Applying three principles for efficiency in the (mobile) system stack

Lin Zhong, Rice University
http://recg.org
• Proportionality
  – Use the right hardware for a software task

• Parsimony
  – Idle components should sleep

• Proximity
  – Waste should be discarded close to source
• Proportionality
  – Use the right hardware for a software task

• Parsimony
  – Idle components should sleep

• Proximity
  – Waste should be discarded close to source
Hardware Reality: Heterogeneous Coherence Domains for 1000x Power Difference

Both axes are logarithmic
Hardware Reality: Heterogeneous Coherence Domains for 1000x Power Difference

Both axes are logarithmic

e.g., ARM big.LITTLE

~10x  ~5x
Hardware Reality: Heterogeneous Coherence Domains for 1000x Power Difference

Both axes are logarithmic
Mobile Devices with Multiple Coherence Domains on-chip or on-board

Windows Phone
ST STM32

MS Surface Pro
Atmel UC3

Moto X
TI MSP430

Google Glass
Cortex-M3

Apple A7

iPhone 5S
Apple M7
Architectural Model: Heterogeneous Coherence Domains

- Highly asymmetric cores
- Separated coherence domains
Software Goal:
Span Heterogeneous Coherence Domains
Software Goal:
Span Heterogeneous Coherence Domains

App developers face a programming challenge!

Pinning OS services to strong cores is inefficient!
Applying Proportionality Principle

• Exploit a common boundary to support legacy applications and simplify app development
K2: a single system image over heterogeneous cores on chip

K2: a single system image over heterogeneous cores on chip

Key insight: Core vs Extended OS Services

**Extended Services**
- Device Drivers
- Network Stack
- File Systems
- ...

**Core Services**
- Virt mem
- Page alloc
- Sched
- IRQ

Linux OS

- A large set
- By many parties
- Less perf-critical

- A small set
- Perf-critical
The Shared-Most OS Model

Extended Services

Core Services

Core Services

Cores

Cores

Cores

Shared-nothing Instances

Transparently shared
K2: an implementation of the Shared-Most OS Model
Applying Proportionality Principle

• Exploit a common boundary to support legacy applications and simplify app development

– K2: system image as boundary for heterogeneous cores
Another hardware reality:
More devices, more types of devices
Another hardware reality: More devices, more types of devices

Can K2 be extended to multiple systems?
Applying Shared-Most model to multiple systems
Applying Shared-Most model to multiple systems

Extended Services

Software Distributed Shared Memory

Core Services

Core Services

Ext2
Ext3
FAT
Single system image over multiple systems is hard

- First step: how to unify I/O devices?

Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong, "Rio: a system solution for sharing I/O between mobile systems," ACM MobiSys’14. (received Best Paper Award)
Applying Proportionality Principle

• Exploit a common boundary to support legacy applications and simplify app development
  – K2: system image as boundary for heterogeneous cores
  – Rio: device file as boundary for I/O devices
Unify multiple systems
• **Proportionality**
  – Use the right hardware for a software task

• **Parsimony**
  – Idle components should sleep

• **Proximity**
  – Waste should be discarded close to source
Too many modules on mobile SoC
Die photo of OMAP4430, red rectangle shows Cortex A9 cores

Idle components should sleep

Device drivers should not do power management
Power management is hierarchical, involves both software & hardware.
Power management starts with software disabling a module.
Software enables/disables a module

power domain

ON

clock domain

clock domain

Software Trigger

I2C
...

GPIO
Hardware triggers domain state transition
Hardware triggers domain state transition

power domain

ON

clock domain

Hardware Trigger

I2C

...  GPIO

clock domain

clock domain
Hardware triggers domain state transition

- power domain
- clock domain
- clock domain
- clock domain
- I2C
- ... (omitted)
- GPIO

Hardware Trigger
Problems with driver power management

#1 Not implemented for many months (years)

#2 Improperly implemented

Hierarchical power management amplifies #1 & #2
    – Unmanaged module => unmanaged domain
Why is power management so hard?

• Implementing sleep(), wakeup() is easy

• Where to call sleep()/wakeup() is hard
Solution to bad device driver PM?
Solution to bad device driver PM?

- Get better driver developers
What information power management needs

- Users’ QoS requirements
- Module’s wakeup latency, remote wakeup capability, etc.
- Whether a module has pending task

Available in the device drivers
What information power management needs

- Users’ QoS requirements
- Module’s wakeup latency, remote wakeup capability, etc.
- Whether a module has pending task

Available in the device drivers
Also available outside the device drivers!
Solution: Central Power Management

- Drivers register PM callbacks at system initialization

Drivers do not contribute code to central PM

Chao Xu, Xiaozhu Lin, and Lin Zhong, "Device drivers should not do power management," ACM APSYS’14
Applying Parsimony Principle

• Idle components should sleep

• Rethink runtime power management architecture
  – Relieve drivers from power management
• **Proportionality**
  – Use the right hardware for a software task

• **Parsimony**
  – Idle components should sleep

• **Proximity**
  – Waste should be discarded close to source
Data path of computer vision apps
Simplified data path

Image sensing → Image processing → Computer vision

Specialized hardware → Main mem. → General purpose processor
1.3 Watt just to produce an image

Image sensing  Image processing  Computer vision

Specialized hardware  Main mem.  General purpose processor

0.3 Watt  1.0 Watt
1.3 Watt just to produce an image

Image sensing → Specialized hardware → Image processing → Main mem. → General purpose processor → Computer vision

0.3 Watt → 1.0 Watt

2.1 watt-hours ÷ 1.3 watts = 1.5 hours

Every watt raises the surface temperature by 11°C!
Rethink image sensing!

Image sensing  Image processing  Computer vision

Specialized hardware  Main mem.  General purpose processor

0.3 Watt  1.0 Watt

- Photography ➔ Computer vision

Robert LiKamWa, Bodhi Priyantha, Matthai Philipose, Lin Zhong, and Paramvir Bahl, “Energy characterization and optimization of image sensing toward continuous mobile vision,” ACM MobiSys’13 (received Best Paper Award)
Applying Proximity Principle!

- Rethink image sensing
  - Photography ➔ Computer vision

- Introduce programmability early in the path
  - App knows what it needs
  - What interface for apps to provide intent?

Image sensing ➔ Image processing ➔ Computer vision

Specialized hardware ➔ Main mem. ➔ General purpose processor

0.3 Watt ➔ 1.0 Watt
Acknowledgement

http://recg.org
• **Proportionality**
  – Use the right hardware for a software task

• **Parsimony**
  – Idle components should sleep

• **Proximity**
  – Waste should be discarded close to source