performances and reduce power consumptions. In the light of the previous consideration, the fundamental approach of this framework has been designed according to the main requirements to be: a) *system-wide*, to grant overall benefits considering all the different subsystems; b) *fine-detailed*, to effectively exploit a detailed description of each platform's capabilities; c) *dynamic*, to smoothly adapt to frequently changing working scenarios (use-cases); d) *scalable*, to be easily portable to different systems with also increasing complexity without compromising performances; e) *low-overhead*, to not impact too much on normal system behaviors.

CPM is a cross-layer optimization framework for power and resource management, which implements a hierarchical distributed control model. An overall view of the proposed solution is depicted in **Figure 1**. This cross-layer approach splits the system-wide control problem into two different sub-problems: low-level devices' local controls and an higher-level distributed agreement control. Drivers run a device specific fine-tune policy, based on the system requirements and working conditions.

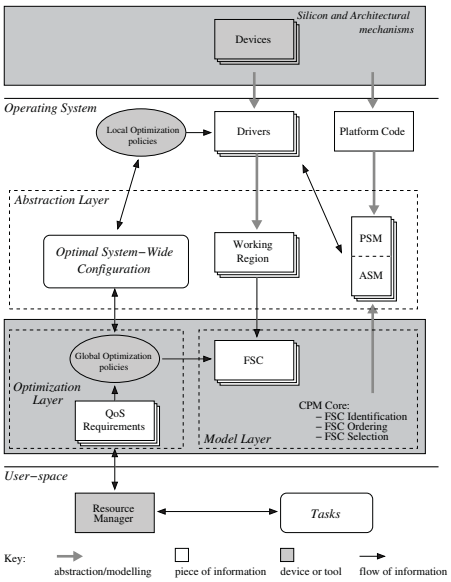

Figure 1: A real computing system, with many devices and applications, is so complex that it is not convenient to model all this complexity in order to solve the consumption and performances optimization problem. Therefore we perform an abstraction and modeling to properly support a platform independent optimization policy.

These policies could exploit the detailed knowledge on the specific device capabilities. The global optimization policy instead is implemented at a higher abstraction level and exploits both low-level informations, related to resource availability and hardware capabilities, and high-level informations related to application's QoS requirements.

Among the high-level global optimization policy and the many low-level local optimization policies, we focused our attention on the upper layer. This choice has a double motivation. On one hand, different local optimization policies have already been investigated, as described in 1.1, by many past researches which focused on different classes of peripherals. Some of these policies are also already implemented within the Linux kernel. On the other hand, the upper layer is the more interesting since, being the more abstract layer, the designed solution will be completely platform independent and most notably because, as we previously observed: there are not yet implementations which provides a complete and effective support for this layer within a real operating system like Linux.

2.1 Cross-layer framework design

From a design perspective CPM is composed by three different levels: a low-level providing "architecture abstraction", a middle-level for "platform modeling" and finally an high-level supporting the "global optimization policy".

The Abstraction Layer provides a suitable abstraction for: resources, architecture and devices. The "resources ab-

straction" defines the possible optimization directions. System resources, regardless of their nature either hardware (e.g., frequencies) or not (e.g., latencies), are represented by the concept of *system wide metrics (SWM)*[1]. Thus, the set of all available SWMs defines a *system-wide configuration space (SWCS)* which can be explored to search the optimal configuration. The "architecture abstraction" satisfies the fine-detail's requirement and allows to represent some platform specific details within the model. This is achieved by classifying the SWMs into: *abstract metrics (ASMs)* and *platform-specific metrics (PSMs)*. The former are abstract metrics, which are exposed to user-space, without compromising the solution portability, to collect QoS requirements asserted by applications. The latter are platform specific metrics, that allow the framework to consider details of the target system details, e.g., devices inter-dependencies.

Concerning the "devices abstraction" we considered that a device generally can have different working modes (WMs), which correspond to different resources usage and supported QoS. What exactly are the WMs of a device is defined by the corresponding driver. A WM could be as simple as 'device on' and 'device off', or even more complex such as all the different operating frequencies of a CPU or the different connection protocols supported by a 3G modem. Moreover, we considered that a device is *sensible* to a SWM if any change of this metric's value can imply a reconfiguration of the device and vice versa[2]. Based on these considerations, we introduced the concept of *device working region (DWR)*, which is an abstract representation of a device WM and it is defined by a set of ranges, one on each sensible SWM. Thus, each device in the system is described within the abstraction layer by a set of DWRs.

The Model Layer takes as input the information representing the system resources and capabilities, described by the abstraction defined in the lower layer, and generates as output an architecture independent representation of all the *feasible system-wide configurations (FSC)* available for the target system.

A number of interesting theoretical techniques can be defined to identify at run-time what is the optimal configuration of a single device, according to both the available resources and the required performances. However, every local configuration is potentially suboptimal, or even dangerous, with respect to system-wide performances and power consumptions, if either implicit inter-dependencies or hardware constraints are ignored by the optimization policy itself. Indeed, an optimized local configuration cannot be evaluated regardless of its system-wide feasibility.

In the light of the previous considerations we defined a model based on *feasible system-wide configurations (FSC)*. A FSC is a region, on the SWCS, defined by a set of validity ranges for each metrics. In these space regions it is granted that each device could be configured to operate in a WM that does not have any conflict with any other device. Thus, these regions are particularly important because they grant that every inter-dependency among devices is safely solved. Thanks to this interesting property, the identification of all system's FSCs is especially important. The optimization technique proposed is our approach is based on the a-priori identification of all and only the system feasible configurations. This allows to grant that, any optimization policy that will be developed on top of the framework, will operate on a set of system-wide configurations which are really available and system-wide valid.

The Optimization Layer exploits the abstract system-wide view offered by the underlying FSC model to support a global optimization policy. Thus, such a policy basically operate on FSCs, with the target to select the best one, considering both the optimization goals and the set of application QoS's requirements.

This global optimization problem could be solved in two steps: FSC ordering and FSC selection. In the first step, all the possible solutions are ordered according to their goodness with respect to the goals of the current multi-objective optimization policy. Since all the possible solutions are all and only the available FSCs, this step could be done by associating a *"goodness level"* to each feasible configuration. This goodness level can be easily computed, for instance by assigning a weight w_s to each metrics[3] $s \in SWM$ and resolving this simple classification function:

$$W_f = \sum_{s \in SWM} w_s m_{sf}, f \in FSC \qquad (1)$$

where m_{sf} is the value of the metric s in the FSC f.

The second step is related to the selection of the best FSC. While the first step takes care of the optimization goals, this time the focus is on the QoS's requirements of applications. Application requirements could be translated on constraints for the optimization problem. Indeed, QoS requirements are represented in our model by one or more ASM's validity range. Since feasible configurations are defined by metrics' ranges as well, it could happen that the assertion of a QoS requirement invalidate one or more FSCs. Considering a feasible configuration $f \in FSC$, it is:

$$valid(f) \Leftrightarrow \forall s \in SWM, compatible(r_{sf}, c_s) \qquad (2)$$

where r_{sf} is the validity range of f on the metrics s, c_s is the more restrictive validity range asserted on s, and:

$$compatible(r_1, r_2) \Leftrightarrow r_1 \subseteq r_2 \qquad (3)$$

At run-time, an invalid FSC cannot be the best solution of the optimization problem. The best solution of the optimization problem correspond instead to the FSC with the

[1]The term "metrics" was chosen to remark their numerical interpretation.

[2]For instance, a DVFS driver for the control of the processor clock frequency is sensible to a SWM like 'CPU Frequency'.

[3]SWMs represent the optimization directions, as described in the abstraction layer.

highest goodness level chosen among the one which are still valid considering all the currently asserted QoS constraints, i.e., the validity ranges on the SWMs.

2.2 Formal validation of the optimization policy

Just to provide a rigorous description of the problem and a formal proof of the solution quality, we formulated the optimization approach described so far as a Linear Programming (LP) problem. It is worth to notice that the formulation using such a formal model is not intended to be used for the effective solution of the problem itself. However, this formulation has been useful to identify a solution strategy which can be efficiently implemented into a Linux kernel, by exploiting all the particularities of the specific formulation. For instance, we should take into account that thanks to the abstraction and model levels we already know all the possible solutions of the problem.

A representation of the LP formulation is depicted in **Figure 2**. In this simple scenario we consider a system with three devices (d_1, d_2 and d_3) and two SWMs (p_1 and p_2). Given the working modes represented by the c_{dw} regions, the available FSCs are only three. At run-time it could happen that some of the feasible configurations (FSC_3) are invalidated by the constraints (v_3) representing application requirements. The set of valid FSCs identify always a convex region which is suitable to represent the convex-hull of an LP problem. Moreover, our global optimization policy could be represented by an objective function o_g which identifies the direction of a multi-objective optimization in the $SWCS$. Indeed, o_g is defined by the composition of multiple goals, each one corresponding to a different metric (p_1 and p_2) with an associated optimization priority ($\vec{o_1}$ and $\vec{o_2}$). The convex-hull and an objective function would allow to properly define a LP problem which, once applied a proper solution algorithm is able to find the optimal solution.
It is possible to prove, but it is also easy to convince ourself, that the solution of the LP problem, *O* in the example, can always be mapped to one[4] FSC. This configuration is granted by the LP theory to be the optimal one with respect to the optimization policy. This proof is left informal for space constraints, however what is important to notice it that it is possible to translate the proposed optimization problem into a formal LP problem which grants the identification of the optimal solution. Than we use this equivalence to design an efficient implementation which can be verified to find the same solution of the LP problem. Thus, this means that the solutions we will find using our implementation are also the optimal ones.

[4]Eventually more than one if the solution is not on a vertex.

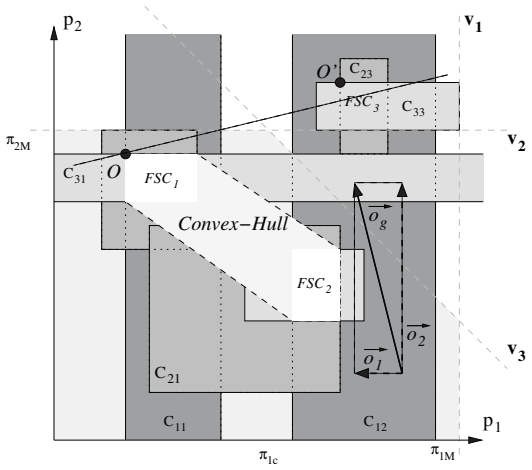

Figure 2: Example of LP formulation for the global optimization problem. Each c_{di} area represents the $i-th$ device working region of the $d-th$ device.

2.3 Framework implementation

The formal representation of the global optimization problem not only certifies the goodness of the configuration identified, but also suggests an efficient implementation of the solution algorithm, which consists of these three steps:
FSC identification: the collection of available FSCs are identified by the model layer according to the working modes defined by every device driver. This step is required only at system boot once device drivers register to the framework.
FSC ordering: a multi-objective optimization policy is settled by simply defining the optimization priority for each metric in the configuration space, i.e. by associating a 'weight' to each SWM. Thus, starting from the set of all FSCs which are identified by the model layer, it is possible to pre-order the solutions of the optimization problem. In the previous example, according to the optimization policy represented by the vector o_g, the solutions ordered starting from the best one: FSC_3, FSC_1 and finally FSC_2. This step is required only when the optimization goals changes, for instance when we switch from a certain 'power saving' policy to another 'performance boost' policy.
FSC selection: at run-time the requirements asserted by applications are properly translated in constraints for the optimization problem. These constraints are used to filter the list of available FSCs, invalidating all these that violate the constraints. The feasible and optimal solution is the first one valid in the list of the ordered FSCs. In the previous example, FSC_3 is invalidated by the assertion of the constraint v_3 and so the optimal solution is the next one valid: FSC_1. This last step is required every time an application assert a QoS requirement which invalidate the current FSC.

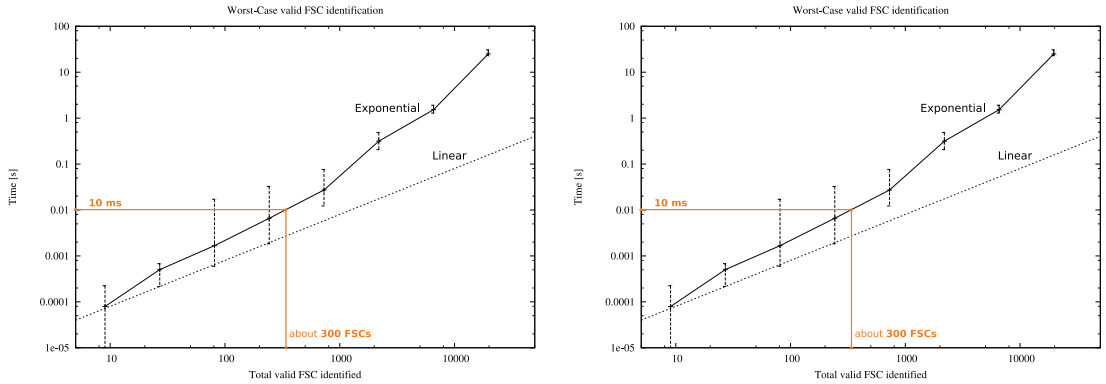

Figure 3: The worst-case performances of the FSC identification algorithm (a) and the *FSC selection* algorithms (b), both evaluated on increasing number of FSCs. While the former shows a exponential trend the latter is upper bounded by a linear trend. However, in both cases the measured overhead is negligible with respect to the average system-wide time required by the corresponding operating system activities.

As a final remark, it is worth to observe that these three steps could happen at different and decreasing timeframes: with the identification being the less frequent operation while the selection the more frequent. This separation at the base of the efficiency of the proposed solution, since as we would show in the experimental section, it allows to move the main complexity to the less frequent operations while keeping simples, and thus with very reduced overheads, the most frequent ones.

3 Experimental results

The proposed optimization model has been implemented as a Linux framework based on kernel 2.6.30[5]. This implementation has been used for a *worst-case* evaluation of the overheads introduced by the different optimization algorithms. Indeed, overheads evaluation is fundamental to show the applicability of the proposed solution in a real usage scenario to efficiently support the fine tuning of devices' local power and performances policies. Although Linux supports a number of devices managed by optimization policies, for the measurements of overheads they have not been directly used. The reason is twofold: it is not possible to setup a worst-case configuration using real devices and to effectively measure the framework overheads only. Thus, the tests has been performed on real hardware, using a demo-board equipped with the Nomadik STn8815 MPSoC by STMicroelectronics, but considering a set of dummy drivers explicitly designed[6] to emulate the required worst-case configurations in terms of number of working regions, available feasible configurations and selection of the best configuration.

The *FSC identification* algorithm (**Figure 3a**) has exponential complexity. However, it could be considered to actually run very fast especially if compared to typical system boot time, where this algorithm is used. Indeed, for example 300 FSC could be identified in about 10ms where the system considered using real devices count around 500 FSCs but require more than 15 seconds to boot. Moreover the algorithm can be replaced by a look-up table approach: in embedded system the configuration of the platform for a final product does not change and so all the FSCs can be pre-computed off-line and written into a file that is then loaded at boot. That way the impact of this algorithm at runtime is completely canceled.

The *FSC selection* algorithm (**Figure 3b**) has linear complexity. This is actually the time needed to scan the FSC's list until a valid candidate is selected. The overhead in this case is really negligible. For instance, just 1ms is required to scan 1250 FSCs. This time is orders of magnitudes lower than the average time interval between two consecutive constraint assertions by applications. Indeed, applications usually assert constraint on changes of the working use-case; we could reasonable expect that such events happens only time to time.

4 Conclusions

We have presented a system-wide power and resources' optimization model implemented on a real Linux kernel framework. This cross-layer framework collects QoS requirements from the application, aggregates them and produces constraints which are exploited to coordinate the coarse tuning of device's local optimization policies. One

[5]The implementation is continuously re-based on the mainline kernel version.
[6]These drivers are available along with the framework code release.

of the main results of the proposed control model is to efficiently support the tracking of hardware dependencies and to coordinate the behavior of the different subsystems composing a typical SoC-based platform for multimedia devices.
The proposed method has been formally justified through the equivalence with a well-known formal technique to solve optimization problems. The experimental validation showed negligible overheads which make the usage of the framework a promising solution to efficiently support system-wide optimization of both performances and energy, starting from the definition of dynamic and multi-objective global optimization policies.

References

[1] Venkatachalam, V., Franz, M.: Power reduction techniques for microprocessor systems. ACM Comput. Surv. **37**(3) (2005) 195–237

[2] Mohapatra, S., Cornea, R., Dutt, N., Nicolau, A., Venkatasubramanian, N.: 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) 582–591

[3] Fei, Y., Zhong, L., Jha, N.K.: An energy-aware framework for dynamic software management in mobile computing systems. ACM Trans. Embed. Comput. Syst. **7**(3) (2008) 1–31

[4] AbouGhazaleh, N., Childers, B., Mosse, D., Melhem, R., Craven, M.: 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) 284–293

[5] Zeng, H., Ellis, C.S., Lebeck, A.R., Vahdat, A.: Ecosystem: managing energy as a first class operating system resource. SIGPLAN Not. **37**(10) (2002) 123–132

[6] Anand, M., Nightingale, E.B., Flinn, J.: 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) 23–35

[7] Snowdon, D.C., Sueur, E.L., Petters, S.M., Heiser, G.: 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) 289–302