

Research Update on big.LITTLE MP Scheduling



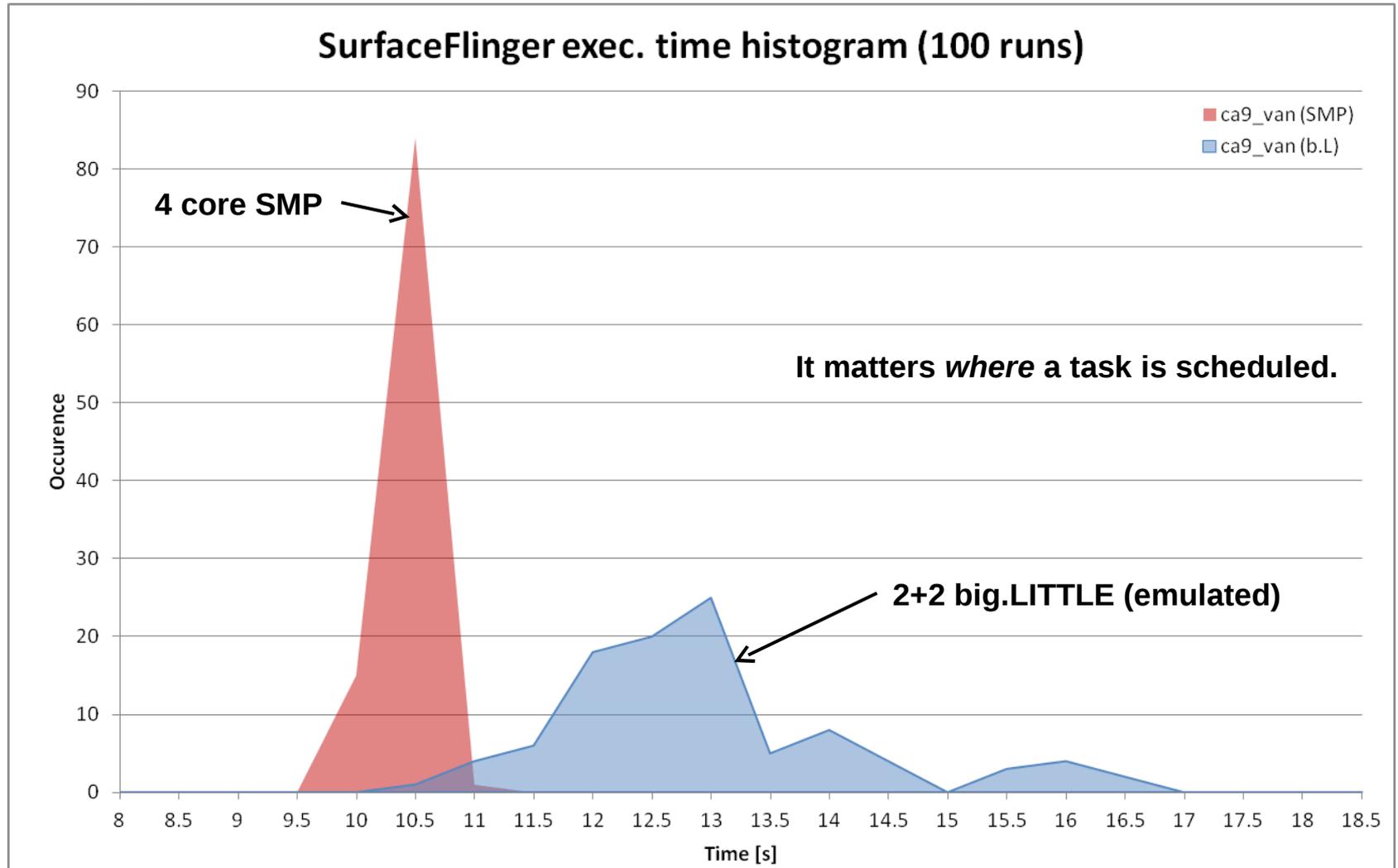
Morten Rasmussen
Technology Researcher

Why is big.LITTLE different from SMP?

- SMP:
 - Scheduling goal is to *distribute work evenly* across all available CPUs to get maximum performance.
 - If we have DVFS support we can even save power this way too.
- big.LITTLE:
 - Scheduling goal is to maximize power efficiency with only a modest performance sacrifice.
 - Task should be distributed *unevenly*. Only critical tasks should execute on big CPUs to minimize power consumption.
 - Contrary to SMP, it matters *where* a task is scheduled.

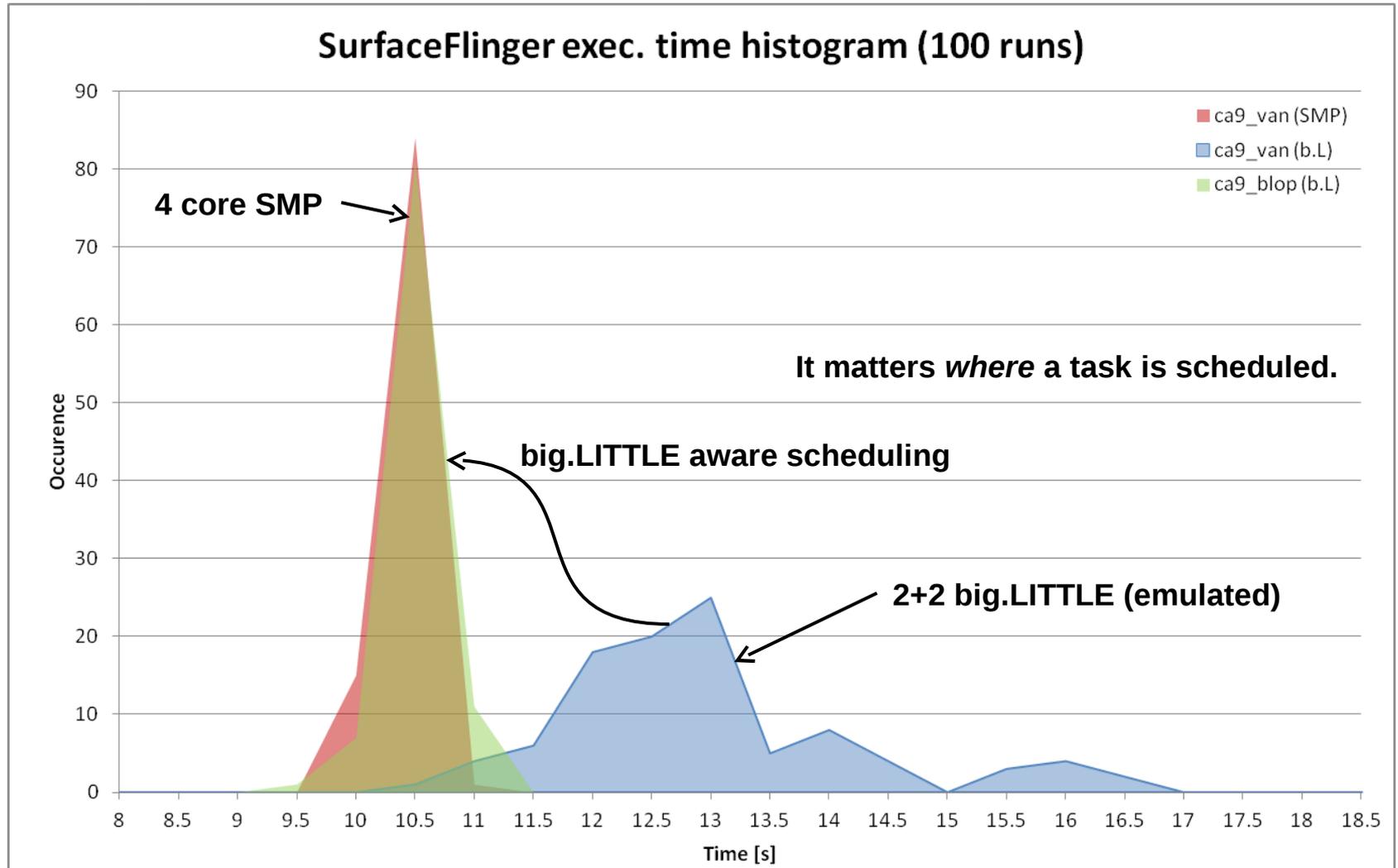
What is the (mainline) status?

- Example: Android UI render thread execution time.



What is the (mainline) status?

- Example: Android UI render thread execution time.



Mainline Linux Scheduler

- Linux has two schedulers to handle the scheduling policies:
 - RT: Real-time scheduler for very high priority tasks.
 - CFS: Completely Fair Scheduler for anything else and is used for almost all tasks.
- We need proper big.LITTLE/heterogeneous platform support in CFS.

- Load-balancing is currently based on an expression of CPU load which is basically:

$$cpu_{load} = cpu_{power} \cdot \sum_{task} prio_{task}$$

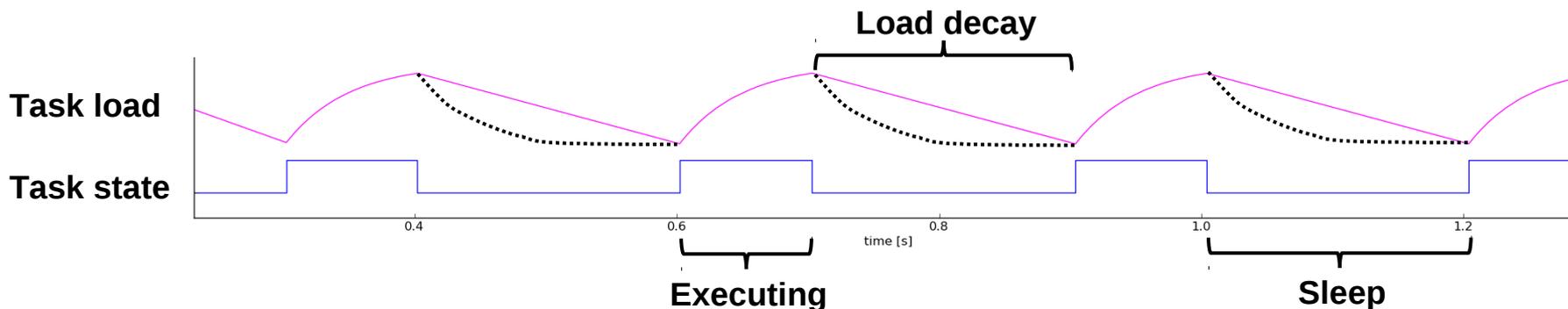
- The scheduler does not know how much CPU time is consumed by each task.
- The current scheduler can handle distributing task fairly evenly based on `cpu_power` for big.LITTLE system, but this is not what we want for power efficiency.

Tracking task load

- The load contribution of a particular task is needed to make an appropriate scheduling decision.
- We have experimented internally with identifying task characteristics based on the tasks' time slice utilization.
- Recently, Paul Turner (Google) posted a RFC patch set on LKML with similar features.
 - LKML: <https://lkml.org/lkml/2012/2/1/763>

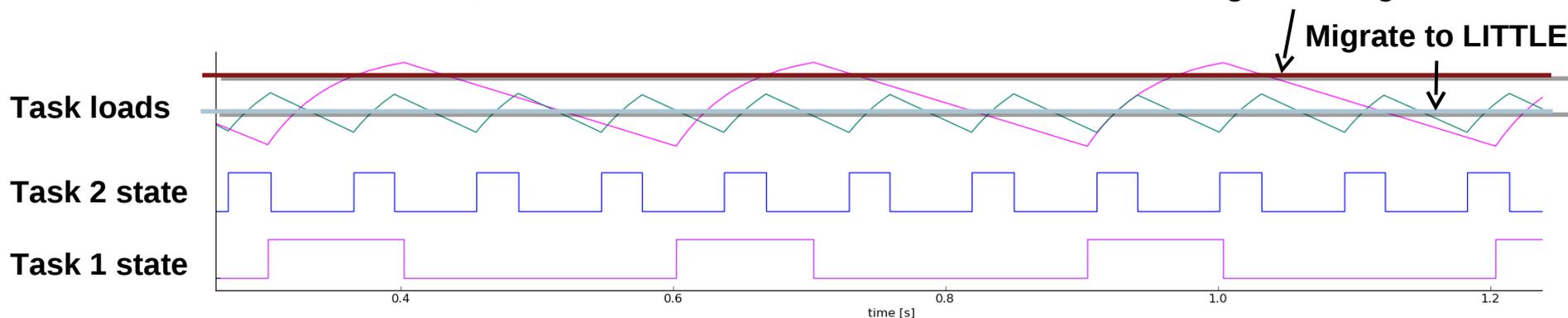
Entity load-tracking summary

- Patch set for improving fair group scheduling, but adds some essential bits that are very useful for big.LITTLE.
 - Tracks the time each task spends on the runqueue (executing or waiting) approximately every ms. Note that: $t_{\text{runqueue}} \geq t_{\text{executing}}$
 - The contributed load is a geometric series over the history of time spent on the runqueue scaled by the task priority.



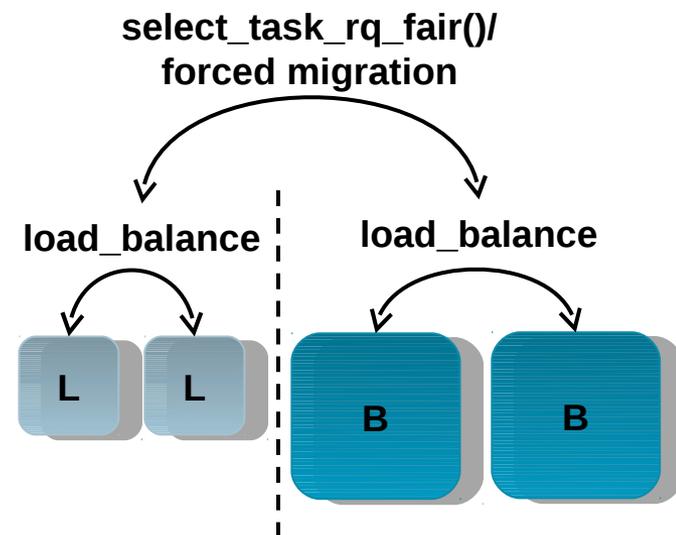
big.LITTLE scheduling: First stab

- Policy: Keep all tasks on little cores unless:
 1. The task load (runqueue residency) is above a fixed threshold, and
 2. The task priority is default or higher ($\text{nice} \leq 0$)
- Goal: Only use big cores when it is necessary.
 - Frequent, but low intensity tasks are assumed to suffer minimally by being stuck on a little core.
 - High intensity low priority tasks will not be scheduled on big cores to finish earlier when it is not necessary.
 - Tasks can migrate to match current requirements.



Experimental Implementation

- Scheduler modifications:
 - Apply PJTs' load-tracking patch set.
 - Set up big and little sched_domains with no load-balancing between them.
 - `select_task_rq_fair()` checks task load history to select appropriate target CPU for tasks waking up.
 - Add forced migration mechanism to push of the currently running task to big core similar to the existing active load balancing mechanism.
 - Periodically check (`run_rebalance_domains()`) current task on little runqueues for tasks that need to be forced to migrate to a big core.
 - **Note:** There are known issues related to global load-balancing.



Forced migration latency:
~160 us on vexpress-a9
(migration->schedule)

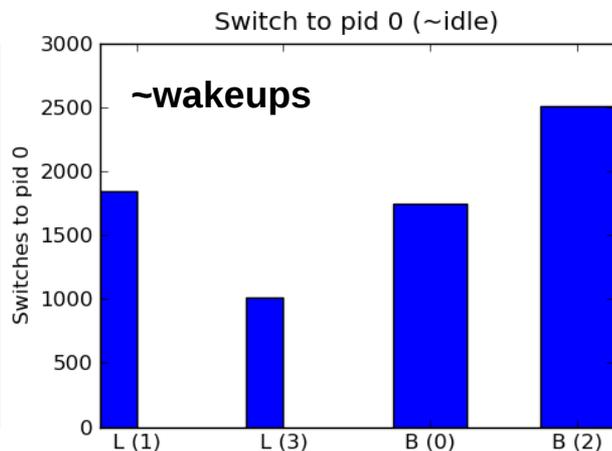
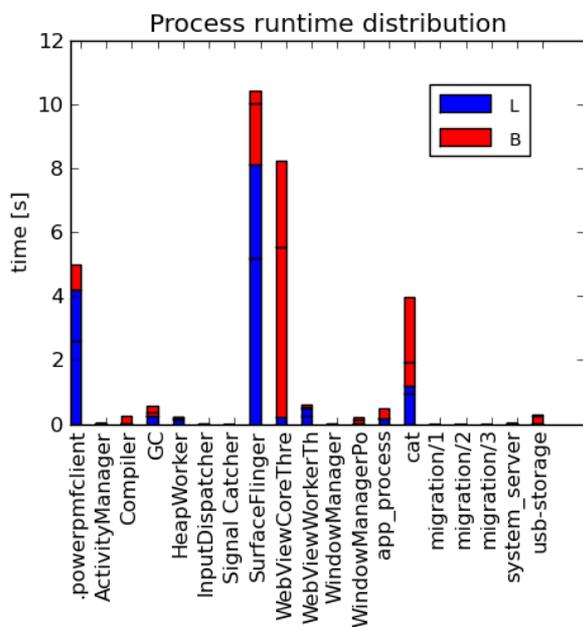
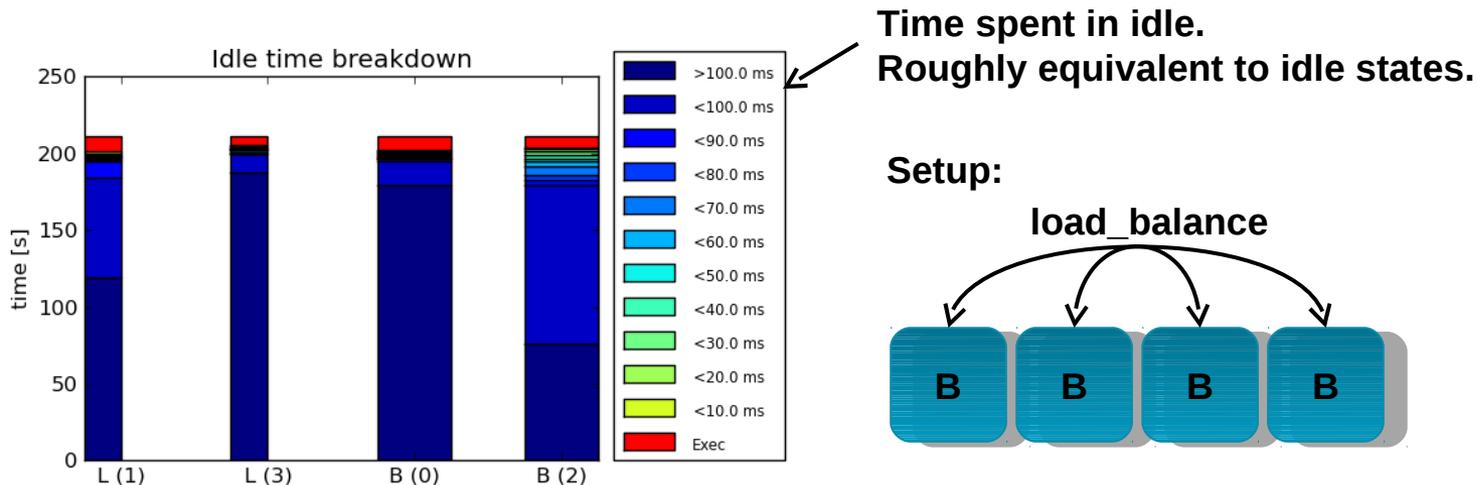
Evaluation Platforms

- ARM Cortex-A9x4 on Versatile Express platform (SMP)
 - 4x ARM Cortex-A9 @ 400 MHz, no GPU, no DVFS, no idle.
 - Base kernel: Linaro vexpress-a9 Android kernel
 - File system: Android 2.3
- LinSched for Linux 3.3-rc7
 - Scheduler wrapper/simulator
 - <https://lkml.org/lkml/2012/3/14/590>
 - Scheduler ftrace output extension.
 - Extended to support simple modelling of performance heterogeneous systems.

Bbench on Android

- Browser benchmark
 - Renders a new webpage every ~50s using JavaScript.
 - Scrolls each page after a fixed delay.
 - Two main threads involved:
 - WebViewCoreThread: Webkit rendering thread.
 - SurfaceFlinger: Android UI rendering thread.

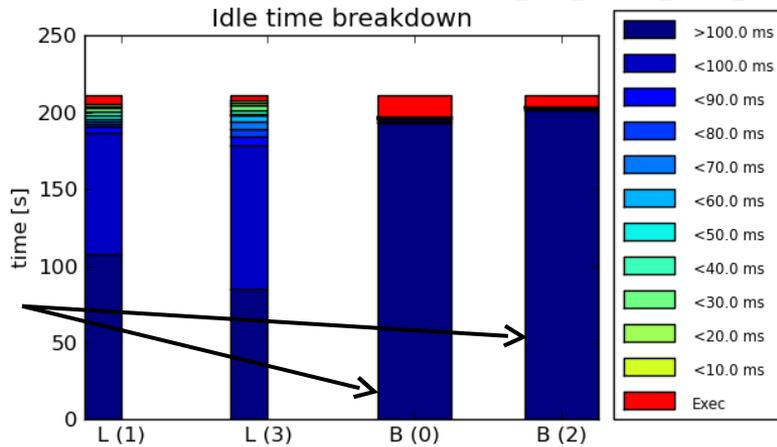
vexpress: Vanilla Scheduler



Note: big and little CPU's have equal performance.

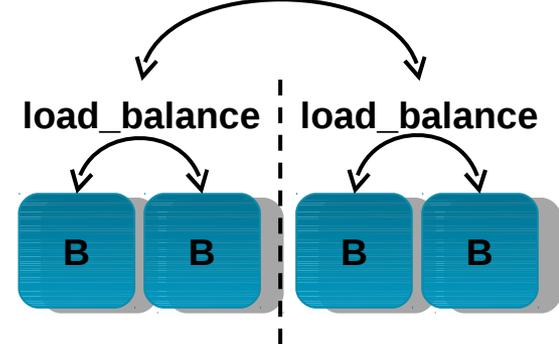
vexpress: big.LITTLE optimizations

Deep sleep
most of time

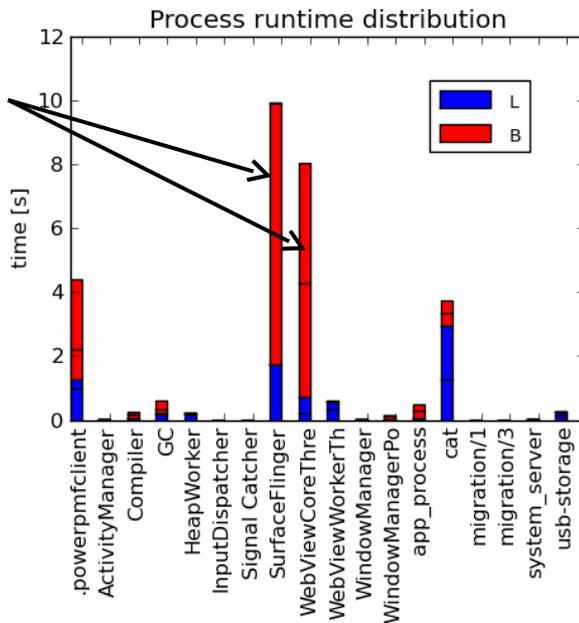


Setup:

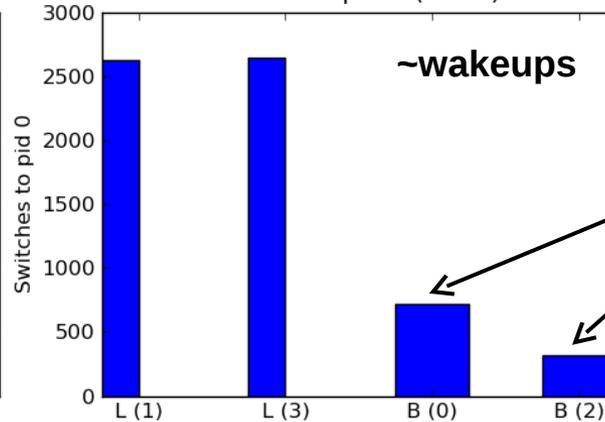
`select_task_rq_fair()/`
forced migration



Key tasks mainly
on big cores



Switch to pid 0 (~idle)



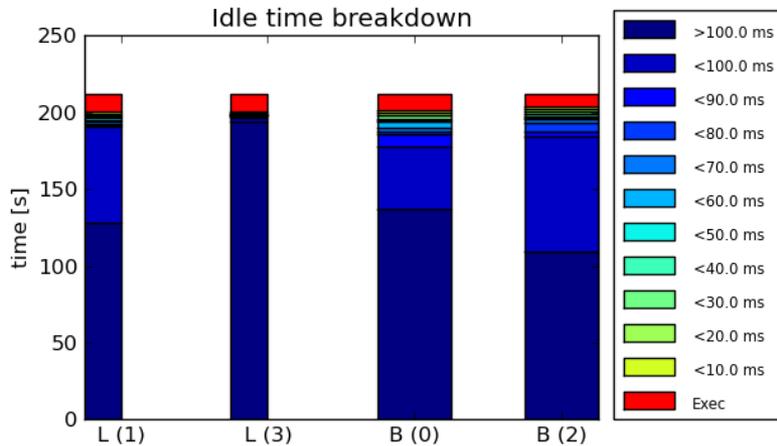
Note: big and little CPU's
have equal performance.

big.LITTLE emulation

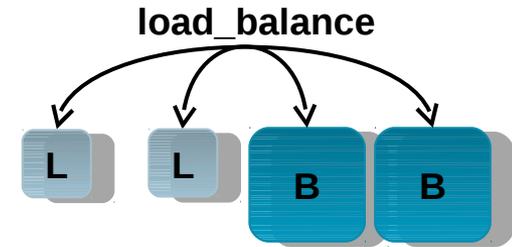
- Goal: Slow down selected cores on Versatile Express SMP platform to emulate big.LITTLE performance heterogeneity.
- How: Abusing perf
 - Tool for sampling performance counters.
 - Setup to sample every 10000 instructions on the little core.
 - The sampling overhead reduces the perceived performance.
 - Details:

```
perf record -a -e instructions -C 1,3 -c 10000  
-o /dev/null sleep 7200
```
 - Determined by experiments a sampling rate of 10000 slows the cores down by around 50%.
 - Very short tasks might not get hit by a perf sample, thus they might not experience the performance reduction.

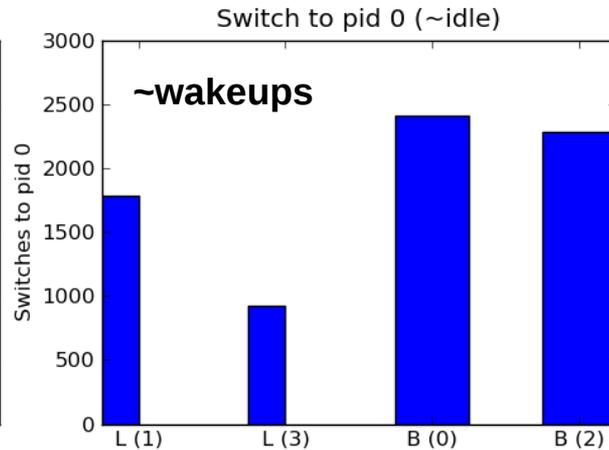
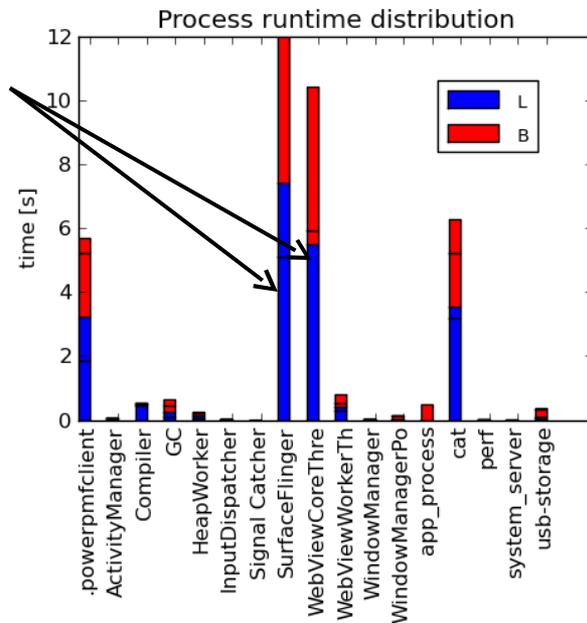
vexpress+b.L-emu: Vanilla kernel



Setup:



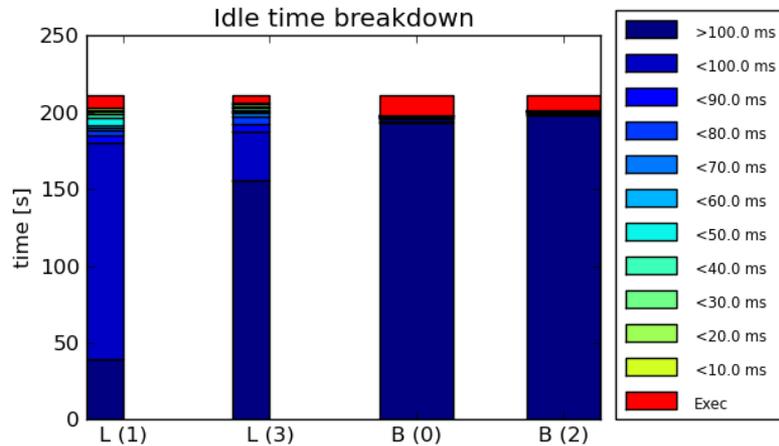
High little residency



Note: Task affinity is more or less random.

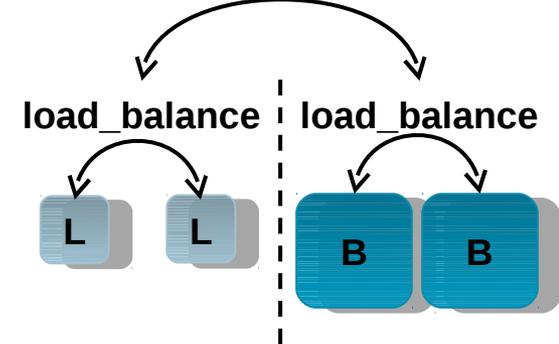
This is just one example run.

vexpress+b.L-emu: b.L optimizations



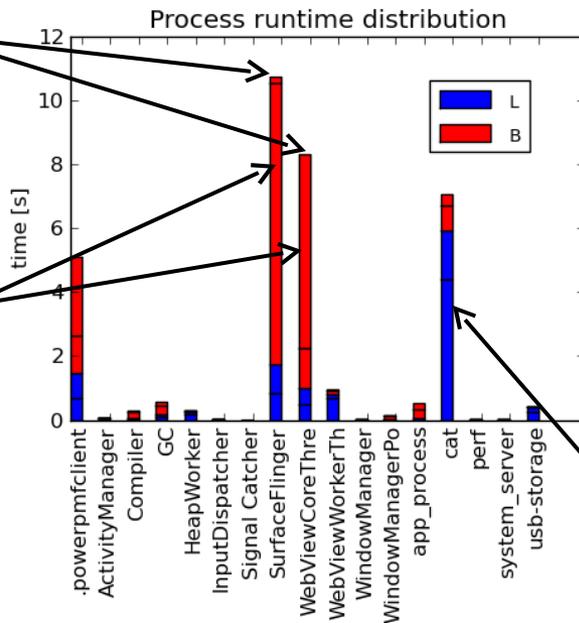
Setup:

`select_task_rq_fair()/`
forced migration

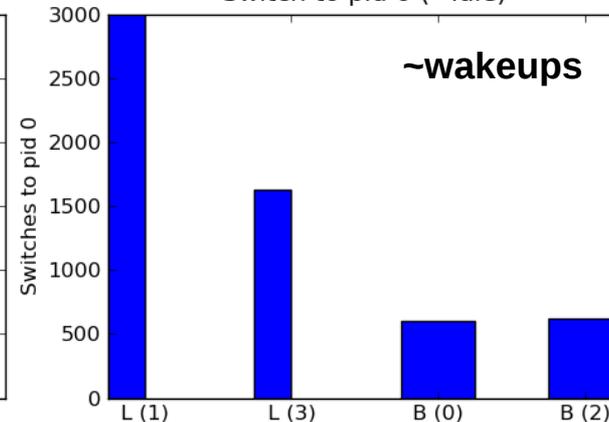


Shorter execution time.

Key tasks have higher big residency.



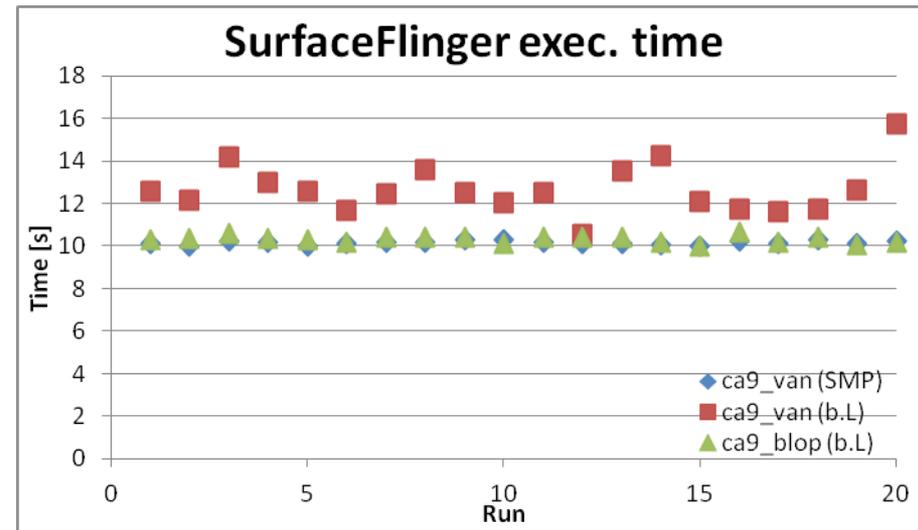
Switch to pid 0 (~idle)



Frequent short task has higher little residency.

vexpress+b.L-emu: SurfaceFlinger

- Android UI render task
- Total execution time for 20 runs:
 - SMP: 4xA9 no slow-down (upper bound for performance).
 - b.L: 2xA9 with perf slow-down + 2xA9 without.
- Execution time varies significantly on b.L vanilla.
 - Task affinity is more or less random.
 - The b.L optimizations solves this issue.

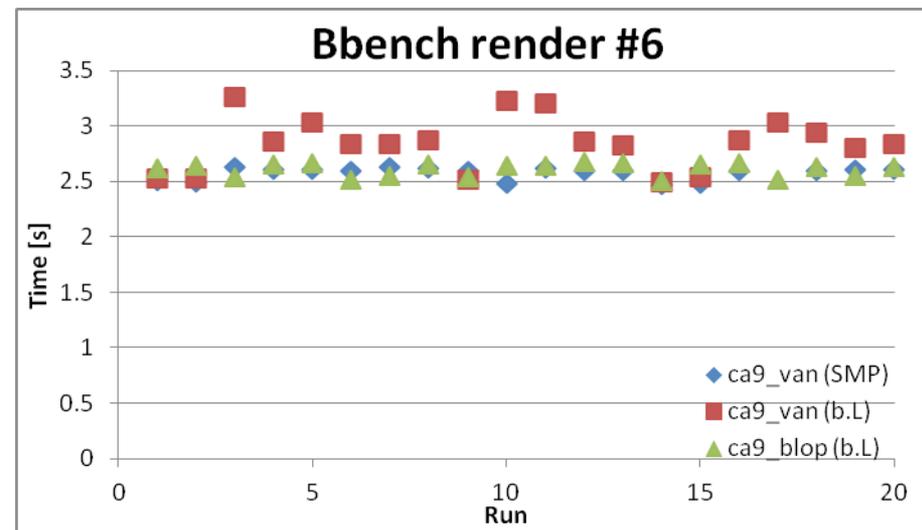


[s]	SMP	b.L van.	b.L opt.
AVG	10.10	12.68	10.27
MIN	9.78	10.27	9.48
MAX	10.54	16.30	10.92
STDEV	0.12	1.24	0.23

vexpress+b.L-emu: Page render time

- Web page render times
 - WebViewCore start -> SurfaceFlinger done
 - Render #2: Page scroll
 - Render #6: Load new page
- b.L optimizations reduce render time variations.
- Note: *No GPU* and low CPU frequency (400 MHz).

	[s]	SMP	b.L van.	b.L opt.
#2	AVG	1.45	1.58	1.45
	STDEV	0.01	0.11	0.01
#6	AVG	2.58	2.88	2.62
	STDEV	0.05	0.24	0.06



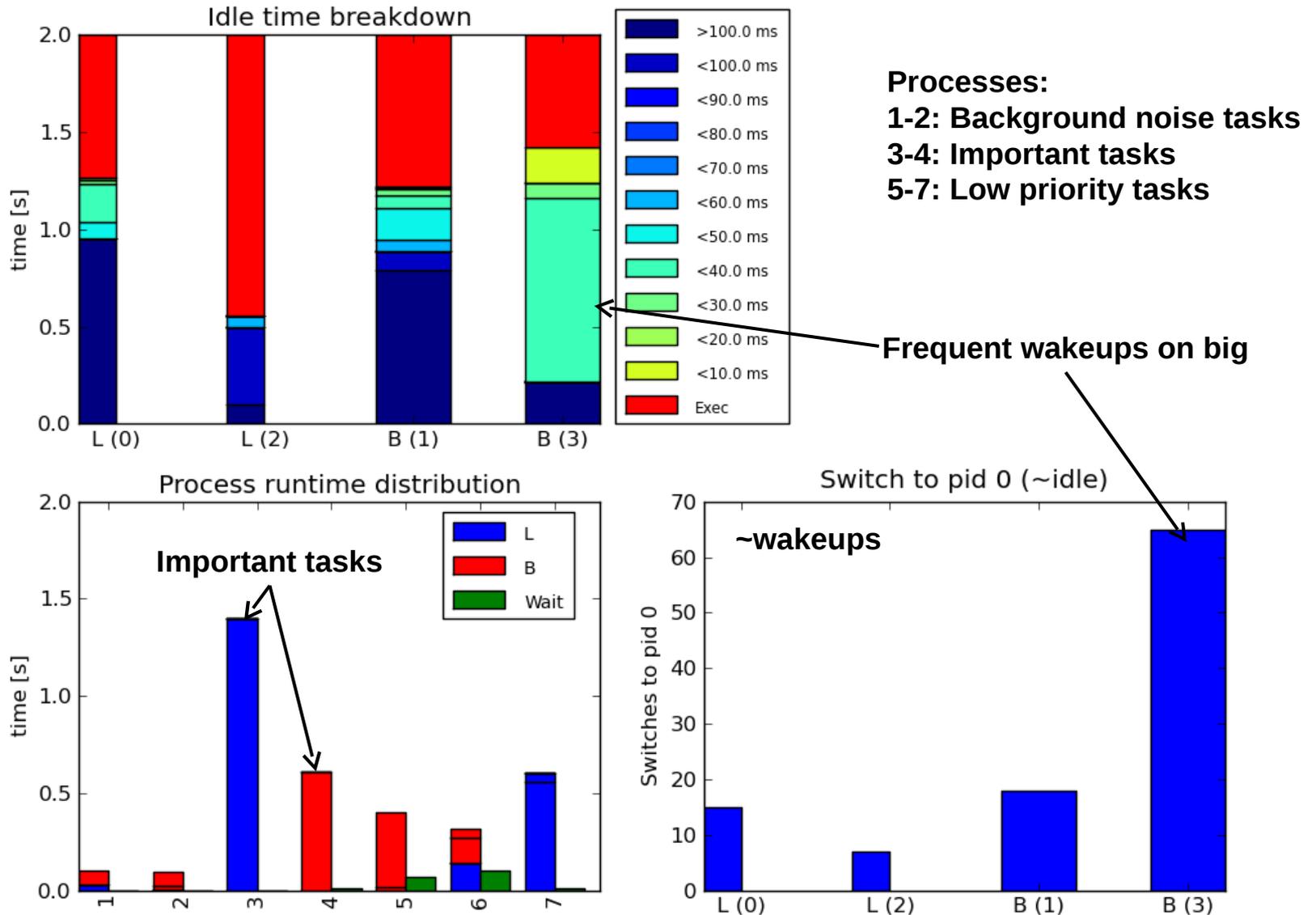
LinSched Test Case

- Synthetic workload inspired by Bbench processes on Android
- Setup: 2 big + 2 LITTLE
- big CPUs are 2x faster than LITTLE in this model.
- Task definitions:

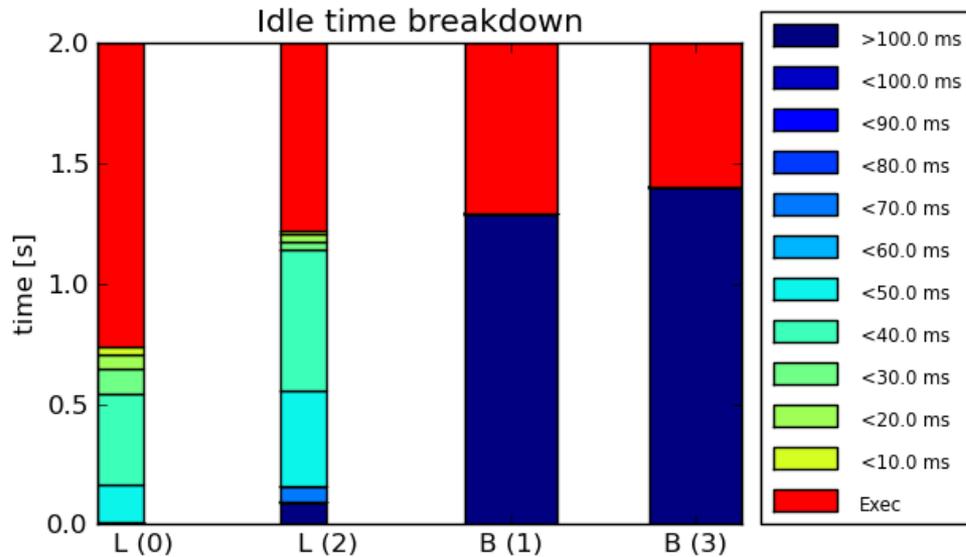
Task	nice	busy*	sleep*	Description
1+2	0	3	40	Background noise, too short for big
3	0	200	100	CPU intensive, big candidate
4	0	200	120	CPU intensive, big candidate
5	10	200	400	Low priority, CPU intensive
6	10	100	300	Low priority, CPU intensive
7	10	100	250	Low priority, CPU intensive

* [ms]

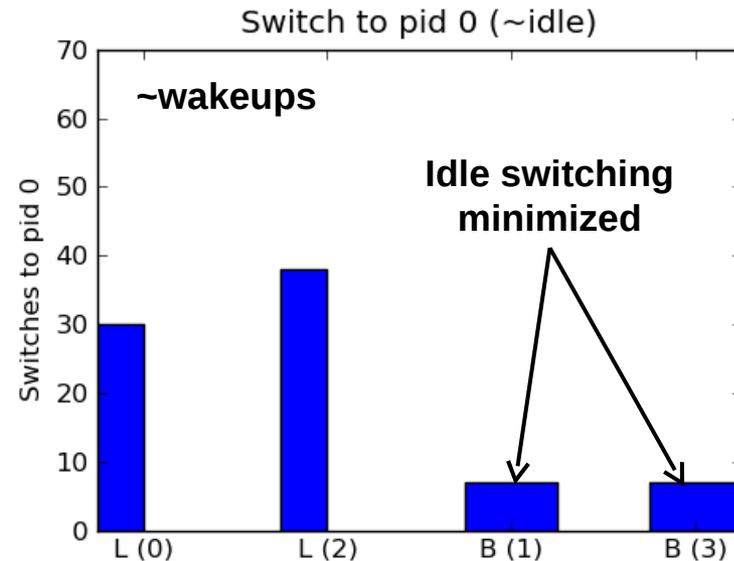
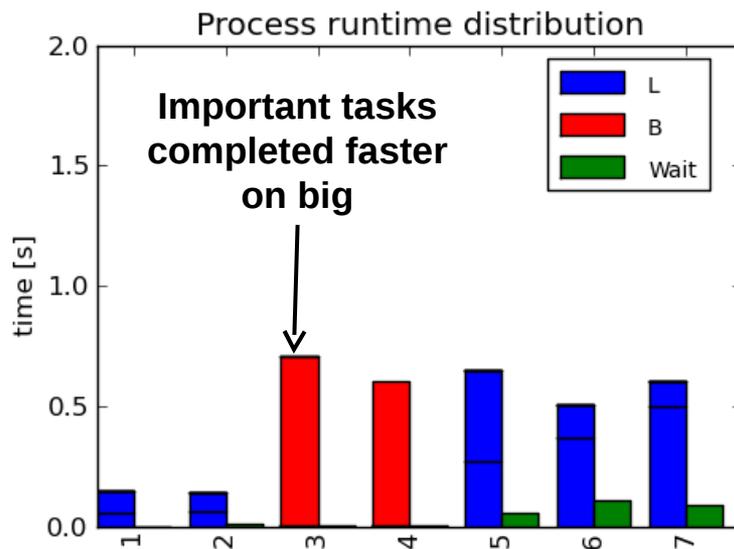
LinSched: Vanilla Linux Scheduler



LinSched: big.LITTLE optimized sched.

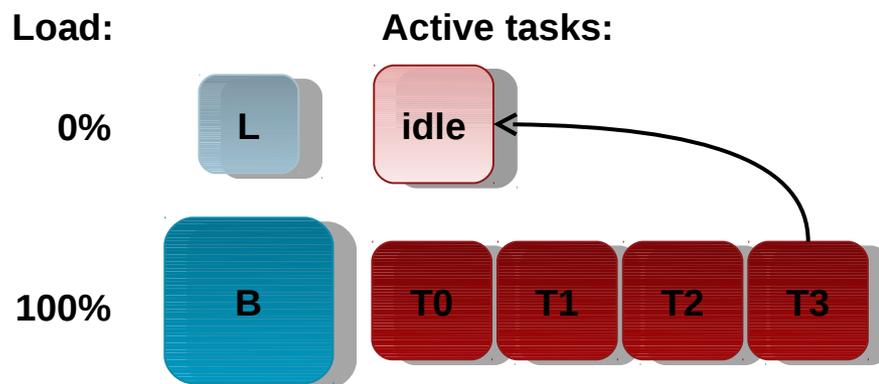


Processes:
 1-2: Background noise tasks
 3-4: Important tasks
 5-7: Low priority tasks



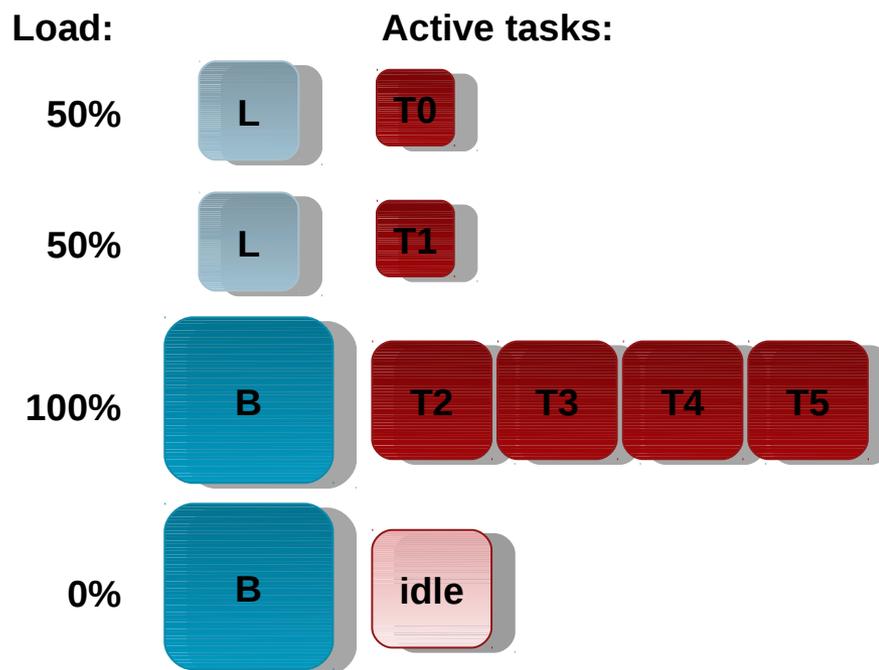
Next: Improve big.LITTLE support

- big.LITTLE sched_domain balancing
 - Use all cores, including LITTLE, for heavy multi-threaded workloads.
 - Fixes the sysbench CPU benchmark use case.
 - Requires appropriate CPU_POWER to be set for each domain.



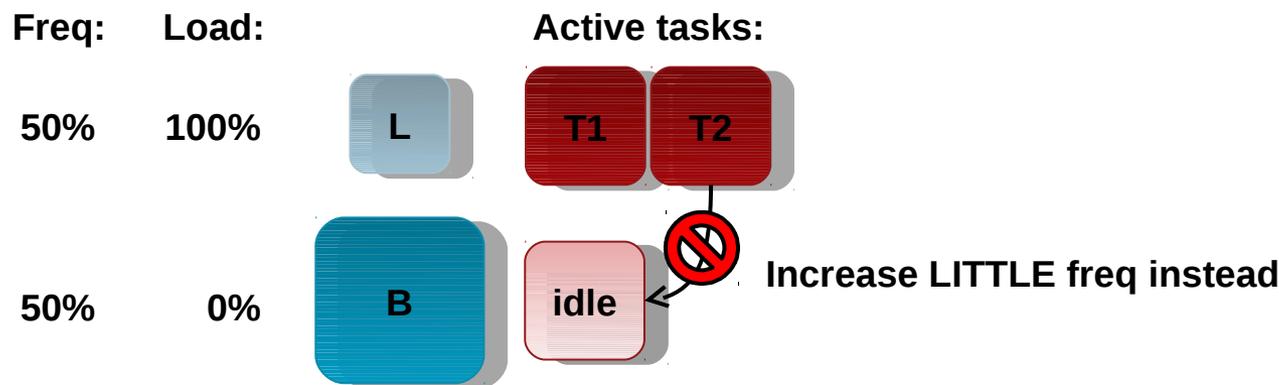
Next: Improve big.LITTLE support

- Per sched domain scheduling policies
 - Support for different load-balancing policies for big and LITTLE domains. For example:
 - LITTLE: Spread tasks to minimize frequency.
 - Big: Consolidate tasks to as few cores as possible.



Next: Improve big.LITTLE support

- CPUfreq -> scheduler feedback
 - Let the scheduler know about current OPP and max. OPP for each core to improve load-balancer power awareness.
 - This could improve SMP as well.
 - Ongoing discussions on LKML about topology/scheduler interface:
 - <http://lkml.indiana.edu/hypermail/linux/kernel/1205.1/02641.html>
 - Linaro Connect session: What inputs could the scheduler use?

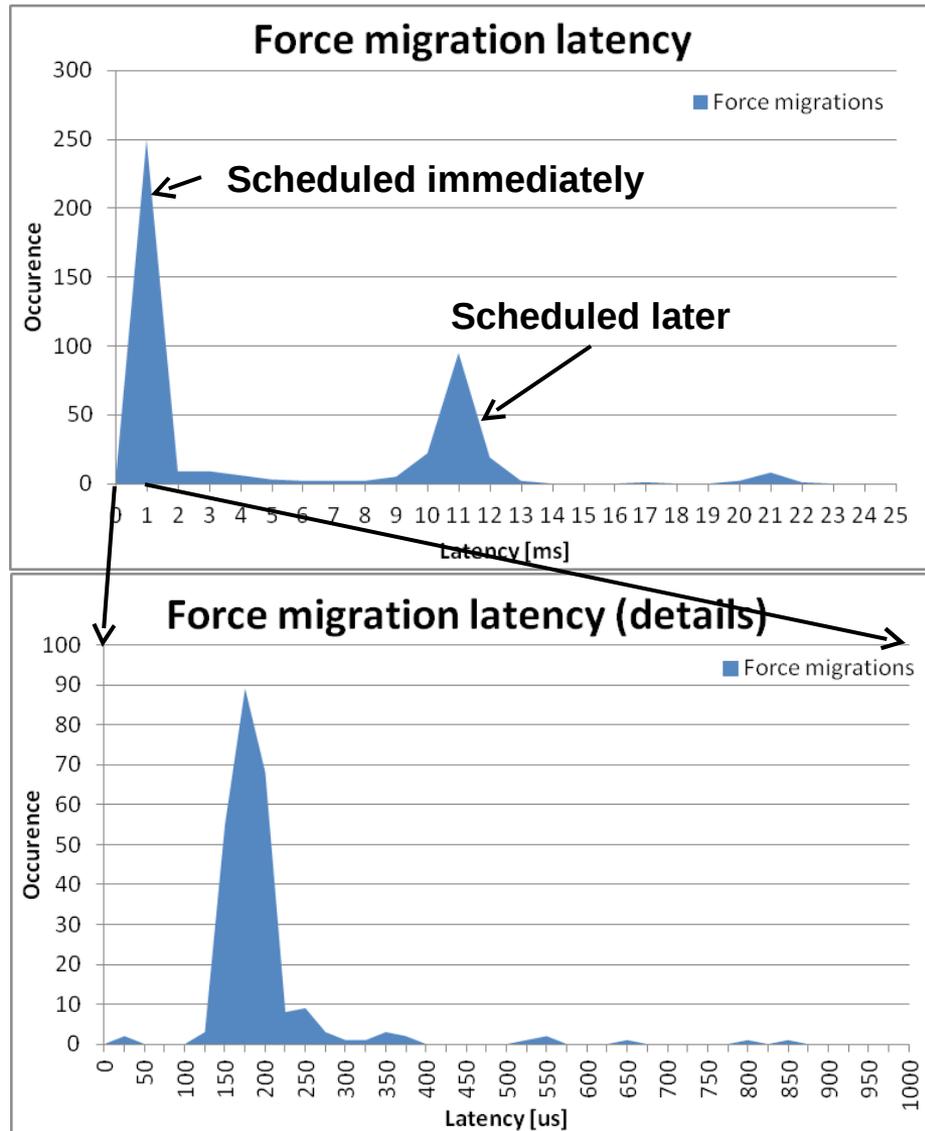


Questions/Discussion

Backup slides

Forced Migration Latency

- Measured on vexpress-a9
- Latency from migration -> schedule on target
 - ~160 us (immediate schedule)
 - Much longer if target is already busy (~10 ms)



sched_domain configurations

Vanilla

```
[ 0.372939] CPU0 attaching sched-domain:  
[ 0.373014] domain 0: span 0-3 level MC  
[ 0.373044] groups: 0 1 2 3  
[ 0.373172] CPU1 attaching sched-domain:  
[ 0.373196] domain 0: span 0-3 level MC  
[ 0.373222] groups: 1 2 3 0  
[ 0.373293] CPU2 attaching sched-domain:  
[ 0.373313] domain 0: span 0-3 level MC  
[ 0.373337] groups: 2 3 0 1  
[ 0.373404] CPU3 attaching sched-domain:  
[ 0.373423] domain 0: span 0-3 level MC  
[ 0.373446] groups: 3 0 1 2
```

big.LITTLE optimizations

```
[ 0.364272] CPU0 attaching sched-domain:  
[ 0.364306] domain 0: span 0,2 level MC  
[ 0.364336] groups: 0 2  
[ 0.364380] domain 1: does not load-balance  
[ 0.364474] CPU1 attaching sched-domain:  
[ 0.364500] domain 0: span 1,3 level MC  
[ 0.364526] groups: 1 3  
[ 0.364567] domain 1: does not load-balance  
[ 0.364611] CPU2 attaching sched-domain:  
[ 0.364633] domain 0: span 0,2 level MC  
[ 0.364658] groups: 2 0  
[ 0.364700] domain 1: does not load-balance  
[ 0.364742] CPU3 attaching sched-domain:  
[ 0.364764] domain 0: span 1,3 level MC  
[ 0.364788] groups: 3 1  
[ 0.364829] domain 1: does not load-balance
```