LAS16-105
Walkthrough of the EAS kernel adaptation to the Android Common Kernel

Juri Lelli
Senior Software Engineer / ARM Ltd.

Linaro Connect
Las Vegas 2016

©ARM 2016
Recap – What is EAS?

- Scheduling tasks while considering the energy implication

- System topology
- Workload
- Task Scheduler
- Frequency selection
- CPU selection
- Idle state selection
Recap – Why EAS?

- Holistic “joined-up” power/perf management in the kernel
- Sensible defaults for performance, including asymmetric systems
- Optional alternative to the default throughput oriented nature of the Linux scheduler: energy efficiency
- Simplified tuning interface (some level of tuning still required!)
- ... that makes it possible to couple an “aware” runtime with the kernel in well defined ways
Recap – Why EAS?

- Holistic “joined-up” power/perf management in the kernel
- Sensible defaults for performance, including asymmetric systems
- Optional alternative to the default throughput oriented nature of the Linux scheduler: energy efficiency
- Simplified tuning interface (some level of tuning still required!)
- ... that makes it possible to couple an “aware” runtime with the kernel in well defined ways

That’s why EAS has landed in the AOSP kernel!
Recap – Why EAS?

- Holistic “joined-up” power/perf management in the kernel
- Sensible defaults for performance, including asymmetric systems
- Optional alternative to the default throughput oriented nature of the Linux scheduler: energy efficiency
- Simplified tuning interface (some level of tuning still required!)
- ... that makes it possible to couple an “aware” runtime with the kernel in well defined ways

That’s why EAS has landed in the AOSP kernel!
EAS - The story so far

All policy, all metrics, all averaging should happen at the scheduler power saving level, in a single place, and then the scheduler should directly drive the new low level idle state driver mechanism.
- Ingo Molnar (31 Mar 2013)

2013

- ARM EAS RFCv5
  - Jul-2015

2014

- Linaro shallowest idle
  - Sep-2014 MERGED

2015

- ARM Schedtune RFC
  - Aug-2015

- Linaro/ARM SchedFreq
  - RFCv7
  - Feb-2016

- ARM Foundation patches
  - MERGED
  - Aug-2015

- ARM LISA tooling
  - Jan-2016

- ARM Asym CPU capacity support
  - v4 - tip
  - Aug-2016

2016

- Intel Schedutil
  - MERGED
  - Apr-2016

- ARM CPU capacity DT
  - v7
  - Sep-2016

- ARM Foundation patches MERGED
  - Aug-2015

- ARM LISA tooling
  - Jan-2016

- ARM Asym CPU capacity support
  - v4 - tip
  - Aug-2016

2017

- EAS in Android Common Kernel
  - Aug-2016

LAS16
Finally.. today’s agenda

- Energy Aware Scheduling: high level introduction
- Android Common Kernel: adaptations breakdown
- Upstream and product code lines
- How to get involved
Agenda

- Energy Aware Scheduling: high level introduction
- Android Common Kernel: adaptations breakdown
- Upstream and product code lines
- How to get involved
Energy Aware Scheduling – Detailed view
Asymmetric capacity support
Energy model driven task placement
EAS – SchedFreq

Scheduler driven DVFS
EAS – SchedTune

Centralized tuning mechanism
EAS – Userspace

Userspace interaction through SchedTune
Agenda

- Energy Aware Scheduling: high level introduction
- Android Common Kernel: adaptations breakdown
- Upstream and product code lines
- How to get involved
EAS – Baseline

- What we call EAS v5.2+
- Good for mainline, but not quite there for Android
  - performance still meant throughput, but latency matters a lot!
  - heuristics designed for classical b.L., might need change with !b.L. topology
  - mainline load tracking good for server/desktop workloads, not so for mobile
  - maintainers asked for a single tuning knob, might be too simplistic
EAS Modifications – Approach

- Primary focus on workloads that matters for the product
- Latency sensitive workloads, such as YouTube, Camera record, UiBench, Jankbench, etc.

1. run baseline (using LISA [1])
2. figure out sub-optimal behaviour by trace inspection or perceived/measured performance
3. brainstorm!
4. propose a change and compare with the baseline (perf and/or energy)
5. looks good? ship it!
6. goto 2

- Primary focus was on UX, secondary focus on competitive benchmarks Geekbench, Vellamo, etc.

[1] Update on LISA hacking session
EAS Modifications – An example (listview\(^1\))

![Graph showing frame rendering duration (ms) vs. number of samples](image)

[1] Coupling schedutil with EAS-core hacking session
EAS Modifications – An example (listview$^1$)

[jank frames!!!]

99th perc.

[1] Coupling schedutil with EAS-core hacking session
Android Common Kernel adaptations

DT

arch (arm, arm64) topology shim

energy model

CIE

FIE

Task scheduler

CFS

PELT

EAS

wakeup path

WALT

SchedFreq

cpufreq

cgroups

SchedTune

Userspace

cpuidle

©ARM 2016
EAS – load tracking

- **Window Assisted Load Tracking**
  - alternative approach (w.r.t. PELT) to track tasks demand and CPUs utilization
  - “windows vs. decayed signal” [1]
- Configuration time option and procfs interface to switch between WALT and PELT

**WHY?**

Better service of latency sensitive applications while meeting energy consumption targets

Limited device testing has shown WALT to be better

**UPSTREAM VERSION UNDER DISCUSSION**

- eas-dev

**PLANS**

- Several modifications and fixes for PELT
- Attend LAS16-311 “Window Based Load Tracking (WALT) versus PELT utilization” and hacking session for more information

[1] WALT vs PELT presentations and hacking sessions
EAS – SchedTune

- prefer_idle flag
- negative boosting

**WHY?**
prefer_idle expresses the desire for low-latency wakeups

Negative boosting makes background / non latency sensitive tasks appear smaller (thus favouring packing and lower clock frequency selection)

**PLANS**

Improve the definition of the power/perf trade-off

Post an updated RFC on LKML (OPP and CPU selection biasing)
EAS – Task wakeup path

- adaptations guarded by
  `/proc/sys/kernel/sched_is_big_little`
  `sched_is_big_little = 0` (SMP-like)
- implemented by `find_best_target()`
  1. establish if task is boosted and if it prefers idle CPUs
  2. all CPUs available on the system are considered
     - starting from first CPU for non boosted tasks
     - starting from last CPU for boosted tasks
  3. a CPU is chosen looking at
     - task (boosted) utilization
     - `prefer_idle`
     - CPU capacity and current utilization
  4. no energy model if boosted or `prefer_idle` and candidate is idle

- CPU discarded if task doesn’t fit
- first idle CPU for a `prefer_idle`
- CPUs with capacity at higher OPP annotated as backup
- enough capacity at current OPP preferred
  - lowest utilization for `prefer_idle`
  - highest utilization for `!prefer_idle`
EAS – Task wakeup path

- adaptations guarded by
  /proc/sys/kernel/sched_is_big_little
  sched_is_big_little = 0 (SMP-like)
- implemented by find_best_target()

WHY?
Flat CPU selection heuristic better fits SMP(-ish) topologies
Tuned to service boosted and latency sensitive tasks
Doesn’t touch original EAS selection heuristic

PLANS
Wakeup path subject to change in upstream codeline
Consider possibilities for merging !big_little and big_little wakeup paths
EAS – SchedFreq

- change “go to max” policy
- up/down throttling thresholds
  - quick in responding to sudden demand
  - hysteresis for brief drops

**WHY ?**
Improve responsiveness while achieving energy savings
(switching too frequently harms energy)

**PLANS**
Superseded by upstream schedutil governor
Port required features to schedutil
Attend LAS16-307 “Benchmarking Schedutil in Android” and hacking session for more information
Userspace

- Keep the tuning out of the kernel and in more “aware” run-time layers
- Tuning happens via a single, localised surface
- Android subsystems and the kernel are wired up via SchedTune
- Attend “AOSP Energy Aware Scheduler Integration” session for more details ...
- but some are still worth showing here as well
Userspace – EAS and Activity Manager

- **prefer_idle** = 0
  - boost = 0

- **prefer_idle** = 1
  - boost = 0
  - boost = 10
Userspace – EAS and Activity Manager

**BACKGROUND**
- prefer_idle = 0
- boost = 0

**FOREGROUND**
- prefer_idle = 1
  - boost = 0

**TOP_APP**
- prefer_idle = 1
  - boost = 50

**touchboost**
- POWER_HINT_INTERACTION
Agenda

- Energy Aware Scheduling: high level introduction
- Android Common Kernel: adaptations breakdown
- Mainline and product code lines
- How to get involved
Mainline codeline

- Mainline strategy (after EASv5)
  - split huge patchset into smaller chunks and get them merged
    BKK16-311: EAS core – upstreaming strategy goo.gl/AxCkK5
  - moving integration branch tracking tip/sched/core goo.gl/zw9iGC

![Diagram showing timeline and integration process]
Product codeline

- **Android Common Kernel**
  - https://android.googlesource.com/kernel/common/+log/android-4.4
  - next target will likely be Android 4.9

- LSK
Product codeline – development

Upstream

AOSP

AOSP Gerrit

Comprehensive testing
And further review

Downstream

LSK

eas-dev

New feature / fix
discussion

Patch(es) reviewed

AOSP

New feature / fix
discussion

Comprehensive testing
And further review
Agenda

- Energy Aware Scheduling: an introduction
- Android Common Kernel: adaptations breakdown
- Mainline and products code lines
- How to get involved
How to get involved

- Discussions on Linux Kernel Mailing List
- tested-by: and reviewed-by:
- Android Common kernel Gerrit
  https://android-review.googlesource.com/#/q/status:open+project:kernel/common
- eas-dev mailing list
  https://lists.linaro.org/pipermail/eas-dev/
- #linaro-eas IRC channel on freenode
<table>
<thead>
<tr>
<th>day</th>
<th>presentation</th>
<th>hacking session</th>
<th>Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon</td>
<td>Las16-103: Walkthrough of the EAS kernel adaptation to the Android Common Kernel</td>
<td>Update on LISA</td>
<td>30</td>
</tr>
<tr>
<td>Tue</td>
<td>Las16-207: Bus scaling QoS</td>
<td>Bus scaling QoS</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>14:00 / 1h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Las16-207: Bus scaling QoS</td>
<td>Cpuidle in details - theory</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>15:00 / 1h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Las16-207: Bus scaling QoS</td>
<td>Cpuidle in details - the code</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>16:00 / 1h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Las16-207: Towards an EAS energy model from DT</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Wed</td>
<td>Las16-307: Benchmarking Scheduling in Android</td>
<td>Benchmarking Scheduling in Android</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>11:15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Las16-311: Window Based Load Tracking (WALT) versus PEIT utilization</td>
<td>Coupling scheduling with EAS-core</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>12:10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Las16-311: Window Based Load Tracking (WALT) versus PEIT utilization</td>
<td>PELT behavior and known limitations</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>13:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thu</td>
<td>Las16-410: Don’t waste power when idle with runtime PM</td>
<td>Don’t waste power when idle with runtime PM</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>12:10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Las16-410: Don’t waste power when idle with runtime PM</td>
<td>Power Farm Evolution</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>15:00 / 1h</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Las16-410: Don’t waste power when idle with runtime PM</td>
<td>Power Farm Evolution</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>16:00 / 2h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>