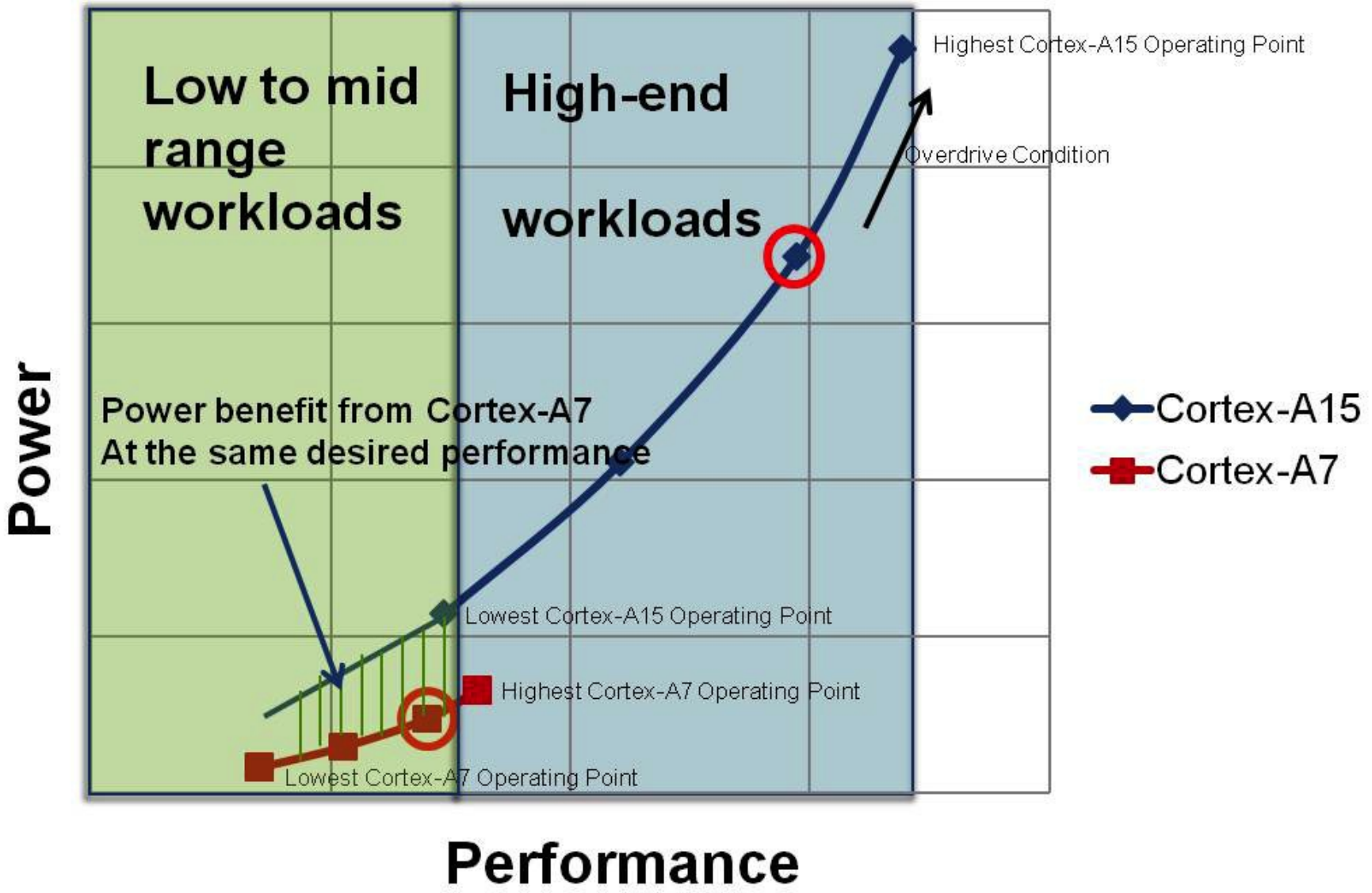
Adventures in (simulated) Asymmetric Processing

Pantelis Antoniou
panto@antoniou-consulting.com
panto@irc.freenode.net / #beagle

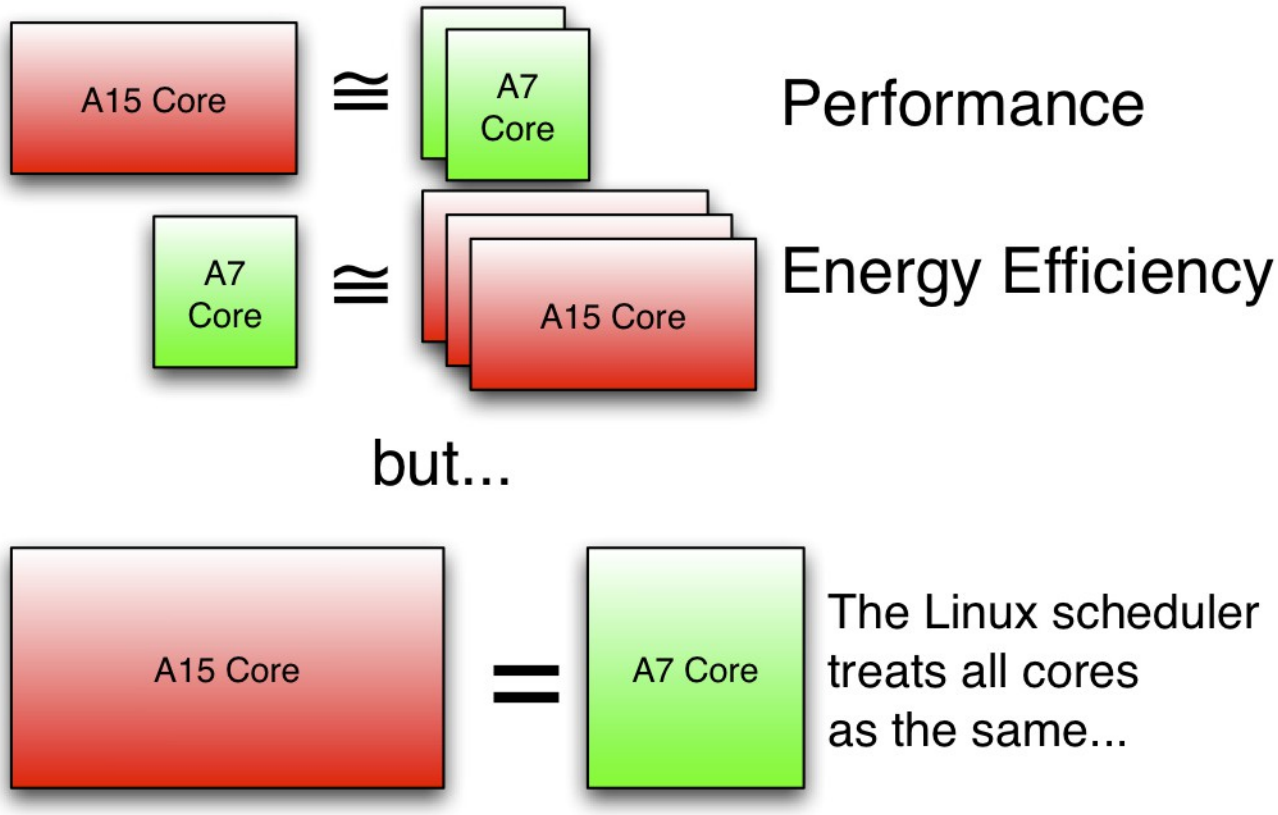
What is Asymmetric Processing, and what the big.LITTLE idea?

- Vanilla Asymmetric processing
  - Performance, power-envelope, micro-architecture differ wildy.
  - Typically the instruction set architecture is different, see CPU/DSP, PS3 CELL, TI's PRU etc.
- big.LITTLE is the combo of a low power Cortex-A7 with a high performance Cortex-A15 which both have the ISA.
  - Single kernel – same userspace.
  - Migration costs are cheap (a few 100s of usecs)
  - Trivial programmer visible differences (L1 I-cache size, PMU differences etc.)

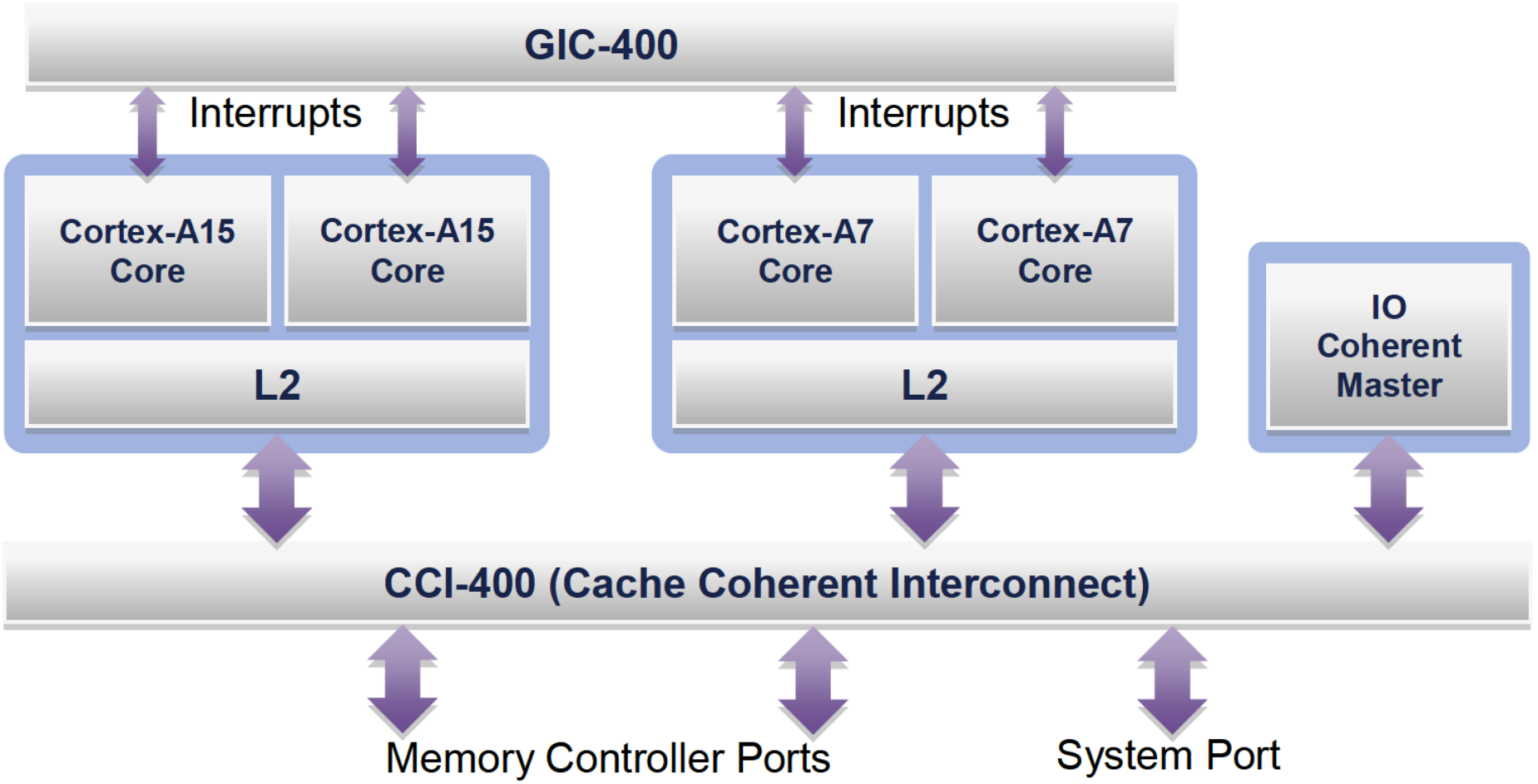
Optimum CPU selection curve



Cortex-A7 vs Cortex-A15 Overview



Complete big.LITTLE system



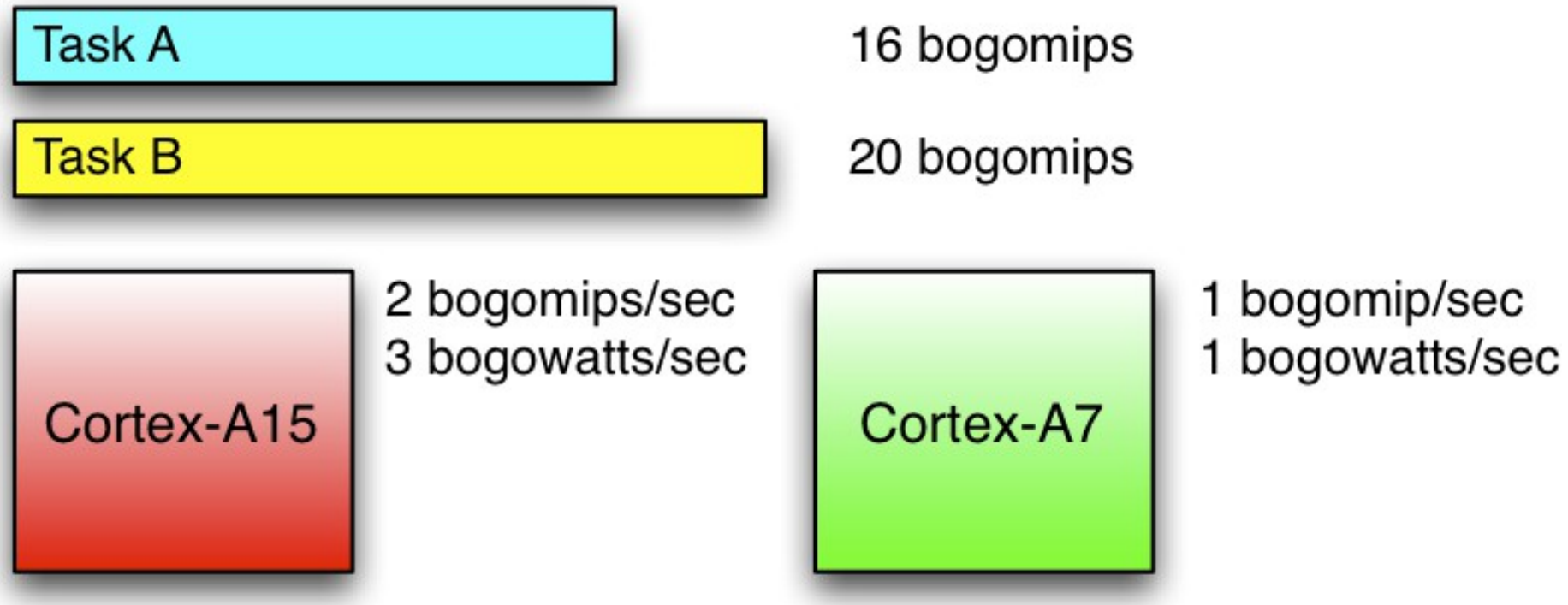
S/W reflects H/W architecture

- Scheduling algorithms need to measure 'work done' in a 'unit of time'.
- Measuring 'work done' is not easy
  - Differences in micro-architecture
  - Cache size effects
  - I/O bandwidth differences
  - PMUs are not standard, and have pretty high overhead.
- When you don't care about power, 'work done' in a unit of time, is directly proportional to the 'unit of time'

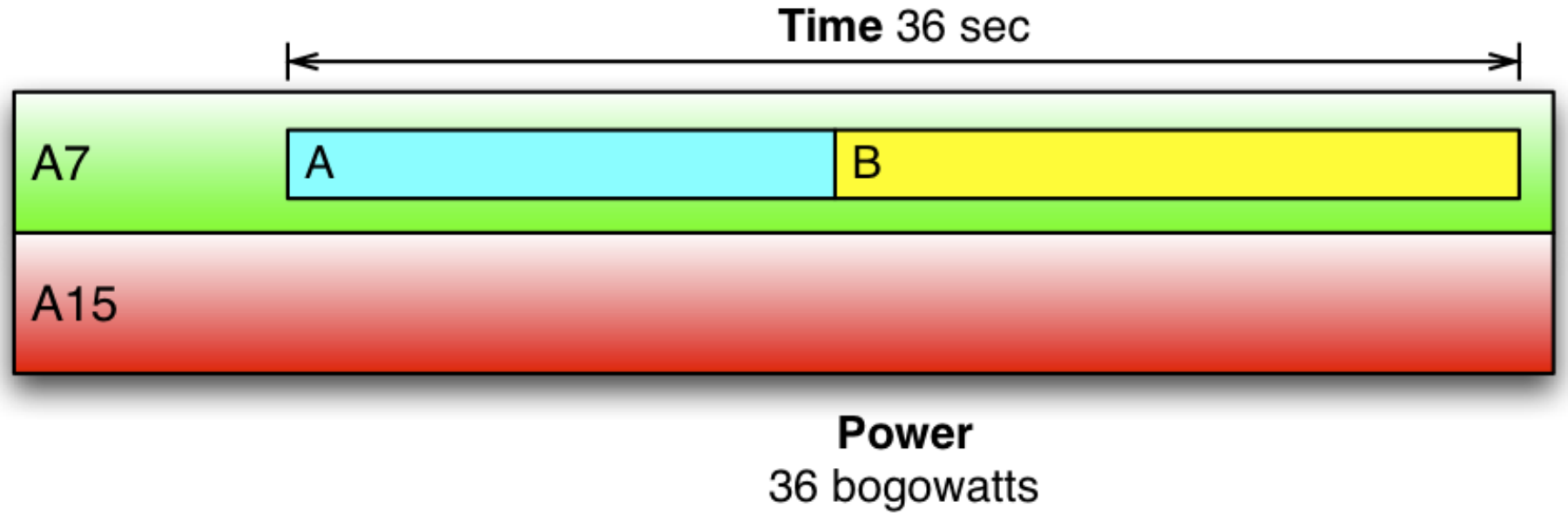
Power Scheduling Bogo-Example

- Simplified system
  - 1 Cortex-A15, 2 bogomips/sec, 3 bogowatts/sec
  - 1 Cortex-A7, 1 bogomip/sec, 1 bogowatt/sec
- Simplified workload
  - Task A, 16 bogomips
  - Task B, 20 bogomips
- A real system is considerably more complex
  - Tasks are memory bound? I/O bound? Cache effects? Micro-architectural differences...

Power Bogo-Example Legend



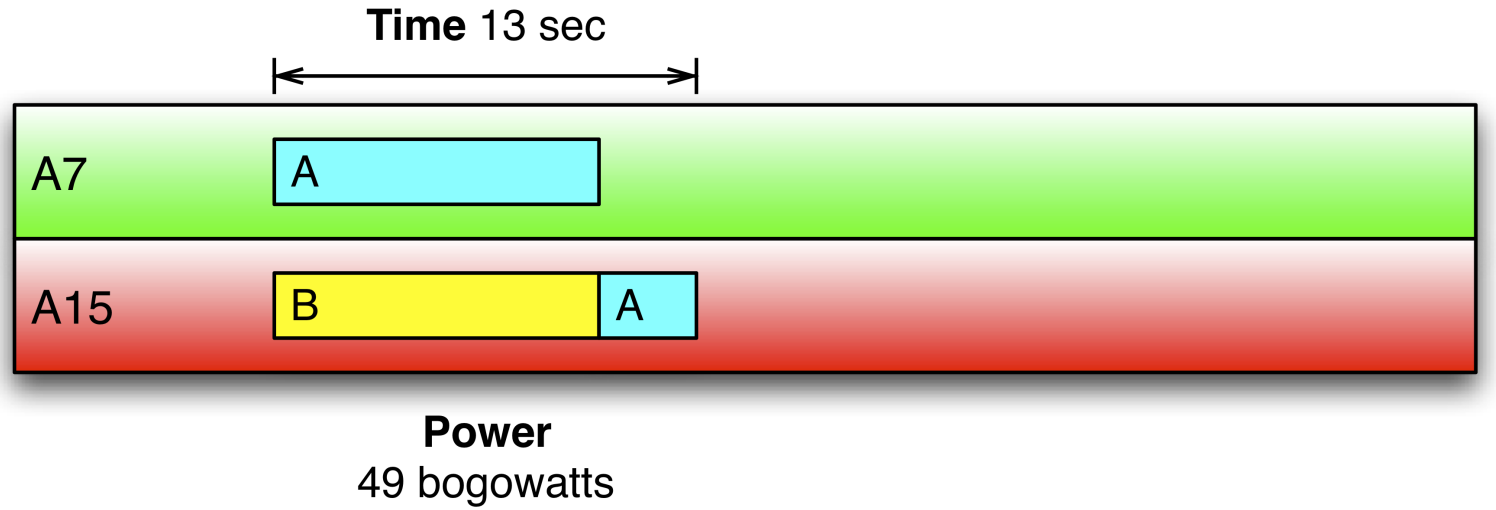
Most power efficient sched policy



Most power efficient (cont)

- Tasks execute sequentially on the most power efficient core, the Cortex-A7.
- Unfortunately it takes the most amount of time.
  - Total power consumed 36 bogowatts
  - Total amount of time 36 secs.
- Cortex-A15 might as well be shut down.
  - Only useful as a last resort (i.e. Battery is dying..)

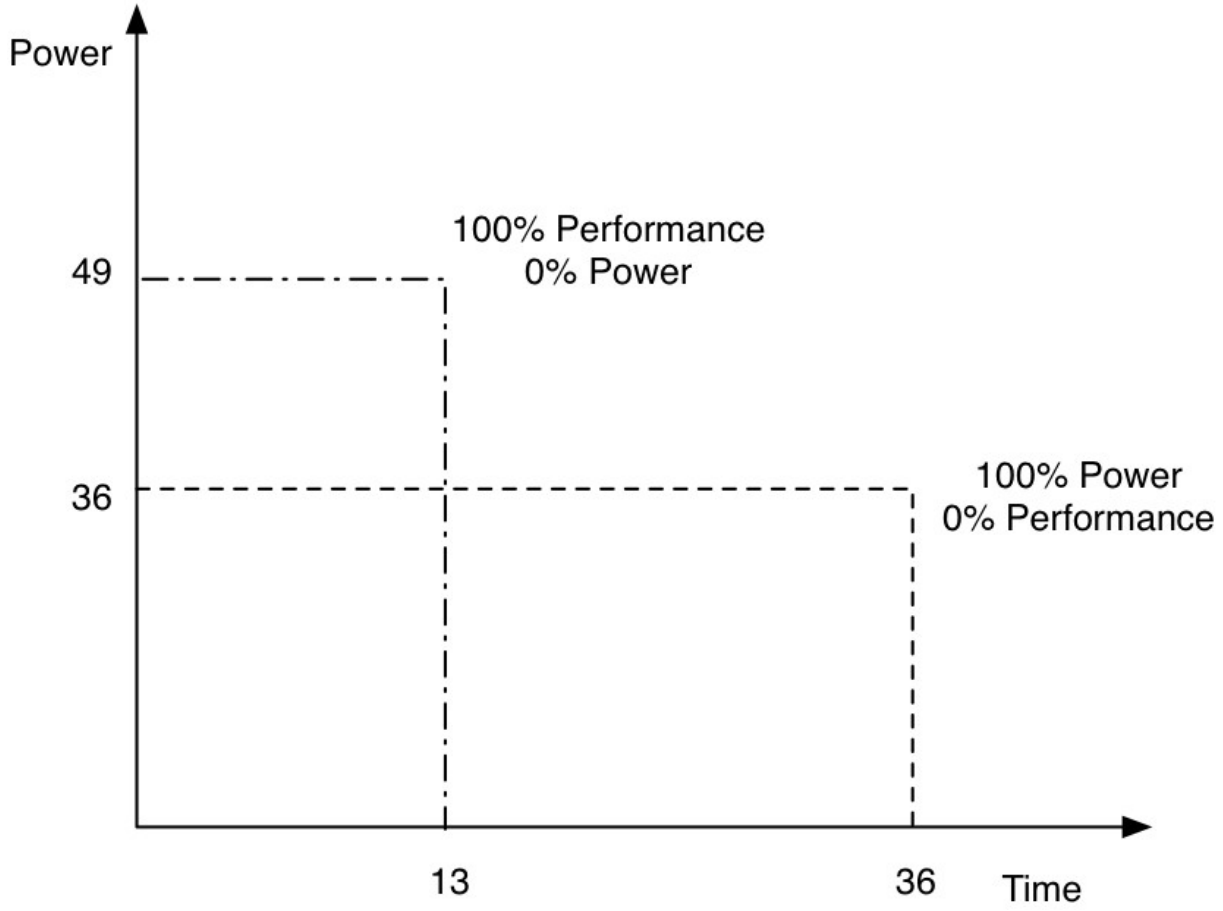
Most performant sched policy



Most performant (cont)

- Tasks use all the computing resources.
- Tasks get allocated on the fastest Cortex-A15 first.
- Tasks not 'fitting' are put to the slower Cortex-A7 if possible.
- It is the fastest policy, but also the most power hungry.
  - Total amount of time 13 secs.
  - Total power consumed 49 bogowatts
- If connected to mains power, makes sense for high performance workloads.
  - You wouldn't want to use it on your phone on battery power

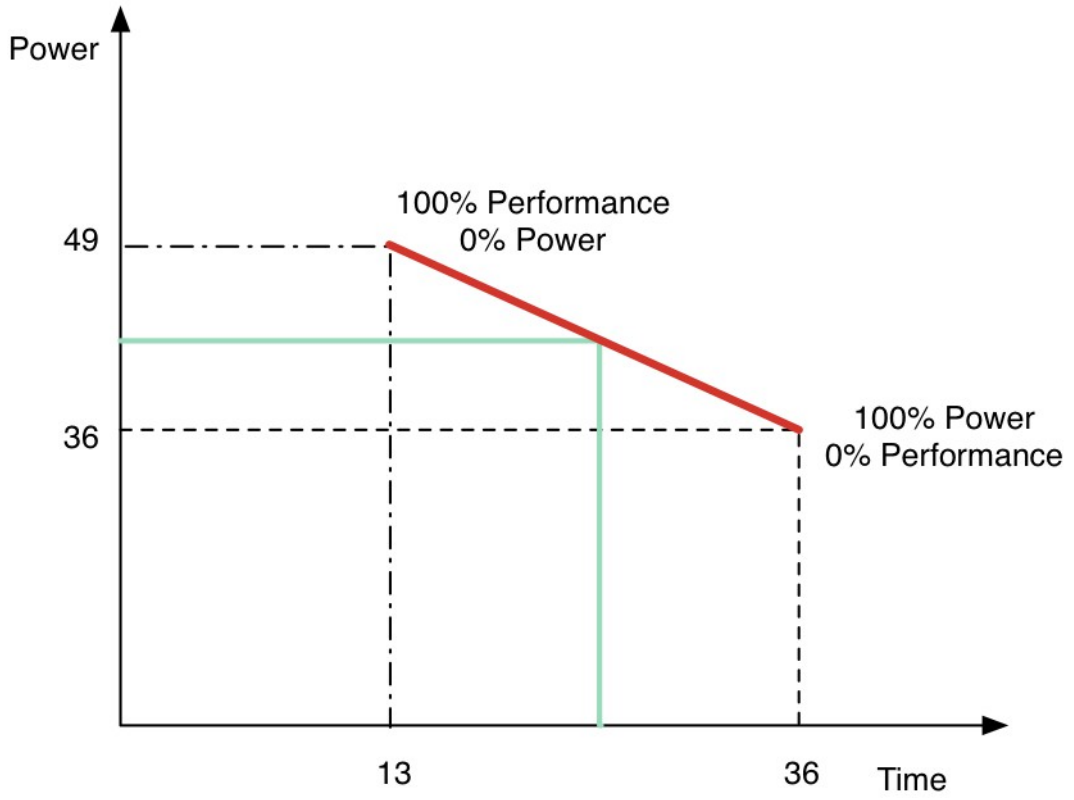
Power Vs Performance (1)



Power Vs Performance (2)

- Two extremes
  - 100% Performance, 0% Power Efficiency
  - 0% Performance, 100% Power Efficiency
- The ideal scheduling policy should lie on the line/curve between those two points.
- System policy dictates where the operation point should be:
  - On battery power
  - On DC power
  - System specific policy
    - Android Home screen – low performance
    - Per application policy, etc.

Power Vs Performance (3)



Power Vs Performance (4)

- No generic way to predict the amount of processing (MIPS of work) a task will do.
- We can only infer from the history of the task.
- Average task load is a reasonably good metric.
- We (currently) have no way to measure power, we can deduce by amount of time a task was executing on a CPU.
- The scheduler does not track MIPS either, so we have to deduce bogoMIPS with a similar manner.

The Linux CFS scheduler and time

- Uses balanced RB trees keyed by 'virtual runtime'
- Fairness is achieved by math calculations.
- No time-slices.
  - No feedback/stimulus for PM decisions.
  - Time == 'work done'
  - Assumption false, even on non asymmetric systems (think CPUFREQ with independently clocked CPUs).
  - Works OK on SMP
  - Simple (comparatively)

Per-entity load average tracking

- Recent development by Paul Turner
- Tracks load by summing per sched entity load instead of per CFS run queue.
- More accurate, less prone to weird artifacts affecting the old algorithm.
- Is the basis for all current work on asymmetric processing.

The Linaro Angle

- The focal point for big.LITTLE MP scheduler development.
- HMP (Hybrid Multiprocessing) scheduler patchset
  - Uses per-entity load tracking to assign tasks to LITTLE or big CPU domains.
  - ARM topology provides CPU-power of each core.
  - Scales load average factor with both CPU-power & CPU-freq
- Perhaps too invasive and big.LITTLE specific.
- WIP – moves fast

Alex Shi's power aware patches

- Generic power framework.
- Not big.LITTLE specific, should offer improvements on any power aware system.
- Relies on per-entity load average patchset too.
- Assumes run-to-idle is beneficial.
- Packs tasks in as few as possible cores/clusters.
- Better chances of landing in mainline.

Miscellaneous power saving ideas

- Paul McKenney's hot-plug cleanup/faster operation patches
  - Significantly speeds up hot-plug path
  - Latencies down from worst case of seconds.
- Cluster aware idle patches
  - Typically power domains are per-cluster.
- Nicolas Pitre's big.LITTLE switcher
  - big.LITTLE is just a different OPP
  - One to one mapping of Cortex A7-Cortex-A15
  - Not as efficient as MP scheduling, but it works.

Sounds great! Where can I get it?

- ARM's Versatile Core Tile Express
  - 2xCortex-A15 + 3xCortex-A7
  - Available but $$$...
- Samsung's Exynos 5 Octa
  - 4xCortex-A15 + 4xCortex-A7
  - Available RSN
- TI's OMAP6
  - Secret!
  - Available never. RIP.

You probably have to simulate

- ARM's FastModel emulator
  - Can be used to verify the h/w design
  - Can be used to verify the s/w design
  - Horribly slow for anything else
- Pandaboard ES (OMAP4460)
  - 2xCortex-A9
  - Not really big.LITTLE but when there's a will...
  - Mainline kernel / Android
  - Cheap!

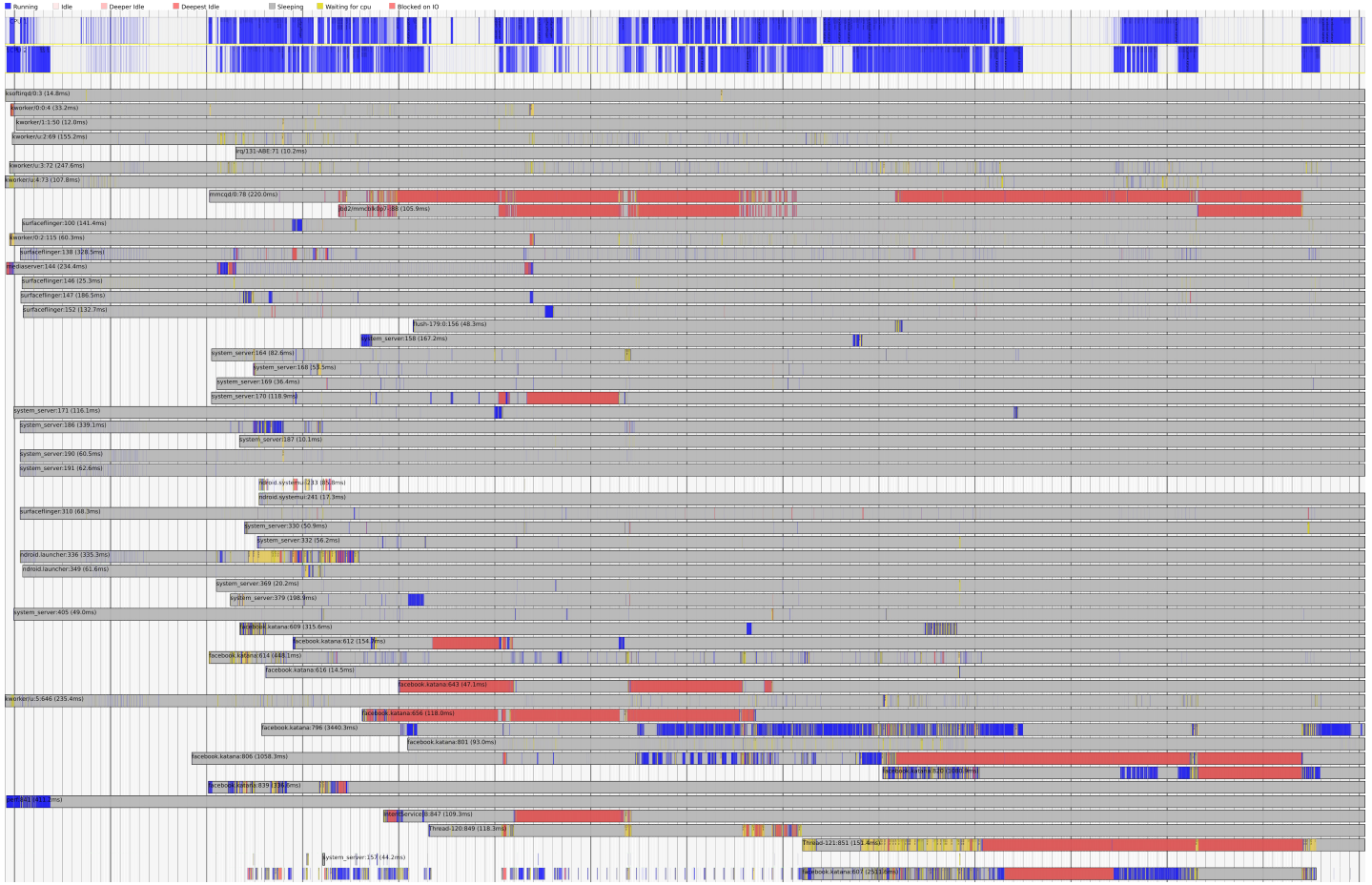
How do I check how and what is running on which CPU?

- Use perf – part of the Linux kernel
- Perf can do more than poll hardware performance counters
- Not portable between arches nor different kernel versions.
- Perf timechart is nice – if you like multi megabyte SVG graphic files.

Perf capture & time chart generation

```
$ perf sched record -e sched:* \
-e:power*
^Ctrl-C
$ perf timechart
$ ls -sh output.svg
10M output.svg
```

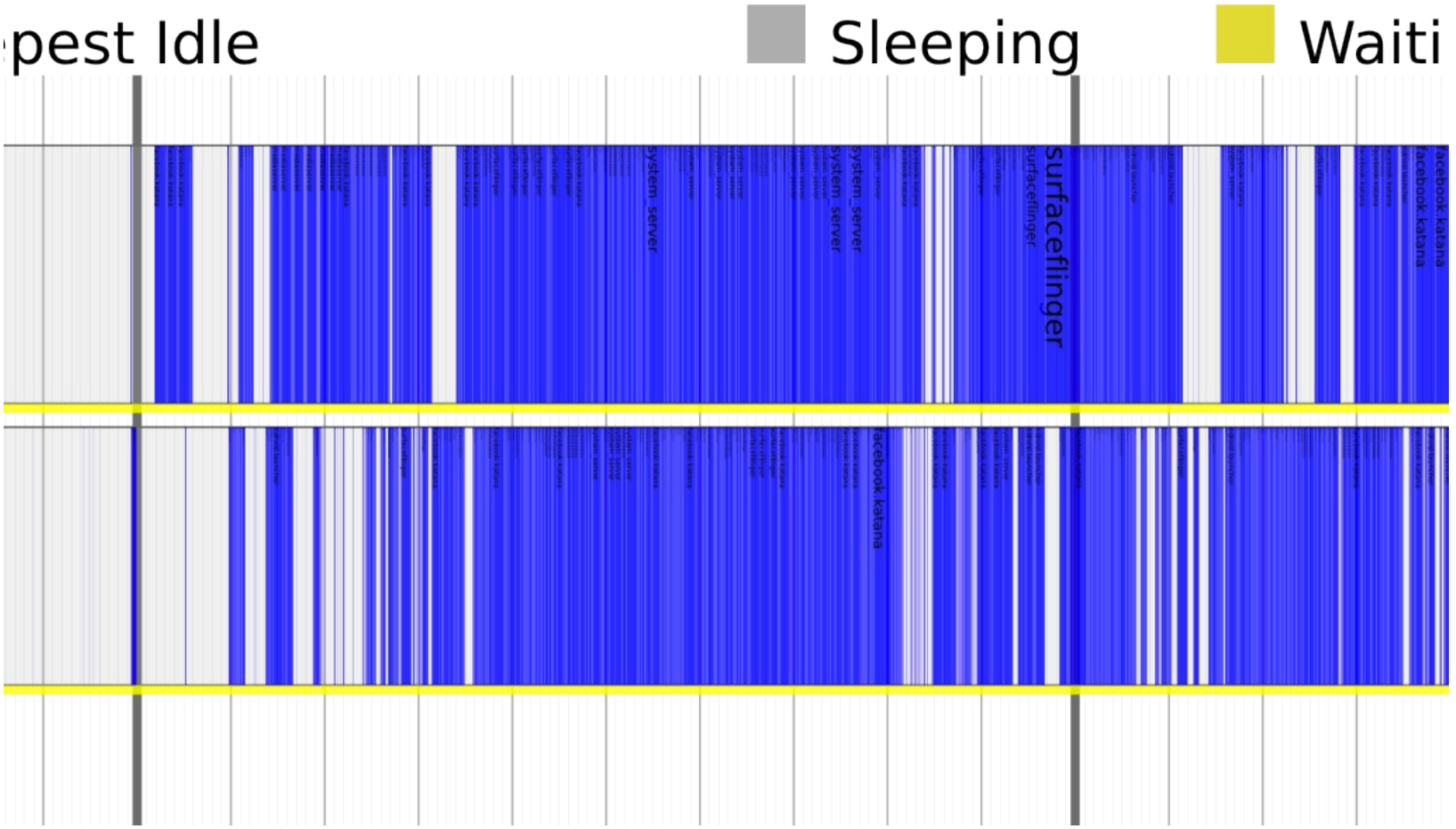
Perf timechart



Perf timechart fallout (1)

- The SVG files generated are **HUGE**
- Inkscape works (albeit slowly) well enough to export a PNG
- It seems it's a mess, you can figure out what's going on.
- You can roughly see per CPU utilization and the tasks running (and waiting).
- You can even see power events (C-states)

Perf timechart zoom



Perf timechart fallout (2)

- The most interesting statistic from a scheduler evaluation point is the utilization of each of the CPUs. It is impossible to discern from the chart.
- If you've looked closely you'd see that the capture is from Android.
- <FLAME>Android is the de-facto consumer level Linux API</FLAME>

Perf analyze

- Simple perf command to summarize per CPU utilization

```
$ perf sched analyze
CPU utilization chart
CPU |        Duration |                        Busy |
--- |        -------- |                        ---- |
  0 |    14160217285 |          7934204109 %56.0 |
  1 |    14160217285 |          7037750235 %49.7 |
```

Scheduler workload capture

- Kernel scheduler hacking must take place on the latest mainline kernel.
- Interesting user workloads will come from wildly different kernels, distributions or even Android versions.
- Need to have a portable method to capture a workload's behavior and replay it in the kernel scheduler development box.

Perf spr-replay (1)

- SPR = Scheduler Playback Replay
- Addition to perf
- Captures scheduler workload and converts to a list of simplified scheduler instructions
- Instructions are simple and user modifiable
- Playback of scheduler instructions on a different system is possible
- Capture on Android, playback on a normal Linux distribution (Angstrom) using a mainline Linux kernel.

Perf spr-replay (2)

- Each run is a list of parallel executing process with each process having it's own instruction list:

```
[NAME/PID]
    [INSN]
    [INSN]...
    END
```

- Portable and easy to understand
- Different that the sched replay already present in perf

Perf spr-replay Instructions

- burn <nsecs>
  - Loop burning CPU power for <nsecs>
- sleep <nsecs>
  - Sleep for <nsecs>
- spawn <nsecs>
  - Spawn a task after <nsecs>
- clone-parent <child-pid>
  - Parent side of fork for <child-pid>
- clone-child <parent-pid>
  - Child side of fork for <parent-pid>
- wait-id <id>
  - Wait until the event with <id> is signaled by another task.
- signal-id <id>
  - Signal the event with <id>

Perf spr-replay usage

```
$ perf sched record
# run workload (i.e. cpucyle cpu0)
# Ctrl-C perf
$ perf timechart
$ perf sched spr-replay -n -l
[perf/1571] R:0
[kworker/1:1/38] R:640868
[init/1] R:183105
[kworker/0:1/39] R:1159667
[sh/1546] R:4211426
[cpucycle/1572] R:2007263185
[migration/0/8] R:61035
[flush-0:11/1386] R:30517
[dropbear/1550] R:274659
```

Perf spr-replay usage (cont)

```
$ perf sched spr-replay -n -s1572 -g >spr.txt
$ cat spr.txt
[cpucycle/1572]
        burn 2007263185
        exit 0
        end
$ time perf sched spr-replay -f spr.txt
#1 tasks, #0 futexes, total work area size 12288
0.00user 2.02system 0:02.18elapsed 92%CPU
```

big.LITTLE simulation

- Real big.LITTLE H/W is generally unavailable (although that will change shortly)
- Needed a way to simulate in acceptable speeds.
- Collect data about scheduler changes for an Android game / graphics heavy workloads.
- Late Intel & AMD CPUs have per CPU CPUFREQ control.
- Most ARM (well TI) SoCs have per CPU cluster frequency controls.

Virtual CPUFREQ (1)

- Generating interrupts on a given processor has the apparent effect of slowing that processor down.
- VCPUFREQ is a method of generating carefully controlled interrupt load so the result simulates a real CPU cluster with individually controlled CPU frequencies.
- OMAP specific backend using DMTIMERs is more accurate.
- Generic backend uses hrtimers (somewhat less accurate).

Virtual CPUFREQ (2)

- CONFIG_VCPUFREQ
  - Enable Virtual CPUFREQ governor
- CONFIG_VCPUFREQ_OMAP2PLUS
  - OMAP governor backend – accurate/not portable
- CONFIG_VCPUFREQ_HRTIMER
    - Generic HRTIMER backend – portable/not accurate

Virtual CPUFREQ (3)

```
# cpufreq-info
analyzing CPU 0:
  driver: vcpufreq
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 500 ns.
  hardware limits: 350 MHz - 920 MHz
  available frequency steps: 350 MHz, 700 MHz, 920 MHz
  available cpufreq governors: conservative, ondemand, powersave,
userspace, performance
  current policy: frequency should be within 350 MHz and 920 MHz.
                  The governor "userspace" may decide which speed to use
                  within this range.
  current CPU frequency is 700 MHz (asserted by call to hardware).
  cpufreq stats: 350 MHz:0.00%, 700 MHz:100.00%, 920 MHz:0.00%
analyzing CPU 1:
SAME as CPU 0
```

Virtual CPUFREQ (4)

```
# cpufreq-set -f 920000
# cpucycle cpu0
87089263
# cpucycle cpu1
87134518
# cpufreq-set -c 0 -f 700000
# cpucycle cpu0
65753329
```

- Pretty accurate emulation
- 700/920 =~ 0.76 - 65753329 / 87134518 =~ 0.755

Conclusion

- Asymmetric scheduling is very much a WIP
- Slow progress is being made
- Competing ideas about how to do power-aware scheduling
- Still a long way to go for mainline acceptance
- Hope you like working on hard problems!

Questions?