Constrained Power Management
an holistic approach to power management

Patrick Bellasi
PhD Student

Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy

ELC-E - October, 15-16 2009
Outline

- Highlight some issues of current Linux kernel PM support
- Advance a proposal to tackle these problems
  - not a finale solution
  - try to focus attention on the topic
  - trigger a discussion to improve this kind of support
Power Management Techniques

- **Focusing on devices and interconnections**
  - almost any direct applications input

- **Device specific’s policies**
  - multiple policies for single device

- **System-wide Policies**
  - tracks subsystem’s *specific* dependencies
  - dependency tree
Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input

- Device specific’s policies
  - multiple policies for single device

- System-wide Policies
  - tracks subsystem’s *specific* dependencies
  - dependency tree

Simple representation of a complex system
Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input
- Device specific’s policies
  - multiple policies for single device
- System-wide Policies
  - tracks subsystem’s *specific* dependencies
  - dependency tree

Simple representation of a complex system
Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input
- Device specific’s policies
  - multiple policies for single device
- System-wide Policies
  - tracks subsystem’s specific dependencies
  - dependency tree

Simple representation of a complex system
Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input
- Device specific’s policies
  - multiple policies for single device
- System-wide Policies
  - tracks subsystem’s specific dependencies
  - dependency tree

Simple representation of a complex system
Multiple sub-system specific policies

Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input
- Device specific’s policies
  - multiple policies for single device
- System-wide Policies
  - tracks subsystem’s specific dependencies
  - dependency tree

Simple representation of a complex system
Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input
- Device specific’s policies
  - multiple policies for single device
- System-wide Policies
  - tracks subsystem’s specific dependencies
  - dependency tree

Simple representation of a complex system
Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input
- Device specific’s policies
  - multiple policies for single device
- System-wide Policies
  - tracks subsystem’s specific dependencies
  - dependency tree

Multiple disjoint policies

Simple representation of a complex system
Power Management Techniques

- Focusing on devices and interconnections
  - almost any direct applications input
- Device specific’s policies
  - multiple policies for single device
- System-wide Policies
  - tracks subsystem’s specific dependencies
  - dependency tree

Multiple disjoint policies

How can we achieve system-wide optimization?
Multiple-Policy Approach: Potential Issues

- **Multiple decision points**
  - difficult inter-dependencies tracking
  - risk of conflicting decisions

- **Only indirect info about applications QoS requirements**
  - user-space know the requirements, kernel should support them
  - application requirements should drive kernel frameworks tuning

- **No proper aggregation on applications requirements**
  - only some frameworks provide it (e.g V/I fw, “new” clock fw)
  - risk of code duplication

- **No feed-back on resources availability**
  - applications could require resources from multiple devices
  - behavior depends on effective availability of all the required resources

The composition of almost independent optimization policies cannot grant a system-wide optimization
Constrained Power Management

- drivers’ local policies
  - targeted to power reduction
  - fine-details, low-overhead

- *coordination entity*
  - exploit system-wide view
  - track resource availability and devices’ inter-dependencies

- global optimization policy
  - multi-objective, low-frequency

- single user-space interface
  - collects QoS requirements
  - feedback on resource availability

- constraint assertion
  - QoS requirements
  - set constraint on local policies
Constrained Power Management

- drivers’ local policies
  - targeted to power reduction
  - fine-details, low-overhead

- coordination entity
  - exploit system-wide view
  - track resource availability and devices’ inter-dependencies

- global optimization policy
  - multi-objective, low-frequency

- single user-space interface
  - collects QoS requirements
  - feedback on resource availability

- constraint assertion
  QoS requirements
  set constraint on local policies
Constrained Power Management

- drivers’ local policies
  - targeted to power reduction
  - fine-details, low-overhead
- *coordination entity*
  - exploit system-wide view
  - track resource availability and devices’ inter-dependencies
- global optimization policy
  - multi-objective, low-frequency
- single user-space interface
  - collects QoS requirements
  - feedback on resource availability
- constraint assertion
  - QoS requirements
  - set constraint on local policies
Constrained Power Management

- drivers’ local policies
  - targeted to power reduction
  - fine-details, low-overhead
- coordination entity
  - exploit system-wide view
  - track resource availability and devices’ inter-dependencies
- global optimization policy
  - multi-objective, low-frequency
- single user-space interface
  - collects QoS requirements
  - feedback on resource availability
- constraint assertion
  QoS requirements
  set constraint on local policies
Constrained Power Management

- drivers’ local policies
  - targeted to power reduction
  - fine-details, low-overhead
- coordination entity
  - exploit system-wide view
  - track resource availability and devices’ inter-dependencies
- global optimization policy
  - multi-objective, low-frequency
- single user-space interface
  - collects QoS requirements
  - feedback on resource availability
- constraint assertion
  QoS requirements
  set constraint on local policies
The PM QoS interface

Linux kernel infrastructure to implement a coordination mechanism among drivers (capabilities) and application (QoS requirements)

- Developed by Intel for iwl4965 WiFi driver on x86
  - since Linux 2.6.25 (linux/pm_qos_params.h)
- Defines a (limited) set of “abstract” QoS parameters
  - i.e. latencies, timeouts and throughput
  - maintains a list of QoS requests and aggregate requirements
    - restrictive aggregation only, i.e. Min/Max
    - this aggregation generates a constraint
  - provides notification chain for constraint update
    - drivers subscribe to parameters of interests
      - e.g. CPUIdle is constrained by ’system latency’
  - Drivers’ local policies should grant required constraints
    - no failures handling on notify chain calls
"The notion of constraint based PM has been rattling around for a while now. PMQoS is just an early application of it. I think a lot more could be done in this area." M. Gross

- Missing support for platform-specific parameters
- No **additive constraints** concepts support
- Only **best-effort** approach

```
E1    E2    QoS
pm_qos
current

OPTIMAL
20Mb/s
MAX 54Mb/s
```

Additive Constraint value
Limitations of the PM QoS Interface

“The notion of constraint based PM has been rattling around for a while now. PMQoS is just an early application of it. I think a lot more could be done in this area.” M. Gross

- Missing support for platform-specific parameters
- No **additive constraints** concepts support
- Only **best-effort** approach

![Diagram showing PM QoS Update Requirement](image-url)
“The notion of constraint based PM has been rattling around for a while now. PMQoS is just an early application of it. I think a lot more could be done in this area.” M. Gross

- Missing support for platform-specific parameters
- No additive constraints concepts support
- Only best-effort approach
Limitations of the PM QoS Interface

“The notion of constraint based PM has been rattling around for a while now. PMQoS is just an early application of it. I think a lot more could be done in this area.” M. Gross

- Missing support for platform-specific parameters
- No additive constraints concepts support
- Only best-effort approach
Limitations of the PM QoS Interface

“The notion of constraint based PM has been rattling around for a while now. PMQoS is just an early application of it. I think a lot more could be done in this area.” M. Gross

- Missing support for platform-specific parameters
- No additive constraints concepts support
- Only best-effort approach
Emerging system-wide optimization support

Limitations of the PM QoS Interface

“The notion of constraint based PM has been rattling around for a while now. PMQoS is just an early application of it. I think a lot more could be done in this area.” M. Gross

- Missing support for platform-specific parameters
- No additive constraints concepts support
- Only best-effort approach

```
pm_qos_update_requirement(bw, 18)
max(18, 20)
```

```
E1 QoS
```

```
OPTIMAL
```

```
pm_qos
```

```
32Mb/s
```

```
pm_qos_update_requirement(bw, 32)
max(20, 32)
```

```
E1
```

```
E2
```

```
QoS
```

```
20Mb/s
```

```
38Mb/s
```

```
MAX
```

```
38+32 Mb/s
```

```
54Mb/s
```

```
70Mb/s
```

```
Additive Constraint value
```

```
unuseful restrictive setting
```

```
grant
```

```
constraint value
```
Our Goals

- Define a **formal model** for system-wide *performance vs power* control
  - based on **constraints-based** approach
  - drivers could **collaborate** to find the optimal system-wide configuration
    - with respect to *all* QoS requirements
  - support multi-objective optimizations

- **Implementation** based on latest Linux kernel
  - overcoming current QoS limitations

- **Validate** the model and the implementation on real hardware
  - STM’s Nomadik platform
  - evaluate overheads and performances
Abstracting the Reality, Modeling the Abstraction

**Hierarchical Distributed Control**

- Devices’ local control
- Abstracting reality
- Modeling Abstraction
- Model optimization
- Considering QoS requirements
- Constraint local policies
Abstracting the Reality, Modeling the Abstraction

Hierarchical Distributed Control

- Devices’ local control
- Abstracting reality
- Modeling Abstraction
- Model optimization
- Considering QoS requirements
- Constraint local policies
Abstracting the Reality, Modeling the Abstraction

*Hierarchical Distributed Control*

- Devices’ local control
- Abstracting reality
- Modeling Abstraction
- Model optimization
- Considering QoS requirements
- Constraint local policies
Hierarchical Distributed Control

- Devices’ local control
- Abstracting reality
- Modeling Abstraction
- Model optimization
- Considering QoS requirements
- Constraint local policies
Abstracting the Reality, Modeling the Abstraction

Hierarchical Distributed Control

- Devices’ local control
- Abstracting reality
- Modeling Abstraction
- Model optimization
- Considering QoS requirements
- Constraint local policies
Abstracting the Reality, Modeling the Abstraction

**Hierarchical Distributed Control**

- Devices’ local control
- Abstracting reality
- Modeling Abstraction
- Model optimization
- Considering QoS requirements
- Constraint local policies
**System-Wide Metric (SWM)**

A *parameter describing the behaviors of a running system and used to track resources availability*

- QoS requirements: are expressed as *validity ranges* on SWM mainly upper/lower bounds
- Different abstraction levels
  - *Abstract System-wide Metric (ASM)*, platform independent exposed to user-land
    e.g. ambient light/noise, power source, specific application requirements
  - *Platform System-wide Metric (PSM)*, platform dependent private to platform code and platform drivers
    e.g. bus bandwidth, devices’ latency
- Allow to track QoS inter-dependencies
  - platform drivers and code can translate ASM’s requirements into PSM’s constraints

Code Example
The mapping on SWMs’ range of a device operating mode

- A device could have different working modes
  - different QoS $\Rightarrow$ different SWM range
- Defined by the device driver
- Implicitly allows devices dependencies tracking
- Graphic representation

A device with 3 DWR ($c_{dm}$) mapping 2 SWM ($p_i$)
The intersection of a least a DWR for each device

- QoS requirements within a FSC
  - all devices can support the required QoS level
  - no conflicts
- identify all and only the feasible system’s working modes
  - all the possible solutions for the PM optimization problem
  - define an abstract model for system-wide optimizations

The 3 FSCs existing on this system
A Formal Optimization Framework

• Using Linear Programming (LP)
  ◦ well known mathematical *multi-objective* optimization framework

• Two-fold goal
  ◦ formally justify the proposal
    • through the equivalence with a well known exact method for optimal solution search
  ◦ guide the design of an efficient implementation
    • we don’t want to solve an LP problem
    • identify possible simplifications
    • exploit problem specificities

*Use LP formulation to identify a solution-equivalent and efficient optimization strategy*
1. DWRs registration
2. FSC identification
3. Constraint aggregation
4. FSC validation
5. FSC selection
   - optimization policy $\overrightarrow{\sigma_G}$

- Different time domains
  - boot-time: steps 1-2
  - run-time: steps 3-5

- step 5 can be simplified by FSC pre-ordering
  - policy defined
e.g. $FSC_3$, $FSC_1$, $FSC_2$
Putting it All Together

1. DWRs registration
2. FSC identification
3. Constraint aggregation
4. FSC validation
5. FSC selection
   - optimization policy $\bar{\omega}_g$

- Different time domains
  - boot-time: steps 1-2
  - run-time: steps 3-5

- step 5 can be simplified by FSC pre-ordering
  - policy defined
    - e.g. $FSC_3$, $FSC_1$, $FSC_2$
Putting it All Together

1. DWRs registration
2. FSC identification
3. Constraint aggregation
4. FSC validation
5. FSC selection
   - optimization policy $\overrightarrow{\sigma_g}$

- Different time domains
  - boot-time: steps 1-2
  - run-time: steps 3-5
- step 5 can be simplified by FSC pre-ordering
  - policy defined
    - e.g. $FSC_3$, $FSC_1$, $FSC_2$
Putting it All Together

1. DWRs registration
2. FSC identification
3. Constraint aggregation
4. FSC validation
5. FSC selection
   - optimization policy $\mathbf{\bar{o}}_g$

- Different time domains
  - boot-time: steps 1-2
  - run-time: steps 3-5

- step 5 can be simplified by FSC pre-ordering
  - policy defined
    - e.g. $FSC_3$, $FSC_1$, $FSC_2$
Putting it All Together

1. DWRs registration
2. FSC identification
3. Constraint aggregation
4. FSC validation
5. FSC selection
   - optimization policy $\overrightarrow{o_g}$

- Different time domains
  - boot-time: steps 1-2
  - run-time: steps 3-5

- step 5 can be simplified by FSC pre-ordering
  - policy defined
    e.g. $FSC_3$, $FSC_1$, $FSC_2$
Putting it All Together

1. DWRs registration
2. FSC identification
3. Constraint aggregation
4. FSC validation
5. FSC selection
   - optimization policy $\bar{o}_g$

- Different time domains
  - boot-time: steps 1-2
  - run-time: steps 3-5

- step 5 can be simplified by FSC pre-ordering
  - policy defined
    - e.g. $FSC_3$, $FSC_1$, $FSC_2$
1. DWRs registration
2. FSC identification
3. Constraint aggregation
4. FSC validation
5. FSC selection
   - optimization policy $\overrightarrow{o_g}$

- Different time domains
  - boot-time: steps 1-2
  - run-time: steps 3-5

- step 5 can be simplified by FSC pre-ordering
  - policy defined
    - e.g. $FSC_3$, $FSC_1$, $FSC_2$
A Real Optimization Framework

- Translate the formal (LP) model into an efficient implementation
- Exploit tree different time domains
  - boot time $\Rightarrow$ FSC Identification (FI)
  - policy update time $\Rightarrow$ FSC Ordering (FO)
  - constraint assertion time $\Rightarrow$ FSC Selection (FS)
- Support complexity partitioning
  - high-overhead operations (FI) are executed once
- Modular design
  - split operations on “governor” and policy
  - better support operation optimization
    - off-line computation (FI)
    - HW acceleration, e.g. look-up based implementation (FO, FS)

Go to Overheads Graph
Framework Design

- framework core
  - data types, ASM
  - glue code
  - user-space API
- platform code
  - PSM definition
- device drivers
  - DWR definition
  - constraints auth.
- governor
  - FSC identification
- policy
  - FSC ordering
  - constraints auth.

Main framework components and their relationship
Framework Design

- framework core
  - data types, ASM
  - glue code
  - user-space API
- platform code
  - PSM definition
- device drivers
  - DWR definition
  - constraints auth.
- governor
  - FSC identification
- policy
  - FSC ordering
  - constraints auth.
Framework Design

- **framework core**
  - data types, ASM
  - glue code
  - user-space API

- **platform code**
  - PSM definition

- **device drivers**
  - DWR definition
  - constraints auth.

- **governor**
  - FSC identification

- **policy**
  - FSC ordering
  - constraints auth.

Main framework components and their relationship
Framework Design

- **framework core**
  - data types, ASM
  - glue code
  - user-space API

- **platform code**
  - PSM definition

- **device drivers**
  - DWR definition
  - constraints auth.

- **governor**
  - FSC identification

- **policy**
  - FSC ordering
  - constraints auth.

Main framework components and their relationship
Framework Design

- framework core
  - data types, ASM
  - glue code
  - user-space API

- platform code
  - PSM definition

- device drivers
  - DWR definition
  - constraints auth.

- governor
  - FSC identification

- policy
  - FSC ordering
  - constraints auth.

Main framework components and their relationship
Resuming the Proposal

- Distributed approach for performances vs power trade-off control
  - supports constraint based PM
  - scalable on upcoming more and more complex architectures
  - provides multi-objective optimizations

- Layered design
  - optimization layer on top of an abstraction layer
  - improved code reuse and portability

- Simple platform code and drivers interface
  - few modifications required
  - easily exploits platform and devices fine-details

- Validated using a formal optimization model

- Up-to-date implementation, rebased on mainline Linux kernel
  - providing a sysfs interface and some dummy test modules to support testing and benchmarking
Looking Forward

- The implementation is going to be released in ML for RFC
  - basic implementation of the designed software architecture
  - public GIT repository: still missing!
  - discuss, review, rework... community feedbacks are welcome!

- Find real-world applications
  - the constrained PM concept should be pushed
    - ... the QoS PM interface is almost unused
  - try it: it's free!

- Provide guidelines for DWR definition
  - distributed control assign different target to different levels
  - local policies should fit well within the model

- Improve the user-space interface
  - integration within a resource management system framework
  - automate constraint assertion

- Investigate on HW acceleration possibilities
Q&A
The PM optimization problem can be formulated as an LP problem.

**LP elements:**
- *solution space* – SWMs Domain
- *objective function* – vector representing QoS optimization directions
- *constraints* – QoS requirements
  - dynamically reduce the number of valid FSCs
- *convex hall* – the smallest convex polygon including all valid FSCs
- *valid solution* – every point inside the convex hall
- *optimal solutions* – vertexes or edges of the convex hall
  - can always be mapped to 1 or 2 FSCs
CPM Overheads

- **Worst-Case Analysis**
  - synthetic drivers to configure the worst operating conditions
  - running on VirtualBox, host: Intel Core 2@1.6GHz
  - note: non-Cartesian logarithmic X axis

---

Hoverheads % wrt 60s timeframe

Go to Implementation Notes
$ git diff 4be3bd78.. --stat
Documentation/cpm/00-INDEX.txt | 61 +
Documentation/cpm/core.txt | 264 +++
Documentation/cpm/governors.txt | 146 ++
Documentation/cpm/overview.txt | 131 ++
Documentation/cpm/platform.txt | 123 ++
Documentation/cpm/policies.txt | 131 ++
Documentation/cpm/testing.txt | 67 +
Documentation/cpm/user-guide.txt | 139 ++
drivers/Kconfig | 2 +
drivers/Makefile | 1 +
drivers/cpm/Kconfig | 112 ++
drivers/cpm/Makefile | 10 +
drivers/cpm/cpm_core.c | 3402 +++++++++++++++++++++++++++++++++++++
drivers/cpm/cpm_governor_exhaustive.c | 420 +++
drivers/cpm/cpm_policy_dummy.c | 122 ++
drivers/cpm/cpm_policy_performance.c | 199 ++
drivers/cpm/test/Kconfig | 38 +
drivers/cpm/test/Makefile | 7 +
drivers/cpm/test/cpm_test_bandwidth.c | 218 +++
drivers/cpm/test/cpm_test_dummy.c | 459 ++++
drivers/cpm/test/cpm_test_mp3gsm.c | 316 +++
include/linux/cpm.h | 491 ++++
22 files changed, 6859 insertions(+), 0 deletions(-)
// SWM Identifiers definitions
#define SWM_AMBA_BANDWIDTH  CPM_ASM_TOT+0
#define SWM_ADSP_CLK       CPM_ASM_TOT+1

// Platform specific SWM (PSM) definition
struct cpm_swm cpm_platform_psm[] = {
    CPM_PLATFORM_SWM("AMBA_BANDWIDTH", CPM_TYPE_GIB, CPM_USER_RW,
                      CPM_COMPOSITION_ADDITIVE, 0, 8000),
    CPM_PLATFORM_SWM("ADSP_CLK", CPM_TYPE_GIB, CPM_USER_RO,
                      CPM_COMPOSITION_ADDITIVE, 0, 266),
};

// PSM Registration
struct cpm_platform_data cpm_platform_data = {
    .swms = cpm_platform_psm,
    .count = ARRAY_SIZE(cpm_platform_psm),
};
cpm_register_platform_psms(&cpm_platform_data);
Example - Device Working Region

```c
struct cpm_swm_range vdsp_dwr0_ranges[] = {
    /* V-DSP MPEG4 decoding mode */
    DEV_DWR_ASM(CPM_VCODEC, 1, 1, CPM_ASM_TYPE_RANGE),
    DEV_DWR_ASM(CPM_DSP_CLK, 40, 132, CPM_ASM_TYPE_RANGE),
};
struct cpm_swm_range vdsp_dwr1_ranges[] = {
    /* V-DSP OFF mode */
    DEV_DWR_ASM(CPM_VCODEC, 0, 0, CPM_ASM_TYPE_RANGE),
    DEV_DWR_ASM(CPM_DSP_CLK, 0, 132, CPM_ASM_TYPE_RANGE),
};
struct cpm_dev_dwr vdsp_dwrs_list[] = {
    /* V-DSP working mode */
    DEV_DWR("Mpeg4", vdsp_dwr0_ranges, ARRAY_SIZE(vdsp_dwr0_ranges)),
    DEV_DWR("OFF", vdsp_dwr1_ranges, ARRAY_SIZE(vdsp_dwr1_ranges)),
};
static struct cpm_dev_data vdsp_data = {
    /* V-DSP DWR’s registration */
    .notifier_callback = vdsp_cpm_callback,
    .dwrs = vdsp_dwrs_list,
    .dwrs_count = ARRAY_SIZE(vdsp_dwrs_list),
};
ret = cpm_register_device(&vdsp.dev, &vdsp_data);
```
Example - Real Use-Case

- Youtube and Torrent
  - 3 ASM:
    - acodec, vcodec, bandwidth
  - 2 PSM:
    - cpu_freq, cpu_dsp_platform
  - devices(dwr):
    - modem(5), vcodec(4), acodec(4), cpu(7), platform(7)
  - Identified FSC: 415
    - 10% out of 3920 of worst case analysis