Advanced Linux Power Management Evaluation using Perf

Hagen Paul Pfeifer <hagen@jauu.net>

Perf Power Analyzer

perf power-analyzer == perf script, bundled with perf, developed in kernel tree

Debian: apt-get install linux-perf

Widely available, even on embedded systems like Yocto or Buildroot. No complex installation, which is often difficult on embedded systems

Not yet committed upstream, still needs some last polish. URL for testing at the last slide

```
$ perf script record power-analyzer -a -- sleep 60
[ perf record: Woken up 77 times to write data ]
[ perf record: Captured and wrote 23,165 MB perf.data (246395 samples) ]

$ perf script report power-analyzer --help
usage: power-analyzer.py [-h] [-m
[{idle-cluster,task,timer,frequency,wakeups-timemap,idle-governor,all}]] [-C CPU] [--extended]
[-v] [--file-out]
```

Info: record power-analyer will capture all required events for all modes!

Supported Modes

Functionality are grouped into "modes"

- Modules can be used as required, focused and with minimal overhead.
- Different modes require different tracepoints. It is often more efficient to manually record the events that are actually required for the specific mode, via `perf record -e <event>`

Available Modes (subset):

Timer	Frequency	Wakeup-Timemap	Idle-Governor
User & kernel space analysis of timers with impact on wakeups as well as timer tick analysis for tickless systems	Creates a chronological overview of the P-States across all cores. Also maps this data to the executed tasks.	Simple chronological overview of wakeups per time unit for tasks, applied at task or CPU level	Provides characteristics of the selected idle governor, for example a residency analysis. Enables the governors to be benchmarked against each other

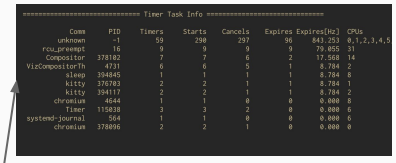
Power Analyzer Modes & Post-Processing

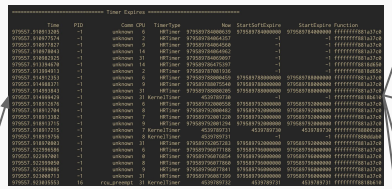
Use Case: Timer Module

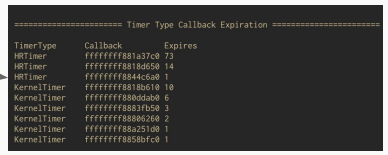
1. Recording
2. Analyzation
3. Post-Processing

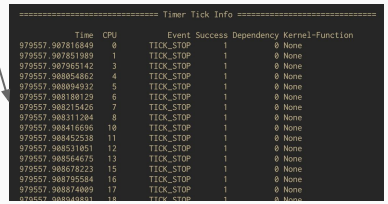
```
perf script record
power-analyzer
or
perf record -e 'timer:*'
```

```
perf script report
power-analyzer -m timer
```

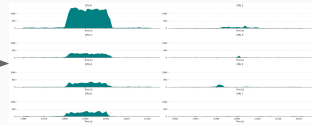




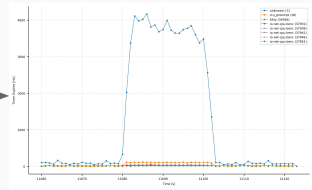




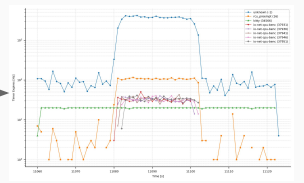
timer-expires-cpu.py



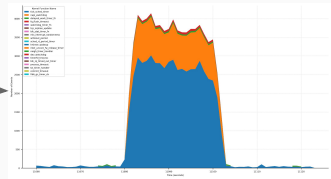
timer-expires.py



timer-expires-log.py



timer-expires-unknown.py

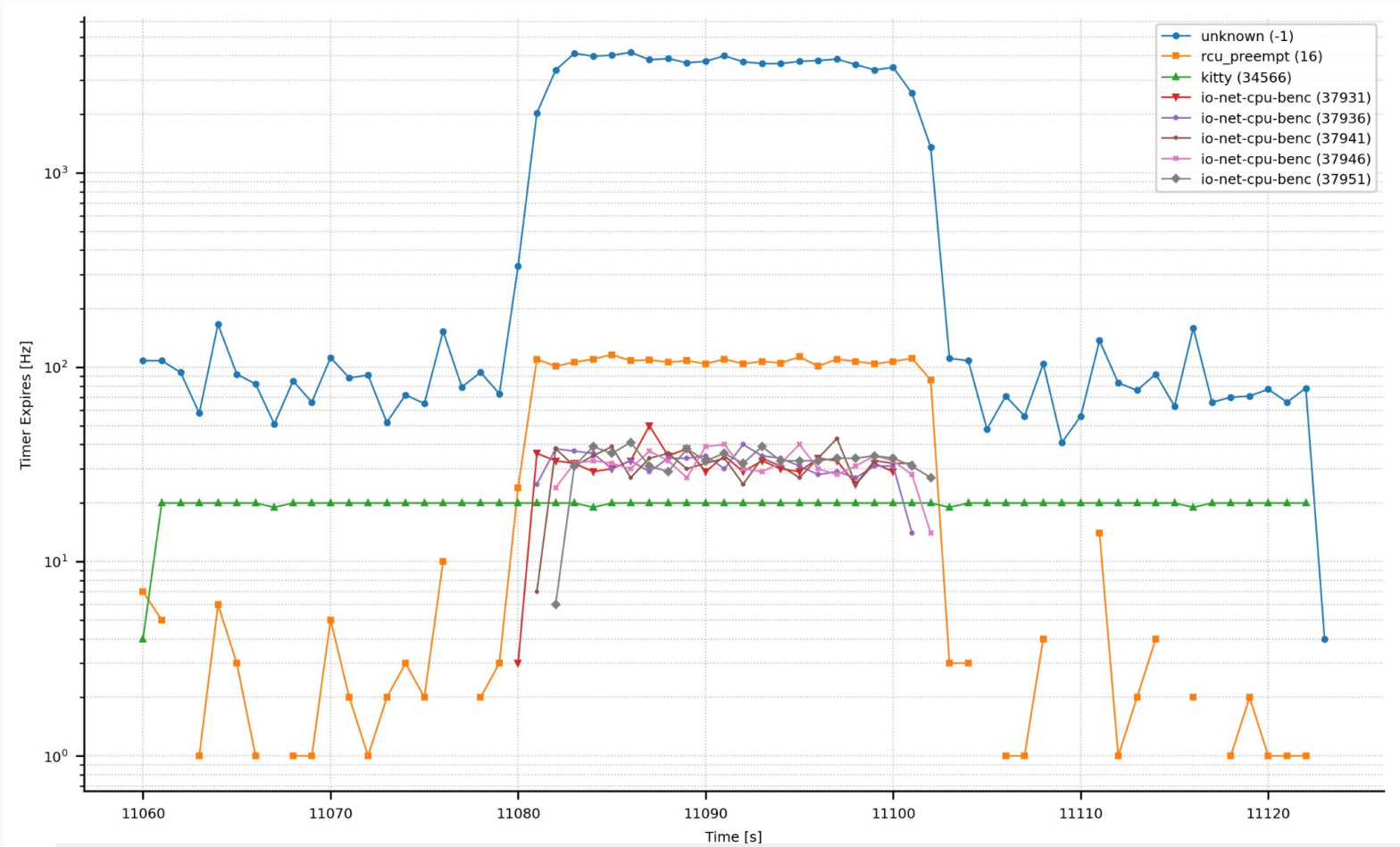


Required events are documented

Command line output is clear and easy to read - even in target. It provides a quick and detailed overview at the same time.

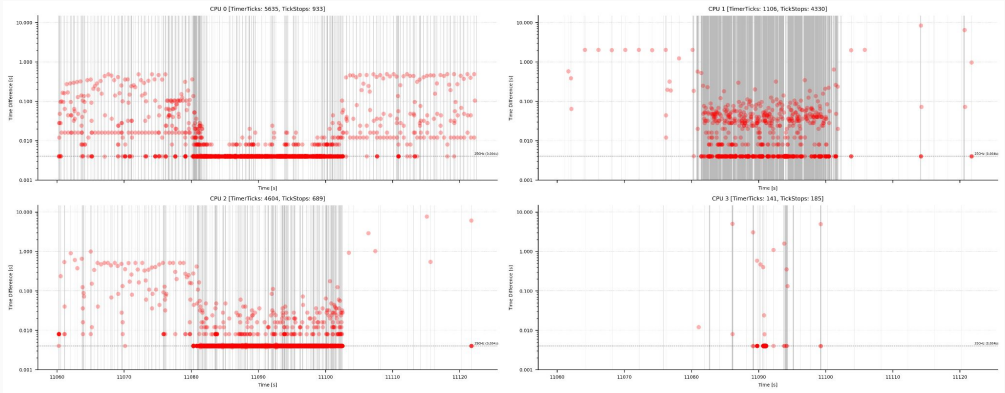
Post-processing helps with visualization, big data analysis or filtering

Showcase Timer Module

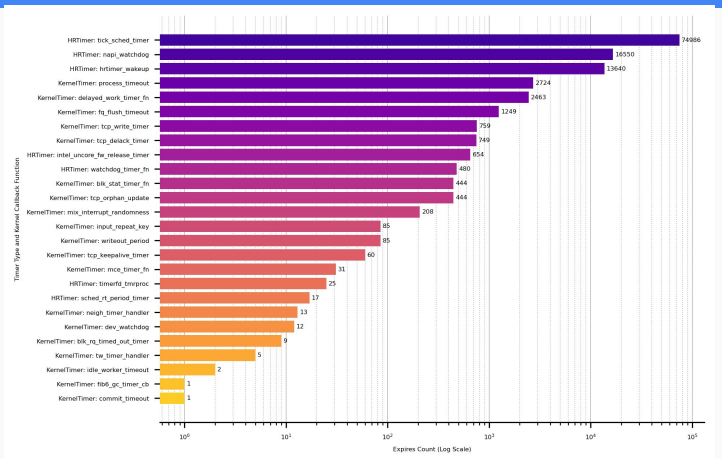


Timer expires over time, mapped to PID, unknown classified timer expires are kernel timer (high resolution timer or classic kernel timer, see next slide)

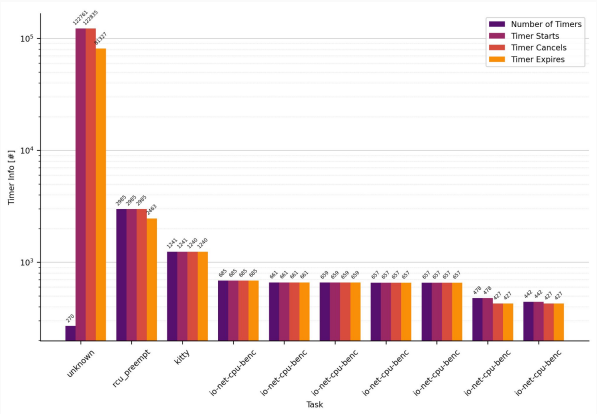
Showcase Timer Module II



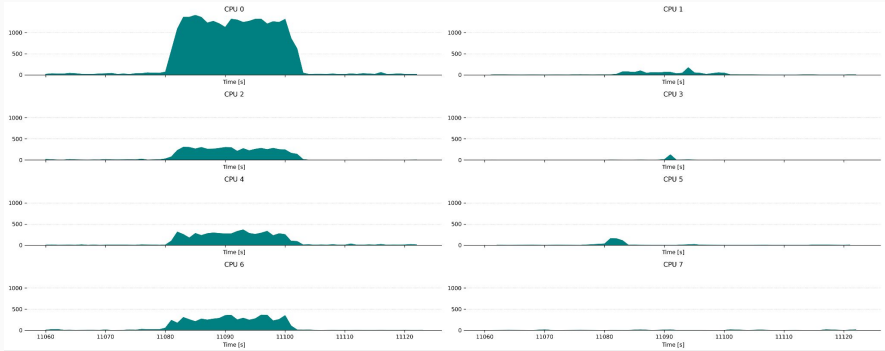
Timer tick behavior on a tickless (CONFIG_NO_HZ_FULL) system over time. Red: time difference between ticks. Gray: tick stop commands



Kernel Timer expires frequency. Here: timer tick, followed by NAPI during NIC load, hrtimer wakeups (e.g. timer expired),



Timer statistics per process, including number of timers, starts, cancel and expirations.



Time expires (not cancels) per CPU core

General Options

--extended

Some of the analyses are computationally intensive and are not always required immediately ("expert analysis"). To switch these on, add --extended

--cpu <n>

Often you want to limit yourself to one CPU - e.g. for C-state analysis - to limit the output, this option can be used. Note: it is often better to limit the recording during recording time (perf record -C <n>)

--file-out

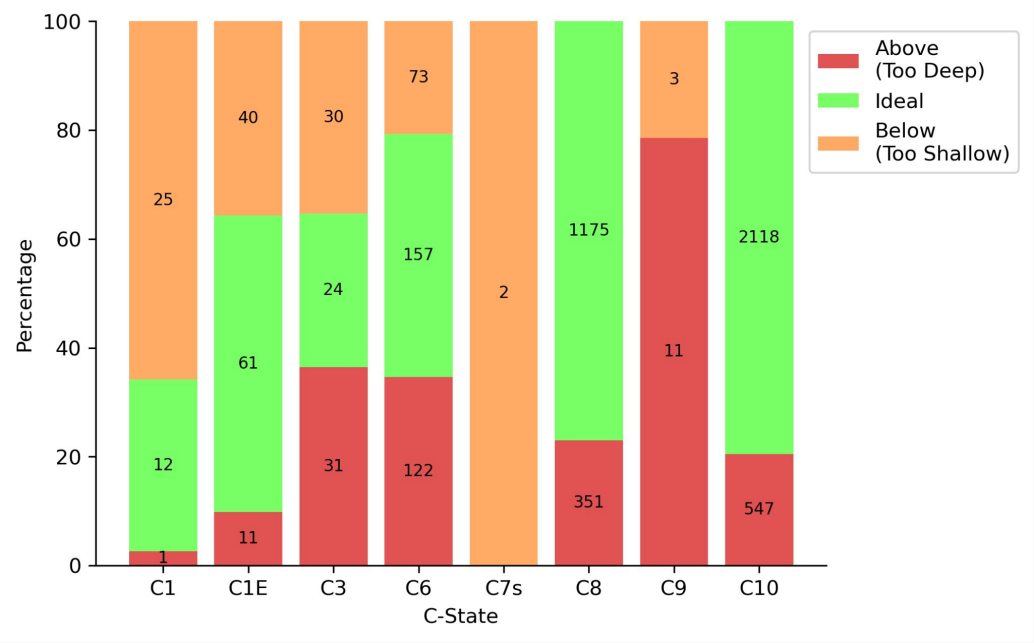
Some of the analyses call for post-processing. For example in numpy and matplotlib. To simplify this, the data is written to dedicated files

In order to make post-processing easy to use, the design was based on the requirement that the data can always be read in via pandas `pd.read_csv(PATH, delim_whitespace=True)`. This makes post-processing quite easy (at least reading the data ;-)

Modules

Many other analyses are still possible with timer module

The remaining modules such as idle-governor, idle-cluster, frequency enable analyses in completely different areas - but crucial for power management analysis and optimization



One last sneak-peak: it shows the C-State choice of the Idle Governor (here menu, Intel based system) and evaluates the correctness of the decision.

Further analyses show the ideal C-states and also point out possible causes of the "wrong" decision

Thank You!

Questions?

Source Code and Project Home:

https://github.com/protocollabs/linux-kernel-perf (branch: perf-powerstat)

Post Processing Scripts:

https://github.com/hgn/perf-power-analyzer-post

The first engineering conference on sustainability in hardware & software

EcoCompute Conference 2024

April 25 - 26

Munich, Germany

https://www.eco-compute.io/

Legal Disclaimer

This presentation is for informational purposes only and is not intended as an endorsement or promotion of the products or brands mentioned herein. The trademarks, logos, and service marks (collectively the “Trademarks”) displayed in this presentation are registered and unregistered Trademarks of their respective owners.

No affiliation or endorsement by the trademark owners is intended or should be inferred. The use of these Trademarks in this presentation does not imply any affiliation with or endorsement by their respective owners.

The views and opinions expressed in this presentation are those of the presenter and do not necessarily reflect the official policy or position of the brands mentioned. This presentation is not sponsored, endorsed by, or associated with any of the brands whose products are mentioned.

All information in this presentation is provided on an “as is” basis with no guarantees of completeness, accuracy, usefulness, or timeliness, and without any warranties of any kind whatsoever, express or implied.