Transkernel: Bridging Monolithic Kernels to Peripheral Cores

Liwei Guo, Shuang Zhai, Yi Qiao, and Felix Xiaozhu Lin

Purdue ECE

http://xsel.rocks
What is Transkernel?

➔ A novel OS model to run **unmodified binary of a monolithic kernel**

➔ on a **microcontroller-like core** ...

➔ of a **heterogeneous SoC**

➔ Key techniques: dynamic binary translation + kernel service emulation
Motivation: Ephemeral tasks in smart things

**Prevalent:** push notifications, periodic data logging, etc.
  ○ User tasks running on a monolithic kernel (e.g., Linux)

**Energy-hungry:** ~30% or higher battery drain in smart things [1]

Device suspend/resume is the key bottleneck [2]

---

[1] Smartphone background activities in the wild: Origin, energy drain, and optimization, Chen et al., *MobiCom’15*
[2] Decelerating Suspend and Resume, Zhai et al., *Hotmobile’17*
Why is device suspend/resume so inefficient?

**Slow power state transitions**  
keep CPU waiting

**Difficult to parallelize due to device dependencies**

Understanding modern device drivers, Kadav et al., ASPLOS’12
Our proposal: suspend/resume on a peripheral core

- **Benefit:** Lower idle power and higher busy execution efficiency

- Asymmetric processors
  - CPU + Peripheral core

- Heterogeneous, yet similar ISAs
  - Same family, different profile
  - e.g., ARMv7a + v7m

- Loose coupling
  - CPU can be turned on/off independently

- Shared platform resources
  - IRQs
  - DRAM

Apple A9 (Chipworks)
Many SoCs fit this hardware model

OMAP4460
Cortex M3 + A9
2010

AM572x
Cortex M4 + A15
2014

i.MX 7
Cortex M4 + A7
2017

iPhone 6
Cortex M3
2014

Azure Sphere
Cortex M4 + A7
2019
Problem statement

➔ On a heterogeneous SoC

➔ Given a commodity monolithic kernel (e.g., Linux)

➔ How to offload device suspend/resume kernel phase to the peripheral core?

The desired workflow
Design space exploration: multikernel

However...
Design space exploration: code transplant

However...

Suspend Resume

Drivers
Driver lib
Kernel services & lib

Linux kernel

CPU
Peripheral Core
Peripheral kernel
Kernel State
IO
Design space exploration: full virtual execution

However...

> 25x overhead with current DBT
Our proposal: Transkernel

- Goal: Linux kernel offloading with affordable overhead
- Approach: the peripheral core dynamically translates the kernel binary, supported by a small set of emulated kernel services
Transkernel in the design space

- QEMU [ATC’05]
- Transkernel
- K2 [ASPLOS’14]
- Popcorn [Eurosys’15]
- Barrelfish [SOSP’09]
- M3 [ASPLOS’16]
Principle 1: translate stateful code, emulate stateless

- Stateful vs. stateless: whether the states of the kernel are shared across cores

- Translated code: state-sharing made easy

- Emulated services: drop-in replacement
Principle 2: identify narrow trans/emulation interface

- The interface has to be:
  - Narrow
  - Stable

- Maintenance of emulated services made easy
Principle 3: specialize for hot paths

- Hot paths: 99% of executions
  - Encounter no errors
  - All needed resources acquired
- Going off? Fall back to CPU
- Simplify DBT implementation on a peripheral core
Principle 4: exploit ISA similarity

● Between the ISAs of CPU & peripheral core:
  ○ General purpose registers
  ○ Control flow registers (SP, LR, PC)
  ○ Flag semantics (NZCV)

● Reducing the number of emitted instructions in DBT
● Key to low overhead!
ARK: an ARM transKernel

Platform: OMAP4460 (Cortex A9+M3)

ARK instantiates the principles on Linux
- Execute unmodified Linux kernel binary on the peripheral core
- Depend on stable ABIs (only 12 functions + 1 variable)
- Focus on hot paths; may fall back to CPU
- Low-overhead ARM v7a -> v7m DBT

Translated Code (stateful)

Stable ABI

Emulation (stateless)
ARK: the cross-ISA DBT engine

- Systemize similar semantics of ARM v7m & v7a from formal specification [1]
- Most instructions have identical semantics (447)
- Others instructions ...
  - Side effect
  - Constant constraints
  - Shift modes
- Our DBT engine correctly executes over 200 million instructions!

<table>
<thead>
<tr>
<th></th>
<th>v7a insn count</th>
<th>Each translated to # of v7m insns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>447</td>
<td>1</td>
</tr>
<tr>
<td>Side effect</td>
<td>52</td>
<td>3-5</td>
</tr>
<tr>
<td>Const constraints</td>
<td>22</td>
<td>2-5</td>
</tr>
<tr>
<td>Shift modes</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>No counterparts</td>
<td>27</td>
<td>2-5</td>
</tr>
<tr>
<td>Total (v7a)</td>
<td>558</td>
<td></td>
</tr>
</tbody>
</table>

[1] Trustworthy Specifications of ARM v8-A and v8-M System Level Architecture, Reid et al., FMCAD’16
Evaluation

• Does ARK ...
  a. Incur low-overhead?
  b. Incur tractable engineering efforts?
  c. Yield efficiency benefit?

• Benchmarks setup
  • Test the whole suspend/resume phase, driven by a userspace test harness
  • Diverse drivers: SD card, Flash drive, MMC controller, USB controller, Regulator, Keyboard, Camera, Bluetooth NIC, Wi-Fi NIC
ARK’s DBT achieves low execution overhead

![Graph showing overhead for various components: SD Card, Flash, MMC-Ctrl, USB-Ctrl, Regulator, KB, Cam, BT, Wi-Fi. The graph compares Baseline (square bars) and ARK (filled bars). The overhead is significantly reduced from 25x to 2.7x when transitioning from Suspend to Resume.]
ARK reuses Linux with low efforts

- Good code reuse: 10K vs. 40K and even more
- Good compatibility: multiple versions and configurations of Linux kernel

<table>
<thead>
<tr>
<th>New implementation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DBT</td>
<td>9K SLoC</td>
</tr>
<tr>
<td>Emulation</td>
<td>1K SLoC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Existing code (unchanged)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Translated</td>
<td>15K SLoC</td>
</tr>
<tr>
<td>Substituted w/ emu</td>
<td>25K SLoC</td>
</tr>
</tbody>
</table>
End-to-end execution time & energy

- Time: prolonged execution time
- Energy: 34% energy saved
- Interesting finding: ARK sees higher DRAM energy

### Accumulated Time (s)

<table>
<thead>
<tr>
<th></th>
<th>Idle</th>
<th>Busy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

### Energy (mJ)

<table>
<thead>
<tr>
<th></th>
<th>IO</th>
<th>DRAM</th>
<th>Core busy</th>
<th>Core idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td></td>
<td>681</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What-if analysis

ARK energy: 66%

w/o optimization energy: 333%

(2.7x, 41%)

(13.9x, 41%)
Take-home messages

- Transkernel & its key techniques
  - An appropriate translation/emulation boundary inside a monolithic kernel
  - Exploit ISA similarity

- To OS
  - A new model to span a monolithic kernel over heterogeneous cores

- To DBT
  - Efficiency loss can enable efficiency gain
  - DBT applies to translate a specific path of a complex software stack!

- To Architects
  - A heterogenous SoC friendly to transkernel