Better compute control for Android using SchedTune and SCHED_DEADLINE

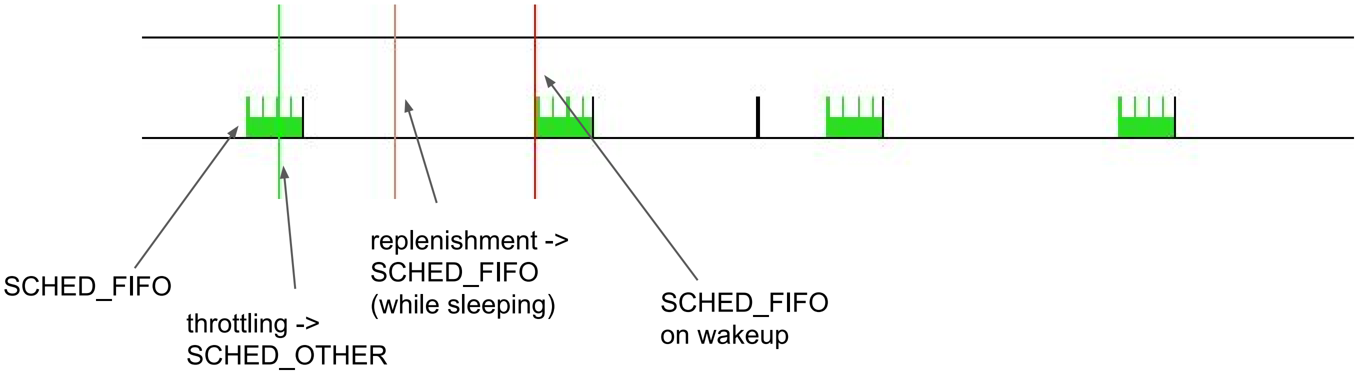
Patrick Bellasi, Juri Lelli (ARM)
Srinath Sridharan (Google)

SCHED_FIFO in Android (today)

- Used for some latency sensitive tasks
  - SurfaceFlinger (3-8ms every 16ms, RT priority 98)
  - Audio (<1ms every 3-5ms, low RT priority)
  - schedfreq kthread(s) (sporadic and unbounded, RT priority 50)
  - others
- Other latency sensitive tasks that are NOT SCHED_FIFO
  - UI thread (where app code resides, handles most animation and input events)
  - Render thread (generates actual OpenGL commands used to draw UI)
  - not SCHED_FIFO because
    - load balancing CPU selection is naive
    - RT throttling is too strict
    - Risk that these tasks can DoS CPUs

SCHED_FIFO (and beyond?)

- use SCHED_FIFO for UI and Render threads
  - Userspace support already in N-DR (to be released in AOSP in Dec timeframe)
  - EAS integrated RT cpu selection in-flight (to be part of MR2 release)
  - Results: ~10% (90th), ~12% (95th) and ~23%(99th) improvements in perf/Watt for jank benchmarks
- TEMP_FIFO
  - demote to CFS instead of throttling (RT throttling)



SCHED_DEADLINE (instead of SCHED_FIFO?)

- ✓ long term ambition is to provide better QoS using SCHED_DEADLINE
  https://linuxplumbersconf.org/2015/ocw//system/presentations/3063/original/lelli_slides.pdf
- ✓ if prototyping results are positive, mainline adoption of required modifications should be easier to achieve (w.r.t. modifying SCHED_FIFO)
- x missing features
  - https://github.com/jlelli/sched-deadline/wiki/TODOs
    - reclaiming (short term flexibility)
    - integration with schedutil
    - cgroup based scheduling
      - demotion to CFS

guinea pig for next steps will probably be SurfaceFlinger
(16ms period, 3-8 ms runtime)

SchedTune in a Nutshell

- Enables the collection of task related information from **informed runtimes**
  - using a **localized tuning interface** to balance **Energy Efficiency** vs **Performance Boost**
  - extending Sched{Freq,Util} for **OPP Selection** and EAS for **Task Placement**
- OPP Selection: running at higher/lower OPP
  - makes a CPU appear artificially more (or less) utilized than it actually is
  - depending on which tasks are **currently active** on that CPU
- Task Placement: biasing CPU selection in the wake-up path
  - based on evaluation of the **power-vs-performance trade-off**
  - using a **performance index** definition which helps define:
    how much power are we willing to spend to get a certain speedup for task time-to-completion?
- Uses CGroups to provide both **global** and **per-task** boosting
  - simple yet effective support for task classification
  - allows for more advanced use-cases where the boost value is tuned at run-time
    e.g. replace powersave/performance governors, support for touch boosting...

A New Design Proposal for SchedTune

SchedTune	Extending CPU Contoller
Boost value	Using the existing **cpu.shares** attribute. - by default tasks have a 1024 share - boosted tasks gets a share >1024 (more CPU time to run) - negative boosted tasks gets <1024 (less CPU time to run)
OPP biasing	Add a new **cpu.min_capacity** attribute. Tasks in the group *may be scheduled* on a CPU which provides at *least this required minimum capacity*
Negative boosting	Add a new **cpu.max_capacity** attribute. Tasks in the group are *never scheduled* (when alone) on a cpu with a higher CPU capacity than this value
CPU selection and prefer_idle	The **cpu.shares** value can be used as a “flag” to know when a task is boosted. E.g. if cpu.shares > 1024 (or another configurable threshold value) we look for an idle CPU. The **cpu.{min|max}_capacity** can also bias the selection of a big|LITTLE CPU.
Latencies reduction	Tasks with higher **cpu.shares** values are entitled to get more CPU time and this improves their chance to get scheduled by preempting other tasks with lower share values. **NOTE:** the CPU bandwidth that is not consumed by tasks with high **cpu.shares** value is still available for tasks with lower share values.

Backup slides

SCHED_FIFO (and beyond?)

Tasks scheduled by SCHED_FIFO today (framework may not be aware of all, e.g. drivers internals)

Task	Priority	Period	Load
camera HFR request thread	1	33 ms	3-4ms (measured on Angler with CPU @ 1.344GHz)
mmcqd	1	unknown	.2-.5ms (measured by taking heavy IO systraces in extremely CPU constrained situations on bullhead)
audio client	2	3 to 5 ms depending on the audio HAL	< 1 ms (not enforced)
FastMixer	3	3 to 5 ms depending on the audio HAL	< 1 ms
kernel IRQ	50	unknown	unknown
cfinteractive	99	unknown	unknown
surfaceflinger	98	16ms	3 to 8 ms (measured on Pixel)
EDS thread in VR	99	12ms (not sure)	50ms (not sure)

SchedTune Backup Slides

SchedTune Discussion Points

- Is the **CGroups interface** a viable solution for mainline integration?
  - CGroups v2 discussions about per-process (instead of per-task) interface?
  - Are the implied overheads (e.g. for moving tasks) acceptable?

- How can we improve the definition of SchedTune’s **performance index**?
  - How much is task performance affected by certain scheduling decision?
  - How can we factor in all the potential slow-down threat?
    e.g. co-scheduling, higher priority tasks, blocked utilization, interrupts pressure, etc

- Is **negative boosting** useful? Can we prove useful and improve the support for **negative boosting**?
  - Where/When is useful to **artificially lower** the perceived utilization of a task?
    identify use cases, e.g. background tasks, memory bounded tasks

Performance Boosting: What Does it Means?

- Speedup the time-to-completion for a task activation
  - by running at an higher capacity CPU (i.e. OPP)
    - i.e. small tasks on big cores and/or using higher OPPs
- To achieve such a goal we need:
  - A) Boosting strategy
    - Evaluate how much “CPU bandwidth” is required by a task
  - B) CPU selection biasing mechanism
    - Select a Cluster/CPU which (can) provide that bandwidth
    - Evaluate if the energy-performance trade-off is acceptable
  - C) OPP selection biasing mechanism
    - Configure selected CPU to provide (at least) that bandwidth
    - ... but possibly only while a boosted task is RUNNABLE on that CPU
  - ... do all that with no noticeable overhead

Patches Availablity and List Discussions

- The initial full stack has been split in two series
  - 1) Non EAS dependant bits
    - OPP selection biasing
    - Global boosting strategy
    - CGroups based per-task boosting support

Posted on LKML as RFCv1[1] and RFCv2[2]

  - 2) EAS dependant bits
    - CPU selection biasing
    - Energy model filtering

Available on AOSP and LSK for kernels 3.18 and v4.4 [3,4]

[1] https://lkml.org/lkml/2015/9/15/679
[2] http://www.mail-archive.com/linux-kernel@vger.kernel.org/msg1259645.html
[3] https://android.googlesource.com/kernel/common/+/android-3.18
[4] https://android.googlesource.com/kernel/common/+/android-4.4

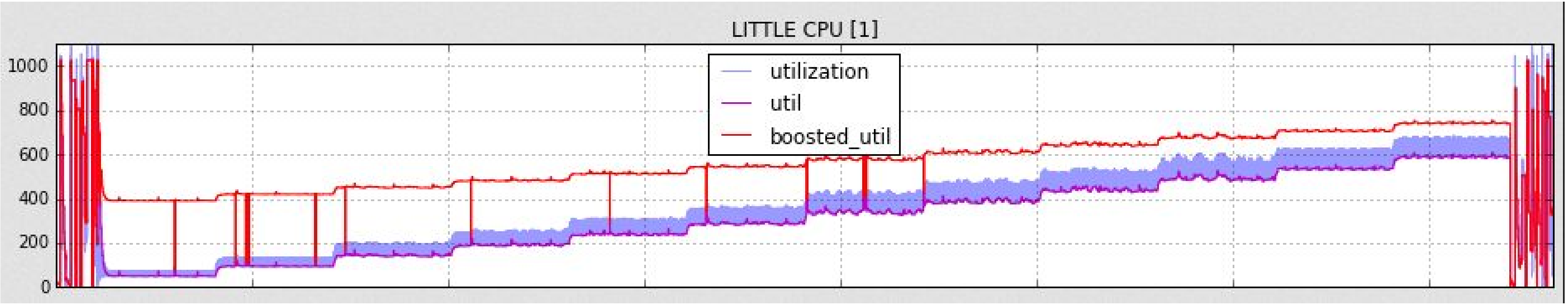
Boosting Strategy: Bandwidth Margin Computation

- Task utilization defines the task's required CPU bandwidth
  - To boost a task we need to inflate this requirement by adding a “margin”
  - Many different strategies/policies can be defined
- Main goals
  - Well defined meaning from user-space
    - 0% boost run @ min required capacity (MAX energy efficiency)
    - 100% boost run @ MAX possible speed (min time to completion)
    - 50%? ==> “something” exactly in between the previous two
  - Easy integration with SchedFreq and EAS
    - By working on top of already used signals
    - Thus providing a different “view” on the SEs/RQs utilization signals

Signal Proportional Compensation (SPC)

- The boost value is converted into an additional margin
  - Which is computed to compensate for max performance
    - i.e. the boost margin is a function of the current and max utilization

$$margin = boost\ pct * (max\ capacity - cur\ capacity)\ ,\ boost\ pct \in [0,1]$$



Ramp task: 5-60% @5% steps every 3[s] – SPC boost @30%

OPP Selection Biasing Mechanism

- Goal: account for boost margin on OPP selection
- Use RQ's boosted_utilization defined using:
  - Global boost value, when using global boosting
  - MAX boost-group's boost value, when using per-task boosting

Per CPU Boost Groups

RQ's Boost						
	50	60	30	20	80	Boost value
50	1	0	0	1	0	# Runnable Tasks
20	0	0	0	1	0	# Runnable Tasks T1
80	0	0	0	2	1	# Runnable Tasks T2
80	0	1	0	1	1	# Runnable Tasks T3

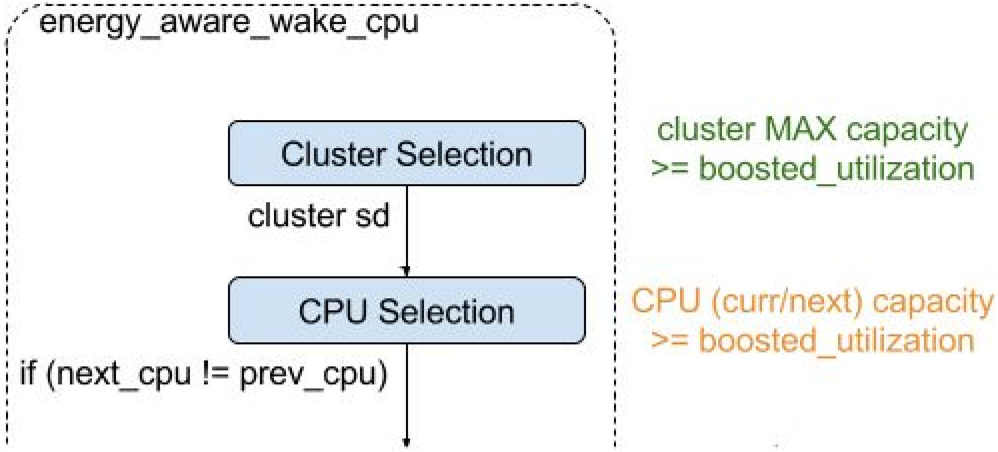
For OPP Selection:

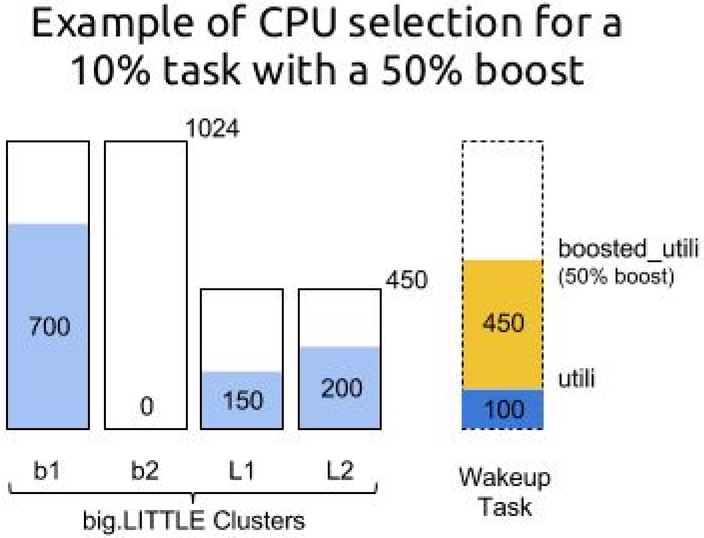
RQ's boost updated at each {enqueue/dequeue}_task_fair

update_capactity_of() uses **boosted_cpu_util()** instead of cpu_util()

CPU Selection Biasing Mechanism (1/3)

- Energy-Aware Wakeup Path
  Goal: find a CPU which can host the boosted utilization
  - using the boosted_utilization signal on some EA wakeup checks

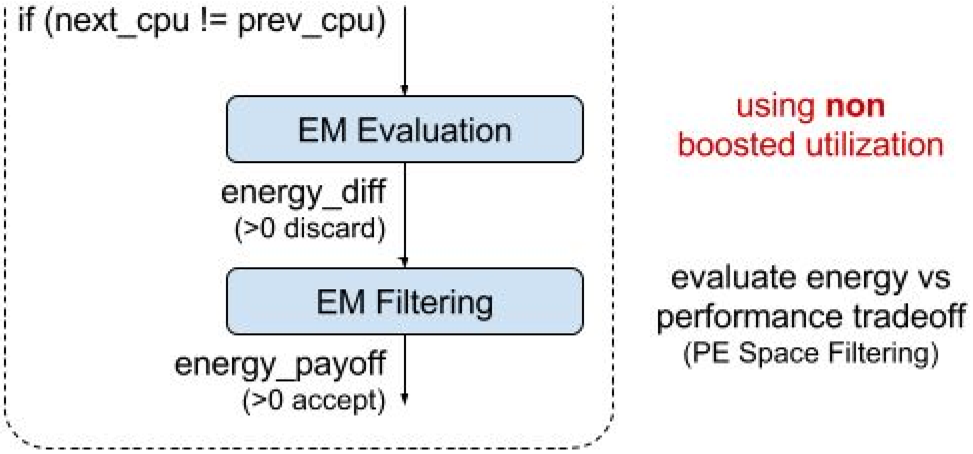




CPU Selection Biasing Mechanism (2/3)

- Evaluation of the Energy-Performance trade-off
  Goal: evaluate if the **increased energy consumption** is compensated by a "reasonable" **performance gain**

- Running small tasks on higher capacity CPUs requires more power
- Performance boost is computed by the EM evaluation step

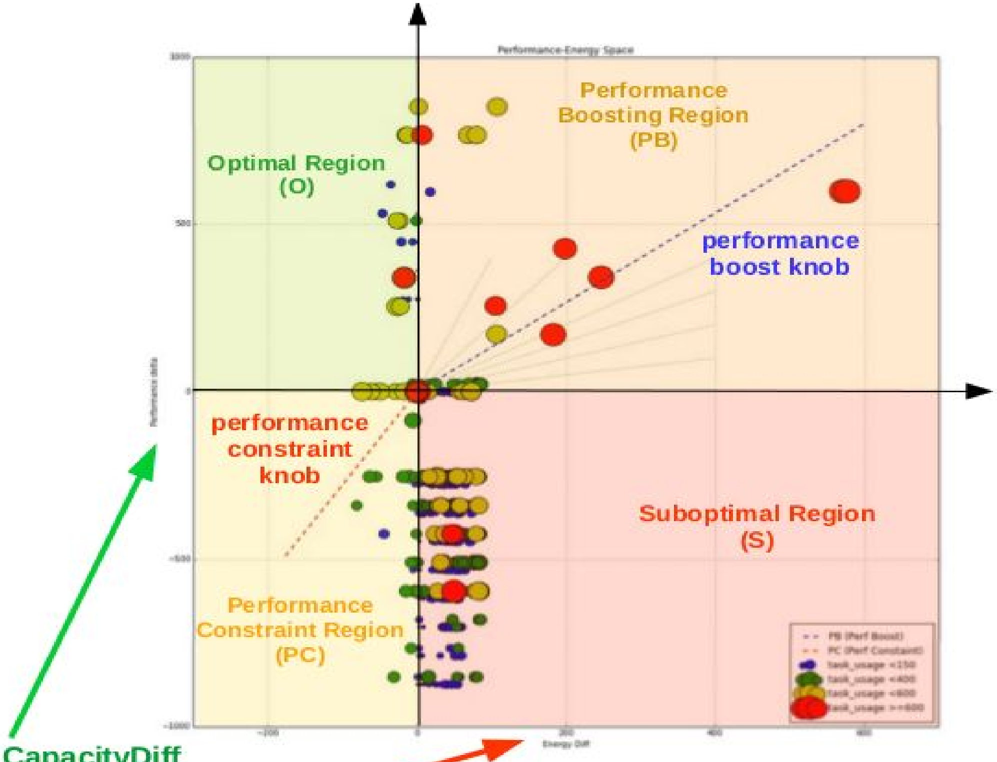


How much power are we willing to spend to get a certain speedup on time-to-completion?

CPU Selection Biasing Mechanism (3/3)

- PE Space Filtering

- 4 Performance-Energy Space Regions
- 2 'cuts', mapped to the same boost knob value
- “Standard” EAS behaviour for boost=0
    - I.e. vertical cut



Normalized CapacityDiff
capacity_after - capacity_before

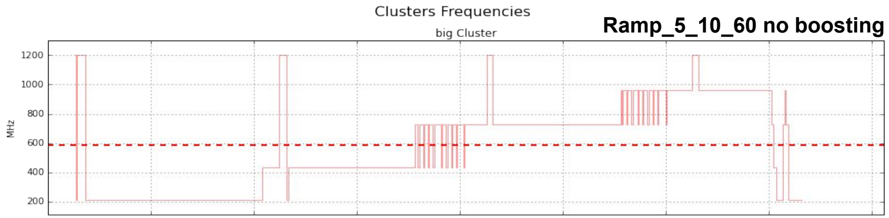
Normalized EnergyDiff
energy_diff / max_system_energy

SchedTune OPP Boosting

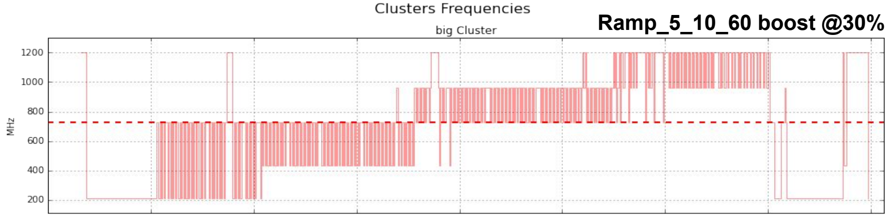
RTApp Generated RAMP Task

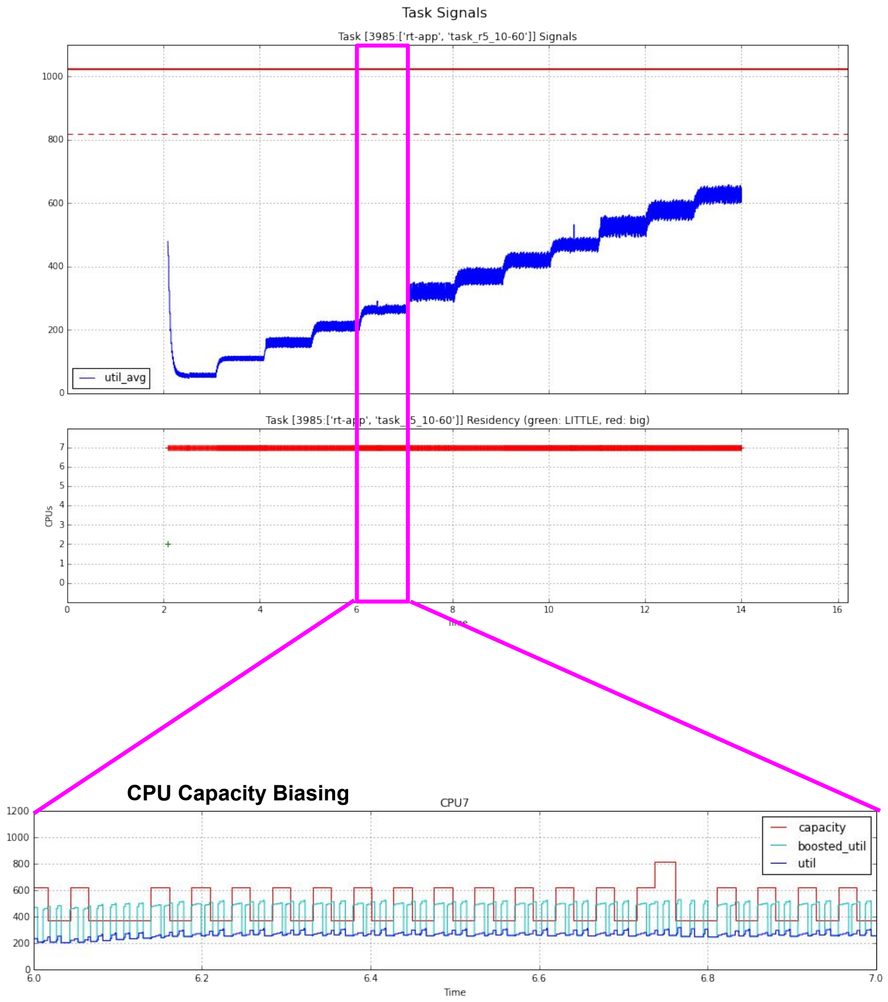
```
"r5_10-60" : {
          "kind"   : "Ramp",
          "params" : {
                    "period_ms" : 16,
                    "start_pct" :  5,
                    "end_pct"   : 60,
                    "delta_pct" :  5,
                    "time_s"    :  1,
                    "cpus"      : [7],
          },
          "tasks" : 1,
},
```

Ramp_5_10_60 no boosting



Ramp_5_10_60 boost @30%

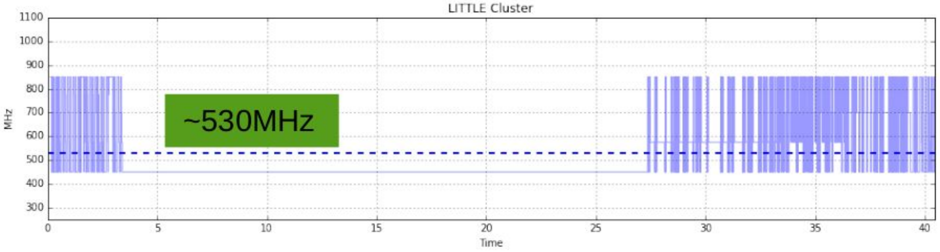




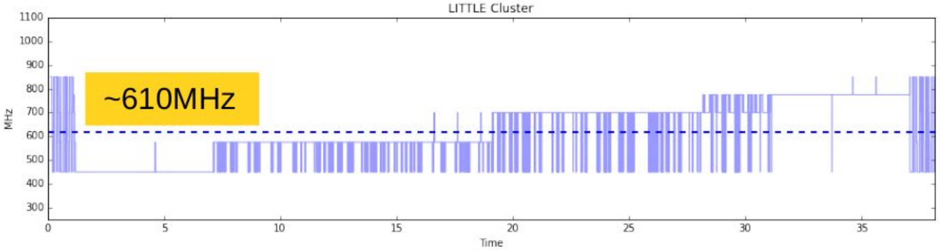
CPU Frequency Selection

- The higher the boost value the higher the avg frequency in this example the task is pinned to run on LITTLE

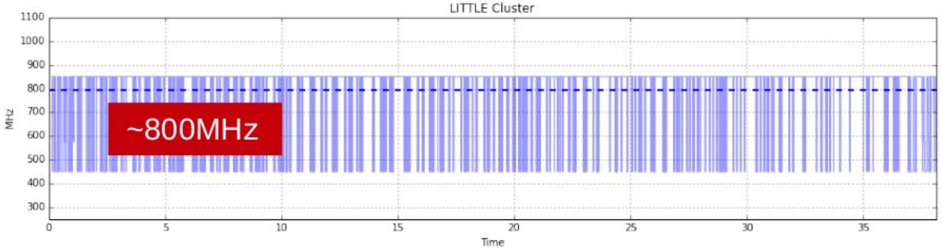
Ramp task: 5-60% @5% steps every 3[s]



No boosting



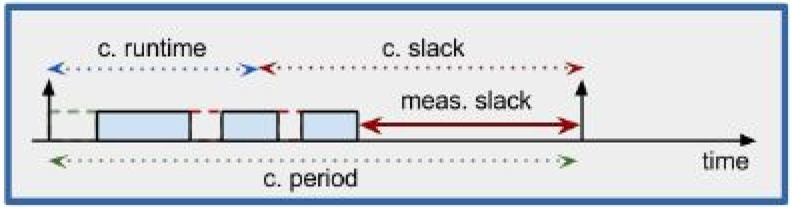
SPC 30% boost



SPC 45% boost

Performance Evaluation (1/2)

- RT-App extended to report slack time related metrics



$$MaxSlack = Period_{conf} - RunTime_{conf}$$

$$PerfIndex = \frac{Period_{conf} - RunTime_{meas}}{MaxSlack}$$

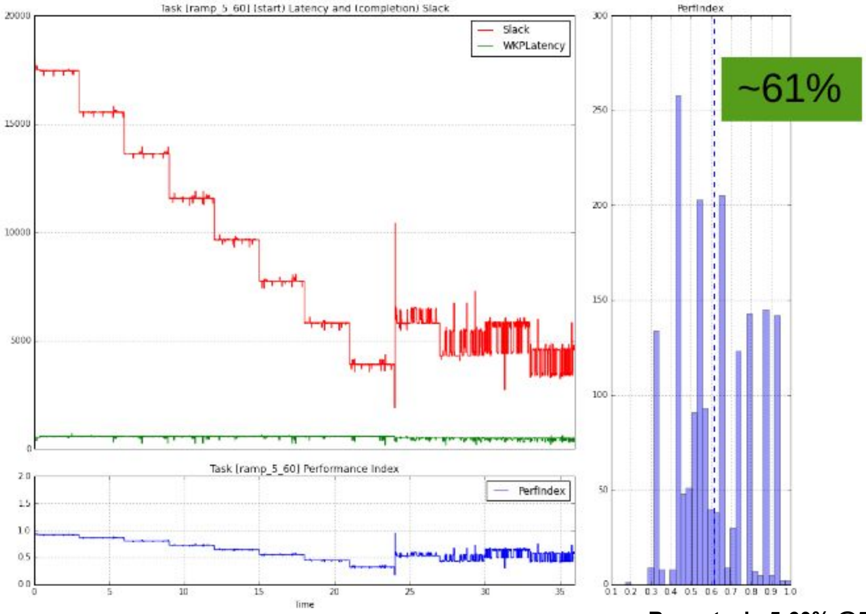
$$NegSlack_{percent} = \frac{\sum Max(0, RunTime_{meas} - Period_{conf})}{\sum RunTime_{meas}}$$

  - too pessimistic on single period missing
    - keep adding negative slack even if the following activations complete in time
    - can be solved by resetting the metrics at each new activation
- Linaro proposed a “dropped-frames” counter
  - we should integrate that as well

Performance Evaluation (2/2)

- Slack Time Distribution

No boosting



SPC 45% boost

Ramp task: 5-60% @5% steps every 3[s]

SchedTune Performance Index

- Based on the composition of two metrics

```
Perf_idx = SpeedUp_idx - Delay_idx
```

- SpeedUp_Index: how much faster can the task run?

```
SpeedUp_idx = SUI = cpu_boosted_capacity - task_util
```

- Delay_Index: how much slowed-down can the task be?

```
Delay_idx = DLI = 1024 * cpu_util / (task_util + cpu_util)
```