Distributed Execution of Smartphone Workloads on Loosely Coupled Processors

Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong

Rice University
Goal: energy-proportionality

Smartphones
High performance for intensive workloads
Conserve energy as workloads mitigate

Tens of milli Watt ↔ 100x ↔ A few Watt
Architectural support for energy-proportionality
Core

Performance

Power

E.g. ARM big.LITTLE, Nvidia Tegra3

Core

Heterogeneous
Tight coupling

DVFS
Core Performance

DVFS

Heterogeneous Tight coupling

Heterogeneous Loose coupling

E.g. TI OMAP4/5

Power

Performance

Processor

Core
Challenges

• Exploit loosely-coupled processors for smartphone workloads

• Programmability
  – 800K apps, by 1 million programmers
  – Numerous legacy code: 1 million lines in Android
Characterize smartphone workloads

Temporal / spatial variations?

App / OS split?

All experiments done on Samsung Galaxy S2 + ICS
CPU Usage

Time / Sec

Home-idle
CPU Usage

Time / Sec

Home-idle
Observation 1

Workloads are time-varying

CPU Usage

Time / Sec

User Swiping

Clock Animation

Home-idle
Observation 2

Same threads experience varying workloads

- Rest of System
- UI Thread

Home-idle
Observation 2

Same threads experience varying workloads

---

**CPU Usage**

- Rest of System
- Thread: User
- Thread: Kernel

**Time / Sec**

- 0%
- 20%
- 40%
- 60%
- 80%
- 100%

**Home-idle**
Use of OS Services

Home-idle (light)  VS  Browser (heavy)
Browser : Heavy
Home-idle: Light
Observation 3

Same OS functions experience varying workloads

# of Unique OS Kernel Functions

By Browser Only
By Home-idle Only
By Both

OS Subsystems

kernel | drivers | fs | net | mm | lib | arch | block | security | sound

0 | 50 | 100 | 150 | 200 | 250 | 300 | 350 |
Observation 4

Workloads invoke diverse OS functions
Implications: for user threads

– Ob1: Workloads are time-varying
– Ob2: Same threads experience varying workloads

➢ Enable dynamic use of different processors
➢ Retain existing program structure
Example: UI thread of Home

CPU Usage

Time / Sec

User Swiping → Clock Animation

Rest of System | UI Thread

Home-idle
Example: UI thread of Home

CPU Usage

Time / Sec

Home-idle
Implications: for OS functions

– Ob3: Same OS functions experience varying workloads
– Ob4: Workloads invoke diverse OS functions

➤ Execute same OS services on multiple processors
➤ Preserve a single system image
Example: Instant Messenger

```
s=socket()
```

App

OS

Big

Small
Example: Instant Messenger

App
recv(s, ...)

OS

Z z z...
Our solution: Kage runtime lib + OS

- Kage Runtime
- Non-coherenet Shared Mem
- Big
- Small

Apps

Kage OS

OS component

OS component
Kage: runtime lib

Apps

Kage Runtime

Replicated

Thread

Thread

Kage OS

OS component

OS component

Big

Small

Non-cohereenet Shared Mem
Kage: OS structure

Kage OS

Non-coherent Shared Mem

Local Kernel

Big

Small

Sensor

Net

Display

Replicated

Thread

Apps

Kage Runtime

27
Kage: OS structure

[Diagram showing the structure of the Kage OS with layers for Local Kernel, Kage Runtime, and Legacy OS Code, along with components like Sensor, Net, Display, Big, and Small, connected through arrows labeled "Replicated" and "Non-coherent Shared Mem"]
Open questions and plans

• State consistency
• Limiting inter-processor concurrency

• Implementation: TI OMAP4 (two types of processors)
• Two local kernels with separate memory allocators
Related Work

Heterogeneity with HW cache-coherence
  – big.LITTLE

OS with partitioned functions
  – Helios, Barrelfish (for scalability)

OS with replicated functions
  – V, fos
Summary

Goal: energy proportionality for smartphone workloads

Observations:
- Architecture: loosely coupling, asymmetry
- Workloads: high variations, in both app and OS

Challenges:
- Exploiting the architecture for smartphone workloads
- Maximizing reuse of legacy code

Solution: Kage, a suite of runtime lib and OS
- Replicating user thread execution
- Replicating OS functions