Decelerating Suspend and Resume in Operating Systems

XSEL, Purdue ECE

Shuang Zhai, Liwei Guo, Xiangyu Li, and Felix Xiaozhu Lin

02/21/2017

http://xsel.rocks
• Mobile/IoT devices see many short-lived tasks
  • Sleeping for a long time; Waking up frequently
    • Smartwatch: 72 times per day
  • Each task is short-lived
    • Smartwatch: 10 secs
    • Background task: < 1 sec
Suspend/Resume OS Workflow

Suspend

1. CPU ON
2. Sync Filesystem
3. Freeze Tasks
4. Call IO Drivers
5. CPU OFF

Resume

1. CPU OFF
2. Call IO Drivers
3. Thaw Tasks
4. CPU ON
Suspend/Resume Is Expensive

• Slow suspend/resume is long known for desktop/server
  • Suspend/resume mostly slowed down by SATA and USB devices
  • These machines suspend/resume only occasionally

• Much worse on mobile/IoT due to short-lived tasks
  • Suspend/resume takes ~500 ms on Samsung Note4 Smartphone
  • E.g. consume 43% of total energy on sensing benchmark[1]

• Need to understand suspend/resume on mobile/IoT devices

Profiling Suspend/Resume

Nexus 5  Samsung Gear  Samsung Note 4  Panda Board
Suspend/Resume on Mobile SoC Is Slow

<table>
<thead>
<tr>
<th></th>
<th>Nexus 5</th>
<th>Gear</th>
<th>Note 4</th>
<th>Panda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suspend</td>
<td>119 ms</td>
<td>191 ms</td>
<td>231 ms</td>
<td>262 ms</td>
</tr>
<tr>
<td>Resume</td>
<td>88 ms</td>
<td>159 ms</td>
<td>316 ms</td>
<td>492 ms</td>
</tr>
</tbody>
</table>
Main Reason: IO Power Transitions Are Slow

- Suspend
- Resume
Slow IOs Are Various and Diverse

<table>
<thead>
<tr>
<th>Nexus 5</th>
<th>Gear</th>
<th>Note 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial_hsl</td>
<td>mdss</td>
<td>pcieh</td>
</tr>
<tr>
<td>mmc_host</td>
<td>mmc_host</td>
<td>dwmmmc2</td>
</tr>
</tbody>
</table>

Top IO devices

<table>
<thead>
<tr>
<th>Nexus 5</th>
<th>Gear</th>
<th>Note 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>187</td>
<td>120</td>
<td>116</td>
</tr>
</tbody>
</table>

Number of IO drivers for each platform
Alternative Solution: Async IO Power Transitions

- Asynchronous PM **overlaps** power transitions of multiple IO devices
- Key difficulty: dependencies among hundreds of IO devices
  - Subtle and implicit
  - OS may not know them
- Linux kernel community has a long debate
- Still very conservative about async PM
Our Key idea

Objective
Offloading suspend/resume to a very weak core

Hardware support
A weak core (common on mobile SoCs)

Software support
A baremetal virtual executor on the weak core

Offloading suspend/resume via virtualization
Weak Core on Modern SoCs

• Low power cores already exist on modern SoCs
  • E.g. Apple motion coprocessor (Cortex-M3)
  • Shared memory and IO bus; incoherent cache domain
  • Heterogeneous but similar ISA (ARMv7/8 vs ARMv7m)
Weak Core on Modern SoCs

• Low power cores already exist on modern SoCs
  • E.g. Apple motion coprocessor (Cortex-M3)
  • Shared memory and IO bus; incoherent cache domain
  • Heterogeneous but similar ISA (ARMv7/8 vs ARMv7m)

• Weak cores are ideal for executing OS suspend/resume
  • Idle power 3.8 mW vs 30 mW
  • Kernel execution favors weak cores [1]
    • Small code working set
    • Less predictable control flow

Software Challenges

• Objective: Execute the kernel suspend/resume path on a weak core, without cache coherence and without a unified ISA

• Manually partitioning mature kernels is infeasible
  • Modern kernels are beasts
    • Windows: 45M SLoC\textsuperscript{1}
    • Linux 4.4: 16M SLoC\textsuperscript{2}
    • Suspend resume code is complicate (30k SLoC in Linux)

• Commodity kernels are rapidly evolving

2. https://www.linuxcounter.net/statistics/kernel
Our Solution

• Launching a virtual machine on the weak core to execute *unmodified* kernel binary for the main CPU

• This contrasts with traditional virtualization
  • Host is much more powerful than guest
System Overview

Commodity

- Main Core
  - CPU ON
  - CPU OFF
  - CPU ON

- Weak Core
  - CPU ON
  - CPU OFF
  - Weak ON
  - Suspended

Our System

- Main Core
  - CPU ON
  - CPU OFF
  - Weak ON
  - Suspended

- Weak Core
  - CPU ON
  - Weak OFF

Suspend

Resume

Binary translation of unmodified kernel
Does This Really Work?

- No one would believe binary translation works for us
  - We need aggressive optimizations
- ~20x slow down from initial implementation
- Reason: commodity binary translators are **generic** and **conservative**
  - Status register is emulated
  - Frequent Interrupt Check
Our Key Optimizations

- Exploit ISA similarity (ARMv7 vs ARMv7M)
- Baremetal stacks
- Relaxed handling of interrupts and exceptions
- Kernel virtual memory
Current Implementation

- Platform: TI OMAP4 SoC
- Trimming down QEMU from 2.6M SLoC to 50.5K SLoc
- 4.5K SLoC new code
- A first-of-its-kind virtualization environment on an embedded core
Microbenchmarks

- **Performance metric:** # of CPU cycles
- **Baseline:**
  - native compilation & execution on the main CPU (Cortex-A9)
- **Native:**
  - native compilation & execution on weak core (Cortex-M3)
- **Translated (unoptimized/optimized):**
  - translated execution on weak core

### Overhead in terms of cycle count

- **callback**
- **kfifo**
- **glob**
Microbenchmarks

<table>
<thead>
<tr>
<th>Native</th>
<th>Translated (unoptimized)</th>
<th>Translated (optimized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>0X</td>
<td>5X</td>
</tr>
<tr>
<td>kfifo</td>
<td>0X</td>
<td>5X</td>
</tr>
<tr>
<td>glob</td>
<td>0X</td>
<td>5X</td>
</tr>
</tbody>
</table>

Overhead in terms of cycle count

- **Performance metric:** # of CPU cycles
- **Baseline:**
  - native compilation & execution on the main CPU (Cortex-A9)
- **Native:**
  - native compilation & execution on weak core (Cortex-M3)
- **Translated (unoptimized/optimized):**
  - translated execution on weak core

- **Optimization Result:**
  - 5x overhead reduction
  - 2x within native execution
- **Estimated Energy Saving:**
  - 70% energy reduced in suspend/resume
  - 30% overall battery life extended

Summary

• **Observation**: Busy/idle waits for IOs bottleneck OS suspend/resume path

• **Goal**: Offloading suspend/resume to a weak core with incoherent cache and heterogenous ISAs

• **Key idea**: Binary translate and execute **unmodified** kernel on weak core

• **Highlight**: For the first time we run a virtual environment on an embedded core for offloading specific kernel paths
Q/A
## ARM big.LITTLE

<table>
<thead>
<tr>
<th>SoC</th>
<th>Little Core Power</th>
<th>Big Core Power</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exynos 5430</td>
<td>85 mW (Cortex A7)</td>
<td>750 mW (Cortex A15)</td>
<td>8.8</td>
</tr>
<tr>
<td>Exynos 5433</td>
<td>189 mW (Cortex A53)</td>
<td>1480 mW (Cortex A57)</td>
<td>7.8</td>
</tr>
<tr>
<td>OMAP 4460</td>
<td>21.1 mW (Cortex M3)</td>
<td>672 mW (Cortex A9)</td>
<td>31.8</td>
</tr>
</tbody>
</table>

Power Consumption Comparison between ARM big.LITTLE and OMAP4

<table>
<thead>
<tr>
<th>SoC</th>
<th>Performance</th>
<th>Energy</th>
<th>Performance/Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>A15 (Exynos 5430)</td>
<td>99.69MB/s</td>
<td>19.75mWh</td>
<td>~5.04</td>
</tr>
<tr>
<td>A7  (Exynos 5430)</td>
<td>77.93MB/s</td>
<td>10.56mWh</td>
<td>~7.38</td>
</tr>
<tr>
<td>A57 (Exynos 5433)</td>
<td>155.29MB/s</td>
<td>27.72mWh</td>
<td>~5.60</td>
</tr>
<tr>
<td>A53 (Exynos 5433)</td>
<td>109.36MB/s</td>
<td>17.11mWh</td>
<td>~6.39</td>
</tr>
</tbody>
</table>

BaseMark OS II - XML Parsing Energy Efficiency
How do we estimate our energy saving

• Without offloading:
  • \( E_{cpu} = (T_{busy\_exec} + T_{busy\_wait}) \times P_{busy} + T_{idle} \times P_{idle} \)

• With offloading:
  • \( E_{pm} = X \times F \times T_{busy\_exec} \times P'_{busy} + T_{busy\_wait} \times P'_{busy} + T_{idle} \times P'_{idle} \)
Prior Art: Multikernel OSes

- One kernel for each type of cores
  - Helios [1]
  - Barrelfish [2]
  - K2 [3]
  - Popcorn Linux [4]
- Kernels often pass messages to communicate
- They give up compatibility with commodity kernels