Motivations

Focusing the Problem

Power and resource management are key goals for the success of modern and future battery-powered multimedia devices, which are complex systems with multiple functionalities all embedded in a small set of components. These tiny devices are usually based on MPSoCs with a wide range of subsystems, that compete in the usage of shared resources and offer several power saving capabilities, but need an adequate software support to exploit such capabilities. Therefore, offering the best QoS level and user experience, while saving as much power as possible to improve mobile batteries life becomes a challenging goal that involves all the layers of a system, from the application level to the device driver level. Moreover, reduced time to market in the development of new products requires the definition of an agile methodology that permit portability of existing solutions and code across different devices and new product versions. 

The goal of a suitable solution is to permit coordination and communication, between different entities of the system, using dynamic optimization policies to grant the desired QoS levels and with negligible impact on the system. The main requirements for an efficient solution of the power vs performances optimization problem are:

<table>
<thead>
<tr>
<th>Global</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>Smoothly applies to share and make versatile architecture such as the upcoming many-core based systems, without impacting too much in design and portability.</td>
</tr>
<tr>
<td>Portability</td>
<td>Scalibility when board changes in devices or the system without requiring a complete re-design of the control solution.</td>
</tr>
<tr>
<td>Fine-Detail</td>
<td>Exploit this fine-grained knowledge about each device in order to improve the optimization policies.</td>
</tr>
<tr>
<td>System-Wide</td>
<td>Exploit the global view on system resources availability, power-cost model, and device working modes of each device and its applications requirements in order to achieve a system-wide optimal configuration.</td>
</tr>
<tr>
<td>Time-adaptability</td>
<td>Tune the control policy to changing usage scenarios, in order to better track the user expected performances.</td>
</tr>
<tr>
<td>Application pro-active</td>
<td>Exploit runtime capabilities and to give feedback on effective resource availability.</td>
</tr>
</tbody>
</table>

A hierarchical approach (Hi) to power management could ensure that all the main goals are achieved, due to an appropriate control of both local and system-wide parameters. Thus, compared to centralized (C) or distributed (D) approaches, hierarchical control is considered better suited for future many-cores architectures.

Hierarchical Power Management

Overview

A Hierarchical Power Management framework is based on the design of a single coordination entity which allows both to exploit a system-wide view of resources availability and to aggregate all the applications’ requirements. Resources availability is defined by device drivers in a platform independent way. These information can be properly exploited by the framework to support the system-wide optimization policy with fine-details and improved portability. The fine-details are granted by the low-level information collected by drivers. Since these low-level information are collected directly by drivers in a platform independent way and at run-time, the portability of the control solution improves. Indeed, changing architecture or even a single device the new information are automatically detected. Applications’ requirements are collected by a well defined and single user-space interface. The framework aggregates these requirements and translate them in a set of constraints for the global policy.

At run-time, the global optimization policy could exploit all the information collected either by drivers and applications. The former defining resources availability while the latter asserting QoS requirements. These information could be used to effectively solve a multi-objective optimization problem targeted to identify the best system-wide configuration. For scalability reasons, this configuration could not be completely defined by the coordination entity. Instead, this entity will notify proper constraints to drivers and let’s their local optimization policies to do the fine-tuning.

Abstracting Reality and Modeling the Abstraction

A suitable approach to support portability and scalability of the control policy is to define it on the base of an abstraction and modeling layers. An abstraction layer grants portability without compromising the fine-details requirement. Available resources and devices working modes are represented in a platform independent way. Resources are abstracted using a set of metrics (PSM/ASM) which can be used also to setup the multi-objective optimization problem. A working mode of a device can be also represented in the space of these metrics by identifying a corresponding device working mode (DWM). The model layer exploit these abstraction informations to automatically build a representation of all the system-wide feasible configurations (FSCs), each one identifying a working points of the entire system where a certain QoS level can be granted. This model is suitable for supporting an efficient global optimization strategy provided by the optimization layer. This policy should support a multi-objective optimization defined on the abstraction metrics.

2PARMA Integration

Design-Time Support

Design-time support comes from applications description and structure. From exploration of HW/SW codesign.

- application scenarios, and application description
- application constraints and QoS tradeoffs
- system-level performance model
- static information on application structure, and cooperation/competition profile
- exploitation of HW/SW policy design and their effectiveness

Output

- improved system-wide models for the analysis of run-time behavior
- overhead profile of the run-time management and power management support
- high-level system monitoring feedback
- data and memory management effectiveness

Run-Time Support

Run-time manager will monitor the executing environment through those observation points that are exposed from the underlying hardware. Dynamically, power-performance related decisions are taken, feeding back the platform with suitable tuning actions.

- components latency and throughput (e.g., communication links performance)
- memory and processor unit usage (e.g., communication load)
- application requirements and resource reservations
- per core and per chip status information (e.g., thermal, power consumption, faults, cache misses, etc...)
- low-level power management mechanisms

Optimizations

- core and device operation points
- constraints and resource managers agreement
- platform settings
- power/performance control knobs status

Formal Optimization Model

The global optimization policy relay on Linear Programming to identify a solution-equivalent and efficient optimization strategy. This strategy is based on three main tasks:

- FSC Identification – The abstraction layer provides a multi-dimensional solution space, defined by the metrics (m), and the device working regions (DWM) (c) defined by each device (d). These information are used to identify a corresponding device working mode (DWM). The model layer exploit these abstraction informations to automatically identify the set of Feasible System-Wide Configurations (FSC). This model is a platform independent representation of the system resources and capabilities.

- FSC Ordering – The global optimization policy exploits the FSCs’ based abstract system view provided by the model layer. The FSCs previously identified can be ordered according to the multi-objective optimization goal (g). Each time the use-case changes, and thus the optimization goal is shifted, the FSCs can be re-ordered.

- FSC Selection – Application requirements are collected and translated into constraints (h) which invalidate unfeasible configurations (e.g., FSC). The optimal FSC is selected considering both the previously identified ordering and their viability considering the current constraints. Each task has different run-time overheads and activation frequency. The identification is the more complex task, it is required just at system-boot. Instead, FSC selection must run each time an application assert a requirement but it has a negligible impact thanks to the support provided by the previous tasks.