Wake-ups Effect on Idle Power for Intel's Moorestown MID and Smartphone Platform

German Monroy
April 13th 2010

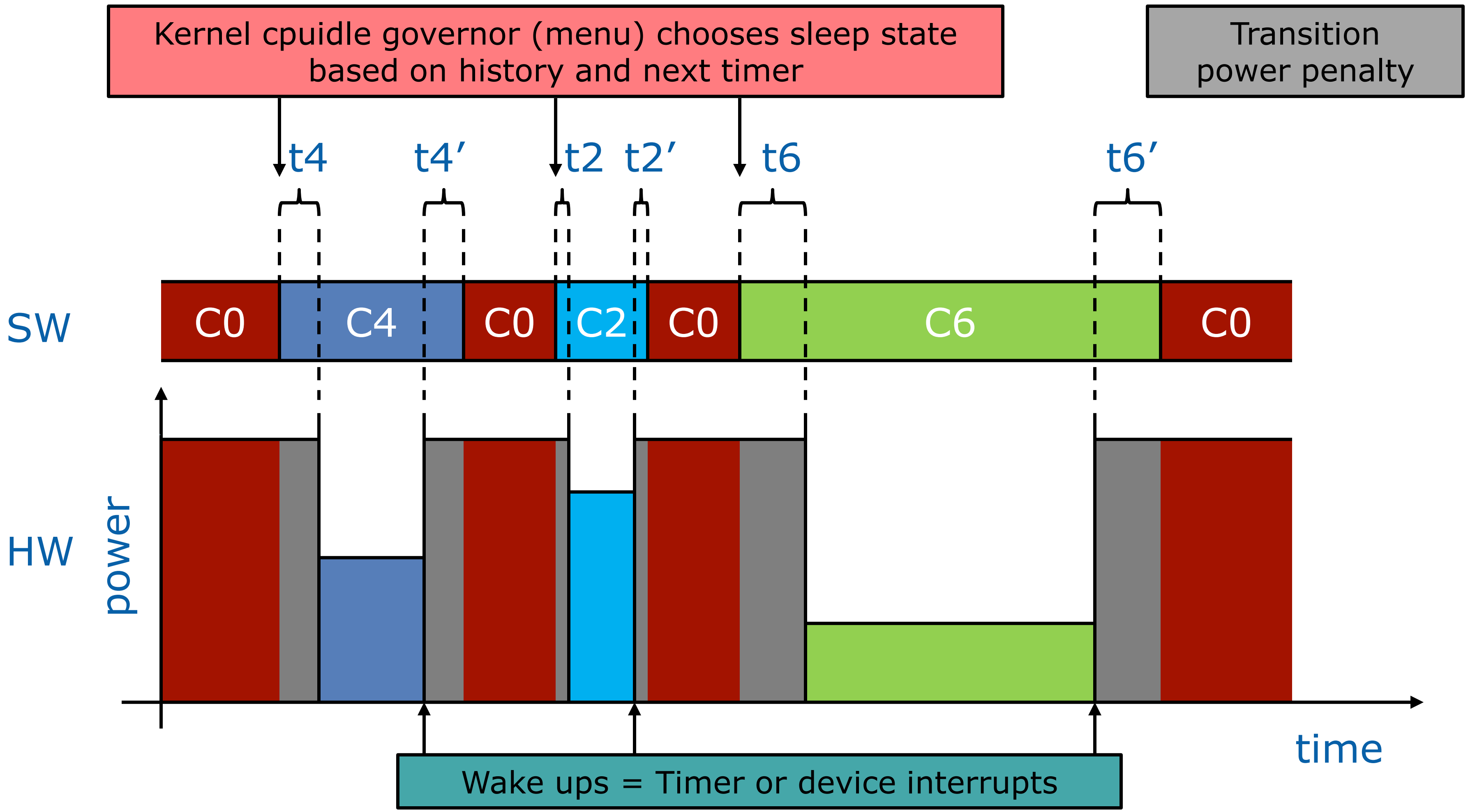
Collaborators: Arjan Van De Ven, Geoff Smith, Pierre Tardy, Mark Gross, Eshwar P

Problem Statement

- Idle power reduction is important for mobile computing
  - Time between battery charges for Laptops, Netbooks
  - Even more critical for handhelds
- Turn off hardware components when not needed
  - User inactive (e.g. in pocket)
  - User active (e.g. between frames during video playback)
- In particular
  - Atom processor supports deep CPU sleep states (C-states)
  - Moorestown extends to platform sleep states
- For a given idle scenario
  - Want to maximize utilization of the deepest sleep states

How can Linux be optimized to decrease platform power?

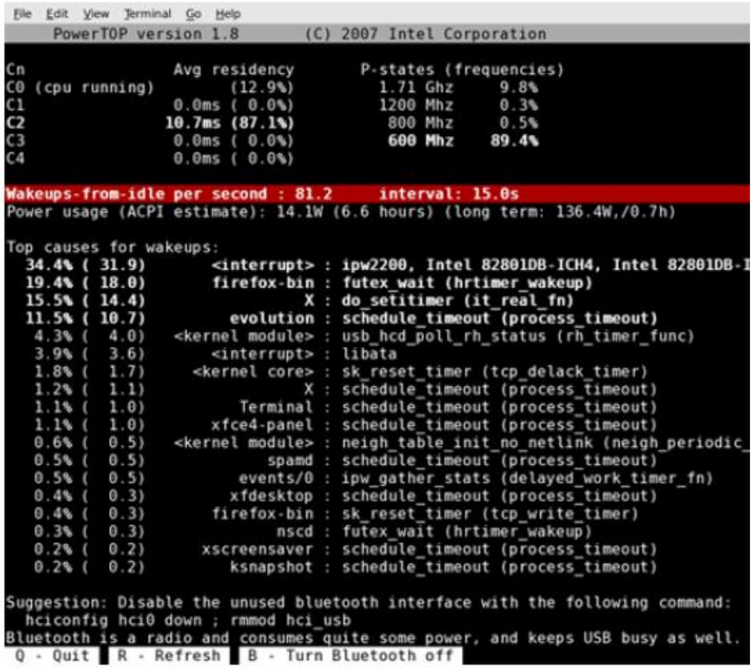
CPU Sleep States (C-states)



Lower power comes at the cost of longer latency

Tool #1: Powertop

- During specific interval
  - Average and % C-state residency
  - Average wakes from idle per second
  - Top wake-up offenders
    - Interrupts
    - application or kernel timers
- Great for optimizing a SW stack
  - Very easy to use
- Main limitations
  - Doesn't say when the wake ups happened
    - burst vs. periodic
    - in or out of sync
  - Doesn't really measure power



Tool helped optimize gnome-based Moblin to ~3 wakes per second

Tool #2: ftrace plus spreadsheet

- ftrace function tracer
  - Doesn't load system
    - ring buffer in memory, not storage
    - not dumped to storage until tracing has ended
  - Dynamic add-remove functions to trace
- Additional ftrace_printk sometimes needed
  - E.g. to find out which timer fired
    - output timer address at programming and at firing time
- Beware of local vs. global clock sources for timestamps
  - Timestamp coherency vs. cost tradeoff
  - For deep C-states may need (expensive) global clock source
    - since TSC hardware gets powered off
- Tip: Use spreadsheet's conditional formatting to color a trace
  - Helps in identifying patterns

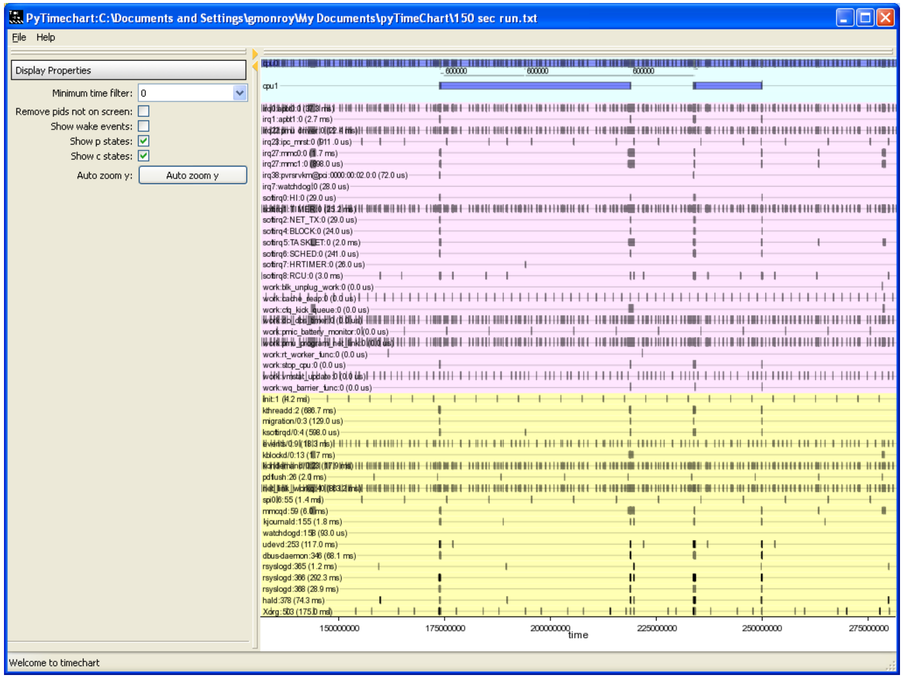
146.237005	5E-05	power_end	dummy=65535	
146.237055	2.5E-05	power_start	type=1	state=6
146.23708	3.4E-05	power_end	dummy=65535	
146.237114	3.3E-05	irq_handler_entry	irq=22	handler=pmu
146.237147	1E-05	sched_wakeup	task	net_link_workq
146.237157	1.7E-05	irq_handler_exit	irq=22	return=handled
146.237174	7E-06	softirq_entry	softirq=1	action=TIMER
146.237181	4.1E-05	softirq_exit	softirq=1	action=TIMER
146.237222	1.3E-05	sched_switch	task	swapper
146.237235	1.8E-05	workqueue_execution	thread=net_link_workq	40
146.237253	2.3E-05	sched_switch	task	net_link_workq
146.237276	0.002859	power_start	type=1	state=6
146.240135	4.8E-05	power_end	dummy=65535	
146.240183	5.5E-05	irq_handler_entry	irq=0	handler=apbt0
146.240238	1.1E-05	irq_handler_exit	irq=0	return=handled
146.240249	2.1E-05	softirq_entry	softirq=1	action=TIMER
146.24027	8E-06	sched_wakeup	task	net_link_workq
146.240278	2.6E-05	softirq_exit	softirq=1	action=TIMER
146.240304	0.000238	sched_switch	task	swapper
146.240542	6.9E-05	sched_switch	task	net_link_workq
146.240611	0.000399	power_start	type=1	state=5013
146.24101	5.1E-05	power_end	dummy=65535	
146.241061	2.4E-05	power_start	type=1	state=6

146.237005	5E-05	power_end	dummy=65535	
146.237055	2.5E-05	power_start	type=1	state=6
146.23708	3.4E-05	power_end	dummy=65535	
146.237114	3.3E-05	irq_handler_entry	irq=22	handler=pmu
146.237147	1E-05	sched_wakeup	task	net_link_workq
146.237157	1.7E-05	irq_handler_exit	irq=22	return=handled
146.237174	7E-06	softirq_entry	softirq=1	action=TIMER
146.237181	4.1E-05	softirq_exit	softirq=1	action=TIMER
146.237222	1.3E-05	sched_switch	task	swapper
146.237235	1.8E-05	workqueue_execution	thread=net_link_workq	40
146.237253	2.3E-05	sched_switch	task	net_link_workq
146.237276	0.002859	power_start	type=1	state=6
146.240135	4.8E-05	power_end	dummy=65535	
146.240183	5.5E-05	irq_handler_entry	irq=0	handler=apbt0
146.240238	1.1E-05	irq_handler_exit	irq=0	return=handled
146.240249	2.1E-05	softirq_entry	softirq=1	action=TIMER
146.24027	8E-06	sched_wakeup	task	net_link_workq
146.240278	2.6E-05	softirq_exit	softirq=1	action=TIMER
146.240304	0.000238	sched_switch	task	swapper
146.240542	6.9E-05	sched_switch	task	net_link_workq
146.240611	0.000399	power_start	type=1	state=5013
146.24101	5.1E-05	power_end	dummy=65535	
146.241061	2.4E-05	power_start	type=1	state=6

Tool #3: Event tracer plus pyTimeChart

- pyTimeChart
  - Written by Pierre Tardy in Python over wx and Chaco
  - Re-implementation of Arjan’s timechart
    - UI optimized for fast navigation
    - Fast even with big traces

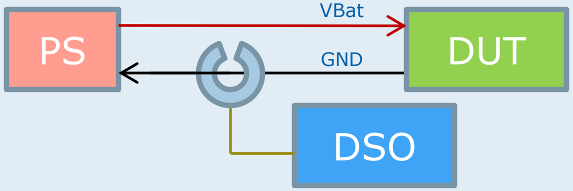
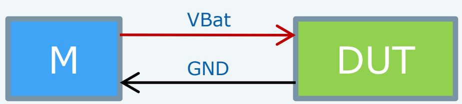
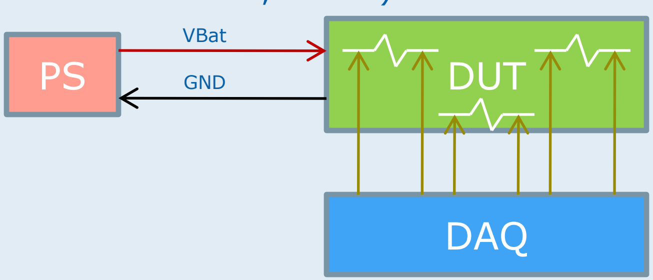
```
#test.sh
mount -t debugfs none /sys/kernel/debug 2>/dev/null
cd /sys/kernel/debug/tracing
echo 1 > options/global-clock
echo sched:sched_wakeup > set_event
echo sched:sched_switch >> set_event
echo workqueue:workqueue_execution >> set_event
echo power: >> set_event
echo irq: >> set_event
echo > trace
echo 1 > tracing_enabled
sleep 150
echo 0 > tracing_enabled
cat trace > ~/trace.txt
```



Tool #4: Measure Platform Power

- Software traces (ftrace, event trace) only give a part of the power picture
- Need to measure platform power which is the ultimate goal
  - Average power
  - Power behavior over time
  - Correlation between SW traces and instantaneous power
- Find the cost of transitions at the platform level
  - E.g. characterize sleep state parameters
- Discovered that sometimes sleep states are recorded in SW but did not happen in HW (the small interval ones)
  - Aborted early by interrupts

Power Measurement Alternatives

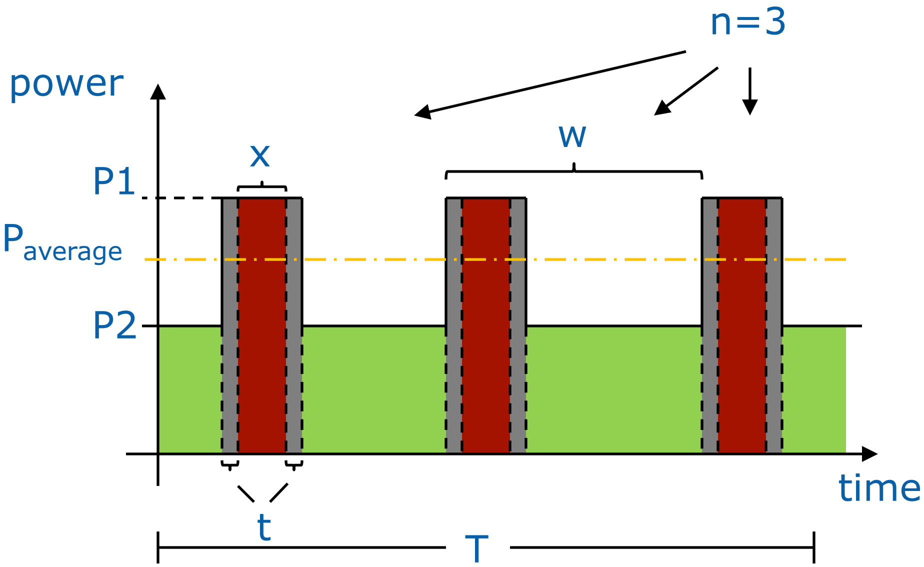
Equipment + Setup	Pros	Cons
Digital Storage Oscilloscope + Current Probe + DC Power Supply 	• Inductive coupling • Useful for other things (multiple signals, general troubleshooting) • High time resolution	• Cost • Hard to measure platform average power due to small dynamic range (8 bit ADC).
Monsoon* Power Monitor 	• Built-in variable power supply and sense resistor • Great to compute average power • Inexpensive • Easy to use software	• 200us integrator sample period
Data Acquisition Equipment (e.g. National Instruments*, Fluke*) 	• Multiple, individual power rails • Root-cause issues at component level	• Cost • Needs precision sensing resistors • Setup time

*Third-party brands and names are the property of their respective owners.

Tool #5: Create a model

- Correlate SW with HW
- Find out transition cost
  - time
  - power
- Estimate power in terms of wake ups per second
- Determine wake-ups per second target
  - Derived from power target

Average Power Model



$$P_{average} = \frac{P_1 \cdot T_1 + P_2 \cdot T_2}{T}$$

where

$$T_1 = n \cdot (x + t)$$

$$T_2 = T - T_1$$

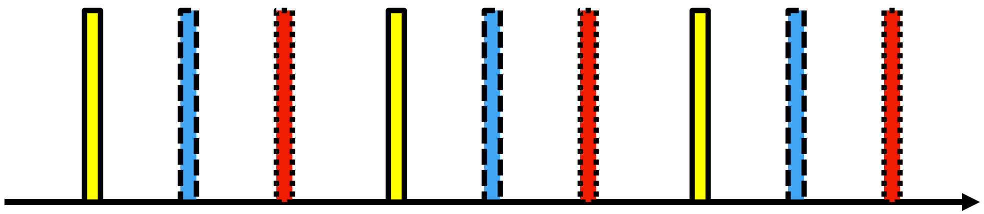
$$w = \frac{T}{n}$$

P1: Power (high)
P2: Power (low)
x: execution time (C0)
t: transition time
T: period of measurement
n: # wake ups in T
w: time between wake-ups

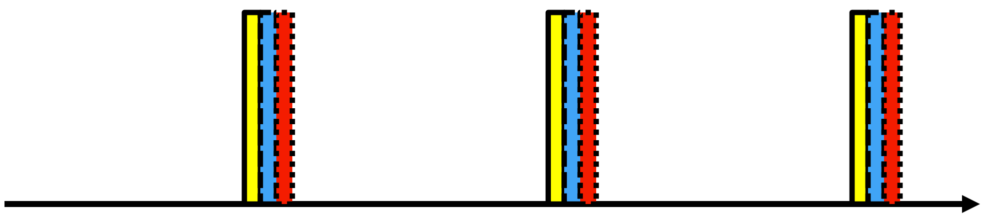
$$P_{average} = \frac{x+t}{w}\left(P_1 - P_2\right) + P_2$$

The timer sync problem

- 3 3-sec timers (applications and/or kernel)
  - Can wake up the platform every second



  - or every 3 seconds



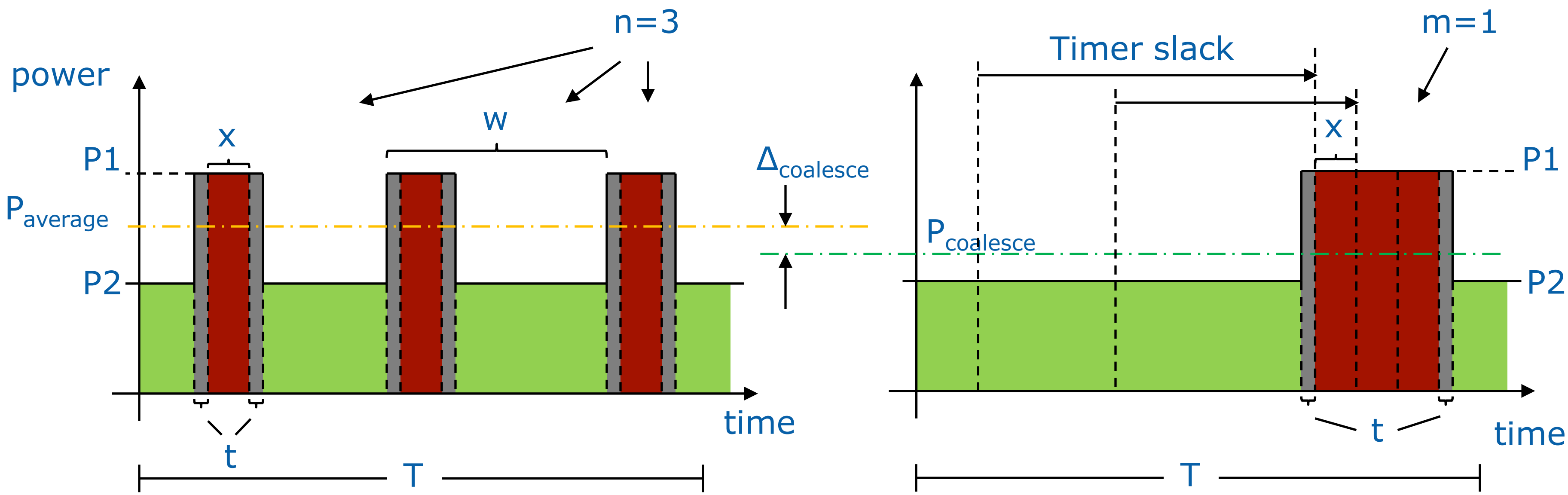
- Depending on relative delay of each timer being programmed
  - Impacts power savings of sleep states
- Solution: timer coalescing
  - Timer owner explicitly defines a “time slack” property of the timer

Coalescing Timers

Kernel Change	Description	Status
Arjan van de Ven's "range" for highres timers	Apps can specify slack range for their own timers (i.e. poll/select, futex and nanosleep system calls) via prctl (per process)	Upstream since Sep 2008
Platform range default changeable via sysfs	Allow power management policy to override system-wide range default when under special conditions (e.g. in a pocket)	To be submitted upstream
Arjan's "timer slack" for legacy timers	Kernel code can specify slack explicitly via new set_timer_slack() API E.g. before calls to add_timer() or mod_timer()	Proposed Feb 2010, not yet upstream
Make some kernel timers deferrable	Only non critical timers E.g. cache_reap (return free memory to the system)	To be submitted upstream
Timers active during idle use set_timer_slack()	For critical timers (risk of data loss or security vulnerability) E.g. memory page write-back, commit ext3 journal to disk, garbage collectors for network protocols	To be submitted upstream
Fixes written by Geoff Smith	• Remove unnecessary no_hz ticks from the RCU disposal subsystem • Look ahead for tick_sched_timer() in hrtimer_interrupt()	To be submitted upstream

Decreased wake ups 3x,
from 2.4 to 0.8 per second, with just a 1 sec slack

Power Effect of Timer Coalescing



$$P_{average} = \frac{x+t}{w}(P_1 - P_2) + P_2$$

$$P_{coalesce} = \frac{P_1 \cdot T_1 + P_2 \cdot T_2}{T}$$

where

$$T_1 = n \cdot x + m \cdot t$$

$$T_2 = T - T_1$$

$$\Delta_{coalesce} = P_{average} - P_{coalesce}$$

$$\Delta_{coalesce} = \frac{(n-m) \cdot t \cdot (P_1 - P_2)}{T}$$

Timer slack coalesces timers, can save power

Benefit of Timer Coalescing

			A	B	C	D	E	F	G
Power (high)	**P1**	mw	1000	1000	1000	1000	1000	1000	1000
Power (low)	**P2**	mw	100	500	1	100	100	100	100
Period of measurement	**T**	ms	1000	1000	1000	1000	1000	1000	1000
Execution time (C0)	**x**	ms	10	10	10	100	1	10	10
transition time	**t**	ms	1	1	1	1	1	10	1
# wake-ups in T	**n**		5	5	5	5	5	5	50
# wake-ups in T, coalesced	**m**		1	1	1	1	1	1	1
time between wake-ups	**w**	ms	200	200	200	200	200	200	20
Average Power	**P_average**	mw	150	528	56	555	109	190	595
Average Power, coalesced	**P_coalesce**	mw	146	526	52	551	105	154	551
Power saving of coalescing	**Δ_coalesce**	mw	4	2	4	4	4	36	44
Power saving of coalescing	**Δ_coalesce**	%	2%	0%	8%	1%	3%	23%	8%

- In some cases coalescing saves power
- Coalescing helps when:
  - Big difference between High and Low power
  - Transition time significant with respect to execution time, or
  - Lots of wake ups get coalesced together

Platform variables determine impact of coalescing timers

Putting it all together

1. Optimize the SW stack, using powertop
2. Analyze timer behavior using ftrace
3. Further understand kernel behavior with pyTimeChart
4. Measure platform power
5. Correlate traces with platform power to create a model
6. Use model to set a wake-up target
7. If needed, coalesce timers to achieve the wake up target
   - Set ranges for user space
   - Set slacks for kernel timers

Next Steps

- Try in other SW stacks
- Try in active use cases (as opposed to idle)
- Devise a mechanism to track timer behavior over time
  - Single tool
  - Quicker turn around
  - Integrate /proc/timer_stats with event tracer?
  - Perf based?

Additional Info

- Lesswatss.org: Saving power with Linux
  http://www.lesswatts.org/
- Petter Larsson’s SW Development Recommendations for Intel® Atom™ based MID platforms:
  http://software.intel.com/en-us/articles/power-efficiency-analysis-and-sw-development-recommendations-for-intel-atom-based-mid-platforms/
- Monsoon Power Monitor
  http://www.msoon.com/LabEquipment/PowerMonitor/
- Pierre Tardy’s pyTimechart for ftrace
  http://gitorious.org/pytimechart/pages/Home
- Arjan van de Ven’s range capability for hrtimers
  http://lwn.net/Articles/296548/
- Arjan van de Ven’s slack for legacy (non high-res) timers
  http://lwn.net/Articles/369549
- Contact us
  german.monroy@intel.com

Questions?