Scheduler behavioural testing

Valentin Schneider `<valentin.schneider@arm.com>`

20/05/2019

Outline

Context

Testing setup

Test samples

Dealing with the noise

Wrapping up

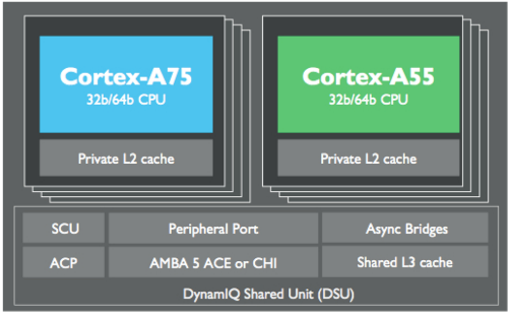
Context

What this is about

- Unit tests mentioned by Oracle folks @ OSPM 2018
  - How do we make sure we don't break other people's stuff?
- This is an overview of how we do things at Arm
  - Powered by LISA
  - Mostly about how we make sure what we care about doesn't get broken
  - Not the one true gospel but at least it's something…
- Two test triggers
  - tip/sched/core + in-flight patches (integration branch) - triggered every 2 weeks
  - Patch validation - can be triggered anytime

Arm big.LITTLE and DynamIQ

- Asymmetric topologies
  - ”clusters” of CPUs w/ different µarchs
  - Frequencies usually shared within a cluster
- Funny requirements on task placement
  - optimal energy/inst (!= minimal power) (EAS)
  - don’t leave cpu-hogs on low-perf CPUs (misfit tasks)
- Backed by some more infrastructure
  - capacity & frequency invariant load-tracking signals (PELT)
  - frequency selection based on scheduler signals (cpufreq)
- All of which we have tests for



Testing setup

rt-app

A highly configurable real-time workload simulator that accepts a JSON grammar for describing task execution and periodicity

– LWN

- Based on a calibration value (amount of work), not pure timer

```
"10pct_task": {
    "delay": 0, # Start running immediatly
    "phases": {
        "p000001": {
            "loop": 62,  # Repeat this phase 62 times (~1s)
            "run": 1600,        # Run for 1.6ms
            "timer": {          # Wait for next timer...
                "period": 16000, # ...Firing every 16ms
                "ref": "10pct_task"
            }
        }
    },
    "policy": "SCHED_OTHER",
    "loop": 1
}
```

Testing dogma

- We’re trying to test specific bits of the scheduler
  - ”Wrong” scheduling decisions can be due to very varied reasons
  - Things like hackbench are way too aggressive
- Using rt-app, we try to reproduce specific scenarios to trigger specific behaviours

- (Most) workloads are parametric on the topology
- Split data collection and analysis
  - Allows ”offline” replaying
  - Lets us test e.g. different margins

Test samples

EAS behaviour

- Goal: Ensure EAS is making the right decisions
  - small tasks on LITTLEs
  - big tasks on bigs
- Example workload: N tasks (with N big CPUs) made of 2 phases
  - low utilization (should be placed on a LITTLE)
  - high utilization (should be placed on a big)
- With an energy model (EM), we can estimate energy costs
  - EM + rt-app description -> estimate cost of energy-optimal task placement
  - EM + scheduling traces (switch/wakeup) -> estimate cost of actual task placement
- rt-app also gives us some latency report for performance analysis

EAS behaviour - energy cost (HiKey960)

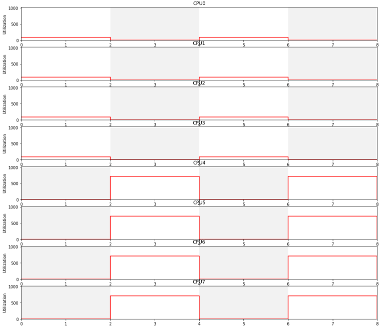


Figure: Expected placement

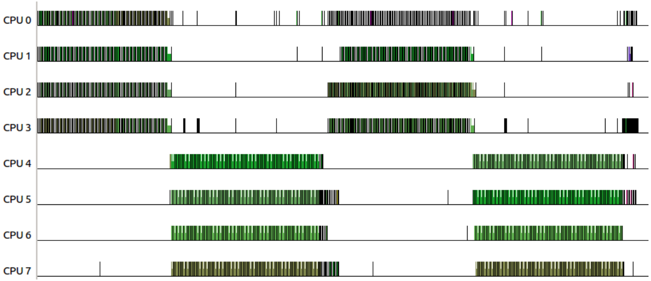
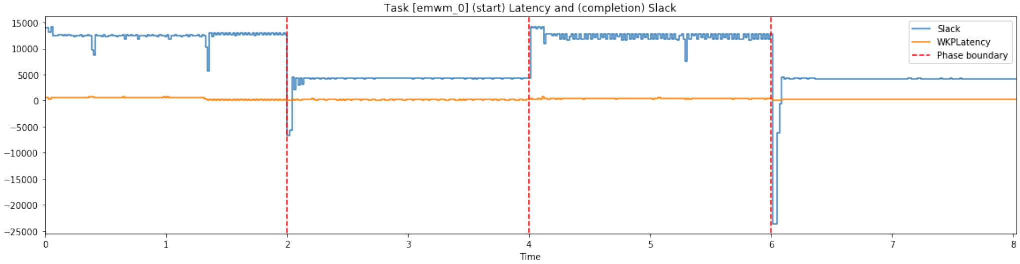


Figure: Actual placement

- Estimated 11’924 bogo-joules < 12’697 threshold (5% margin on optimal placement) (OK)
- Noisiest non-test task was irq/63-tsensor_ (≈0.3% of test duration))

EAS behaviour - performance (HiKey960)

- rt-app gives us a latency report after executing a profile
  - Time between wakeup and execution (wakeup latency)
  - Time from work completion to start of next period (slack)



- Less than 1% of negative slack for all tasks (OK)

EAS behaviour - outcome

- Failures in workload with big & small tasks (≈80% fails)
  - overutilized scenario (no EAS)
  - Very small task co-scheduled with big task on big CPU while LITTLEs are idling
  - WIP: let small tasks through the slow wakeup path?

    (or yet another argument to get rid of DIE level on big.LITTLE)

- Failures in workload with only small tasks(≈5% fails)
  - Small task starts on a big CPU
  - Utilization eventually decays enough that it gets moved to a LITTLE
  - Signal not properly decayed on migration
  - Task ping-ponging between two CPUs of different capacities
  - WIP: `update_rq_clock()` in `migrate_task_rq_fair()`?

DVFS sanity checking

- Goal: make sure cpufreq/DVFS can be relied upon
- Run sysbench on the same CPU at increasing frequencies
- Ensure amount of work done is strictly monotonically increasing
- Highlighted some frequency switching issue on HiKey960, see patch.

CPU	OPP	Base	Fix
0	533000	104	104
0	999000	104	201
0	140200	285	285
0	1709000	285	349
0	1844000	377	377
4	903000	249	248
4	1421000	249	394
4	1805000	500	500
4	2112000	499	583
4	2362000	653	654

Load tracking

- Goal: make sure load tracking signals behave as expected
  - Involves capacity & frequency invariance
- No escape here, need extra trace events: runqueue and entity signals
- Invariance
  - Run the same task on a LITTLE, a big, and with different frequencies
  - Signal values should be about the same
- Signal dynamics
  - Run a task pinned to a given CPU
  - Simulate PELT signal
  - Compare min/max values
  - Compare values at each reported event

Load tracking

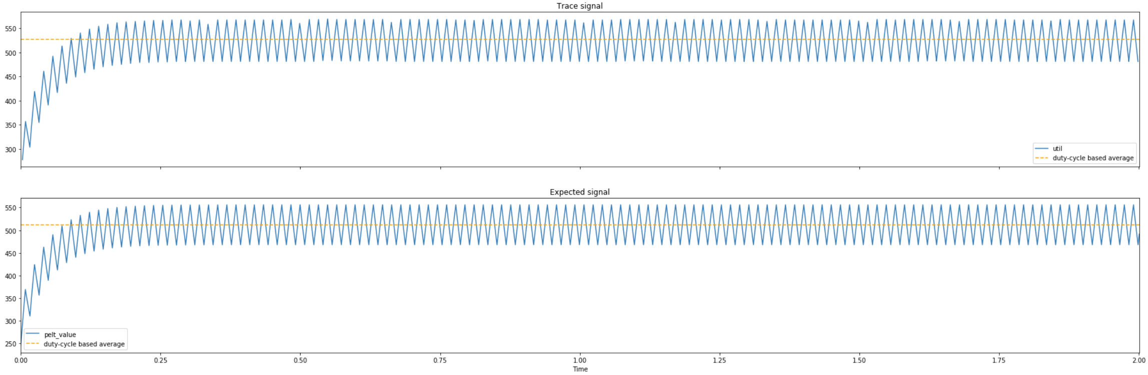


Figure: PELT utilization behaviour test (50% task)

Load tracking - outcome

- NOHZ remote stats update (LKML)
  - Created some tests to validate the patch-set
  - Written by a complete newbie
  - Found a simple condition reordering mishap in a later version

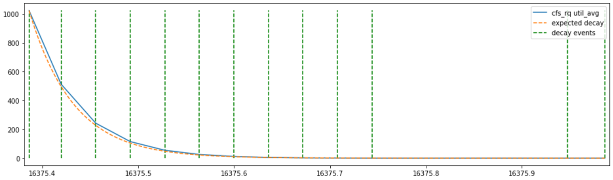


Figure: Blocked load decay by NOHZ balance

- PELT time scaling (kernel.org)
  - Load used to only scale with freq, not capacity (`e3279a2e6d69 ("sched/fair: Make utilization tracking CPU scale-invariant")`)
  - Task on a LITTLE generated ~twice the load than if it ran on a big (HiKey960)
  - Tests started failing
  - Not a bug per se, but useful eye-opener

Dealing with the noise

Limiting what can be executed

- How to prevent non-workload tasks from running?
  - Background activity can impact the scheduling

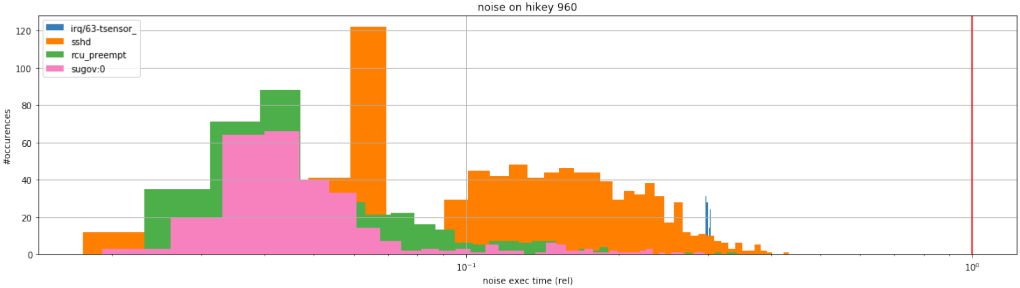
- Buildroot
  - Userspace with the bare minimum
- Freezer cgroup
  - Less ideal than buildroot, but helpful for e.g. Android targets
  - Small exclusion list (`init`, `systemd`, `ssh`, `adbd`, …)

- Improves the situation, but does not cover everything…
  - `sshd`, `adbd`, `NFS`, `USB`…

Noisy tasks we have to live with

- Goodie from rt-app: we know the exact name of our tasks!
  - we can run some stats on the scheduling trace and look at how busy non-rt-app tasks were
  - raise a flag when that's too much (undecided test result)
  - ATM threshold is 1% of total rt-app duration (configurable per-test)

- Sweep of the culprits on all of our tests (HiKey960):



Wrapping up

Recap & todo

- We can do quite a lot with just `sched_switch` & `sched_wakeup`
- But we need more for validating something as fundamental as PELT
  - Trying to guess the signal from the scheduling trace is a no-go
  - See patch-set from Qais

- More varied synthetic workloads
  - Other scheduler bits to look at?
  - Suggestions (and contributions!) more than welcome

Thank you!

Misfit

- Goal: make sure misfit migration works as expected
  - mostly timing aspects
- Example workload: N CPU-hogs (with N CPUs)
  - Tasks on bigs will finish first and then pull misfits from LITTLEs
- Look at every idle window of a big CPU
  - If any LITTLE is busy, assert the idle window duration is < threshold
  - If misfit is doing its job, big CPUs should always have something to do
- Also make sure there’s no coscheduling going on

Misfit - task placement

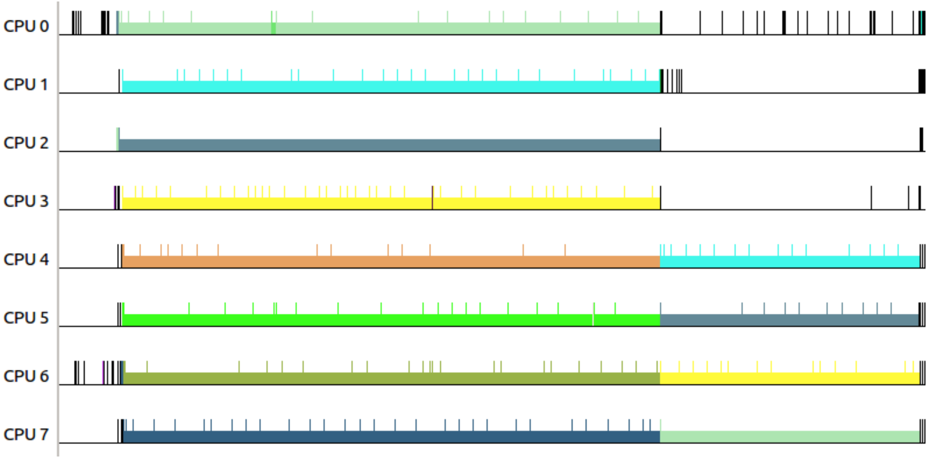


Figure: Misfit test trace

Misfit - outcome

- Somewhat indirectly lead to `3f130a37c442 ("sched/fair: Don't increase sd->balance_interval on newidle balance")`

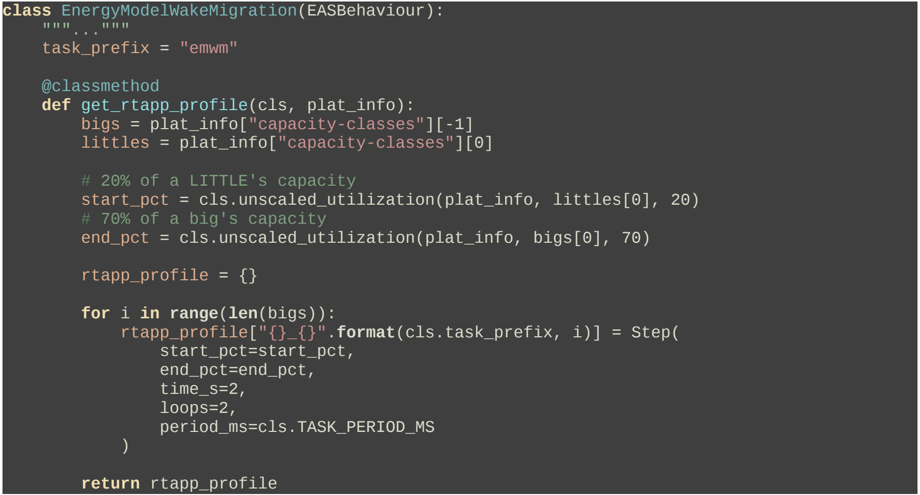
RTA+Ftrace in python

14/05/2019

```
target = ...

rtapp_profile = {
    "10pct_task" : Periodic(duty_cycle_pct=10, duration_s=1, period_ms=16)
}

wload = RTA.by_profile(target, "example", rtapp_profile)

ftrace_coll = FtraceCollector(target, events=["sched_switch", "sched_wakeup"], buffer_size=10240)

with ftrace_coll:
    wload.run()

ftrace_coll.get_trace("path/to/trace.dat")
```

Task placement code snippet

14/05/2019



```python
class EnergyModelWakeMigration(EASBehaviour):
    """..."""
    task_prefix = "emwm"

    @classmethod
    def get_rtapp_profile(cls, plat_info):
        bigs = plat_info["capacity-classes"][-1]
        littles = plat_info["capacity-classes"][0]

        # 20% of a LITTLE's capacity
        start_pct = cls.unscaled_utilization(plat_info, littles[0], 20)
        # 70% of a big's capacity
        end_pct = cls.unscaled_utilization(plat_info, bigs[0], 70)

        rtapp_profile = {}

        for i in range(len(bigs)):
            rtapp_profile["{}_{}".format(cls.task_prefix, i)] = Step(
                start_pct=start_pct,
                end_pct=end_pct,
                time_s=2,
                loops=2,
                period_ms=cls.TASK_PERIOD_MS
            )

        return rtapp_profile
```

Noisy tasks decorator

14/05/2019

```
@RTATestBundle.check_noisy_tasks(noise_threshold_pct=1)
def test_throughput(self, allowed_idle_time_s=None) -> ResultBundle:
    ...
```

Test results summary

- We need several iterations to have confidence in our results
- Results for HiKey960
  - There’s an actual scheduling corner case hiding in there (see task placement test outcome)

testcase	status		
EnergyModelWakeMigration:test_slack:	passed	315/315	(100.0%)
EnergyModelWakeMigration:test_task_placement:	passed	315/315	(100.0%)
[...]			
TwoBigThreeSmall:test_slack:	passed	315/315	(100.0%)
TwoBigThreeSmall:test_task_placement:	FAILED	8/315	(2.5%)

Comparing data sets

- We always compare the test results from one integration to the previous one
- Example here for test results on HiKey960

testcase	old%	new%	pvalue
TwoBigThreeSmall:test_task_placement	0.0%	3.7%	1.36 e-02