Improving Energy Efficiency On Asymmetric Multiprocessing Systems

Work done while assigned to Linaro,
Joint work with Dietmar Eggeman and Robin Randhawa (ARM Ltd.)
Overview

- ARM big.LITTLE Architecture
- ARM big.LITTLE Energy-Efficiency Strategy
- Review: Morten Rasmussen Approach
- RCU and big.LITTLE Energy Efficiency
ARM big.LITTLE Architecture
What is big.LITTLE?

- “Continuation of voltage/frequency scaling by other means”
  - Low voltage and frequency: Per-transistor leakage dominates power
  - Therefore need to reduce the number of transistors
  - But need the full complement of transistors for good performance

- Solution: Have two sets of CPUs
  - Cortex A15: “big” CPUs with deep pipelines and many functional units
    - Optimized for performance
  - Cortex A7: “LITTLE” CPUs with short pipelines and few functional units
    - Optimized for energy efficiency
  - Get high performance and good energy efficiency

- Two basic configurations:
  - big.LITTLE switcher
  - big.LITTLE MP (the focus of this talk)
What is big.LITTLE MP?

- Each physical CPU, whether big or LITTLE, is separately visible to kernel and to applications.
- Allows any combination of big and LITTLE CPUs to be used concurrently, but requires more awareness of big.LITTLE:
  - Emulation: Need software before hardware is available
  - Applications: Need application awareness until kernel support
  - Test workloads: Need to enable tests without big.LITTLE hardware
  - CPU hotplug: There will be corner cases...
  - Scheduler: Automation of CPU choice is the ideal
ARM big.LITTLE Architecture

- Twice as fast
  - Cortex-A15
  - Cortex-A15

- ~3 times more energy efficient
  - Cortex-A7
  - Cortex-A7
  - Cortex-A7

big

LITTLE
ARM big.LITTLE Architecture

Twice as fast

Cortex-A15

Cortex-A15

~3 times more energy efficient

Cortex-A7

Cortex-A7

Cortex-A7

big

LITTLE

Unfortunately, the Linux kernel assumes all CPUs are similar...
ARM big.LITTLE Schematic
ARM big.LITTLE Energy-Efficiency Strategy
ARM big.LITTLE Energy-Efficiency Strategy

- Run on the LITTLE by default
- Run on big if heavy processing power is required
  - Power down big CPUs when not needed
- In other words, if feasible, run on LITTLE for efficiency, but run on big if necessary to preserve user experience
  - Use big CPUs for media processing, rendering, etc.
  - This suggests that RCU callbacks should run on LITTLE CPUs, possibly also for timers and other low-priority asynchronous events
  - Key point: Goal of big.LITTLE scheduling is to distribute tasks unevenly to handle different energy-efficiency and performance goals
  - Unlike traditional SMP, it now matters where a task is scheduled
Review: Morten Rasmussen Approach
Without Morten Rasmussen Approach: Bad!!!

Example: Android UI render thread execution time.

It matters *where* a task is scheduled.

4 core SMP

2+2 big.LITTLE (emulated)
Review: Morten Rasmussen Approach

- Based on Paul Turner's entity load-tracking patches

- Strategy: Run all tasks on LITTLE cores unless:
  - The task load is above a fixed threshold, and
  - The task priority is default or higher

- Implementation:
  - Set up big and LITTLE sched domains without load balancing
  - When long-running high-priority task awakens, run it on a big core.
  - Periodically check for high-priority tasks transitioning into long-running mode, and migrate them to big CPUs

- Performance results rival those on a system with all big cores
  - (See next slide)
With Morten Rasmussen Approach: Pretty Good!

- Example: Android UI render thread execution time.

![SurfaceFlinger exec. time histogram (100 runs)]

It matters *where* a task is scheduled.

- 4 core SMP
- big.LITTLE aware scheduling
- 2+2 big.LITTLE (emulated)
RCU and big.LITTLE Energy Efficiency
What is RCU? (AKA Read-Copy Update)

- For an overview, see http://lwn.net/Articles/262464/
- For the purposes of this presentation, think of RCU as something that defers work, with one work item per callback
  - Each callback has a function pointer and an argument
  - Callbacks are queued on per-CPU lists, invoked after “grace period”
  - Deferring the work a bit longer than needed is OK, deferring too long is bad (splat after 20 seconds) – but failing to defer long enough is fatal
  - RCU allows extremely fast & scalable read-side access to shared data
RCU:
Tapping The Awesome Power of Procrastination
For Two Decades!!!
RCU:
Tapping The Awesome Power of Procrastination
For Two Decades!!

But Procrastination has a Dark Side...
Procrastination's Dark Side: Eventually Must Do Work

Likely waking up a big CPU, needlessly chewing up lots of energy!!!
Two Ways Of Conserving Energy

- Offload RCU callbacks to LITTLE CPUs
- Use RCU_FAST_NO_HZ to reduce wakeups
**Base Case**

call_rcu()

big CPU

Busy

Grace Period

CB

LITTLE CPU

Busy  Busy  Busy
1: ARM big.LITTLE With RCU Callback Offloading

- call_rcu()
- Busy
- Grace Period
- CB
- Busy
- Busy
- CB
- Busy

Slower... But 3x better energy efficiency
1: ARM big.LITTLE With Reduced Wakeups

call_rcu() Must wait for fourth “jiffy”

Grace Period

Busy

Busy

Busy

Busy
## Results Summary

Actual energy measurements taken on real hardware.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Reference</th>
<th>RCU Processing Offload</th>
<th>Enforced Idle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E (J)</td>
<td>T (s)</td>
<td>ECU Processing Offload</td>
</tr>
<tr>
<td>cyclitest</td>
<td>1.75</td>
<td>3.13</td>
<td>1.47</td>
</tr>
<tr>
<td>sysbench</td>
<td>32.68</td>
<td>9.14</td>
<td>31.61</td>
</tr>
<tr>
<td>andebench2</td>
<td>59.51</td>
<td>20.54</td>
<td>57.91</td>
</tr>
<tr>
<td>andebench8</td>
<td>174.04</td>
<td>45.87</td>
<td>170.37</td>
</tr>
<tr>
<td>audio</td>
<td>7.87</td>
<td>30.01</td>
<td>7.39</td>
</tr>
<tr>
<td>audio+bbench</td>
<td>99.05</td>
<td>156.29</td>
<td>93.02</td>
</tr>
</tbody>
</table>
Summary: Which is Better?

- Both produce real benefits
  - Offloading gives slightly better wall-clock time
  - Enforced idle gives slightly better energy efficiency
  - Combining them does not help

- Both are needed
  - Offloading for real time and reduced OS jitter
  - Enforced idle for SMP energy efficiency

- Offloading other deferred operations may be helpful
  - Timers, workqueues, ...
Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.

- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.

- Linux is a registered trademark of Linus Torvalds.

- Other company, product, and service names may be trademarks or service marks of others.
Questions?