TECHNISCHE UNIVERSITÄT DRESDEN

HAEC

DFG

READEX
Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing

DRESDEN concept

Powernightmares: The Challenge of Efficiently Using Sleep States on Multi-Core Systems

Thomas Ilsche, Marcus Hähnel, Robert Schöne, Mario Bielert, and Daniel Hackenberg

Technische Universität Dresden

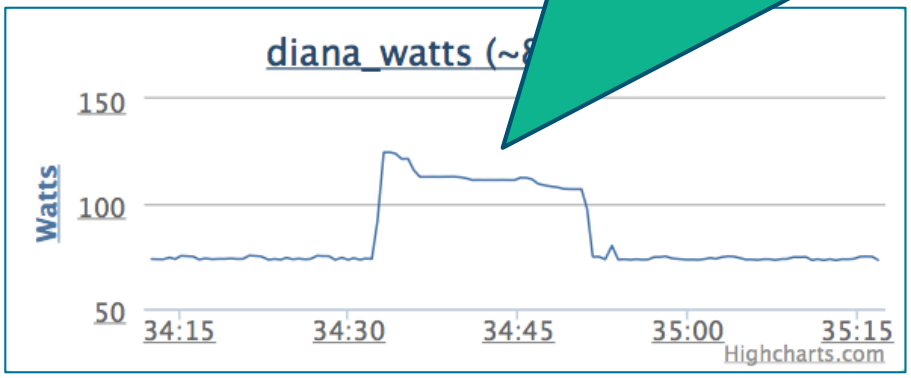
29.08.17

5th Workshop on Runtime and Operating Systems for the Many-core Era

Observation

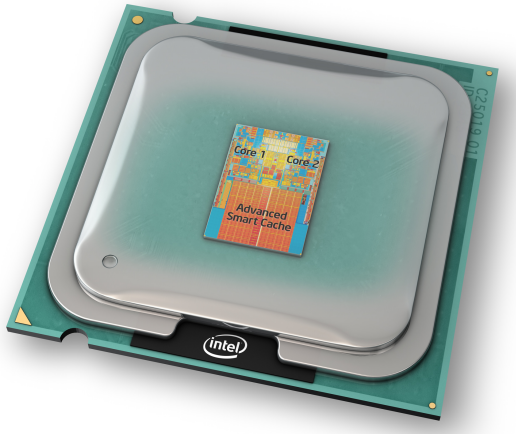
- Systems with continuous energy measurement
- Tuned for low idle power consumption
- Prolonge
  consump

"Powernightmare"



Background – Processor

- Each processor is a package
- A package comprises multiple cores
- Each core has two hardware threads
- A hardware thread is called CPU



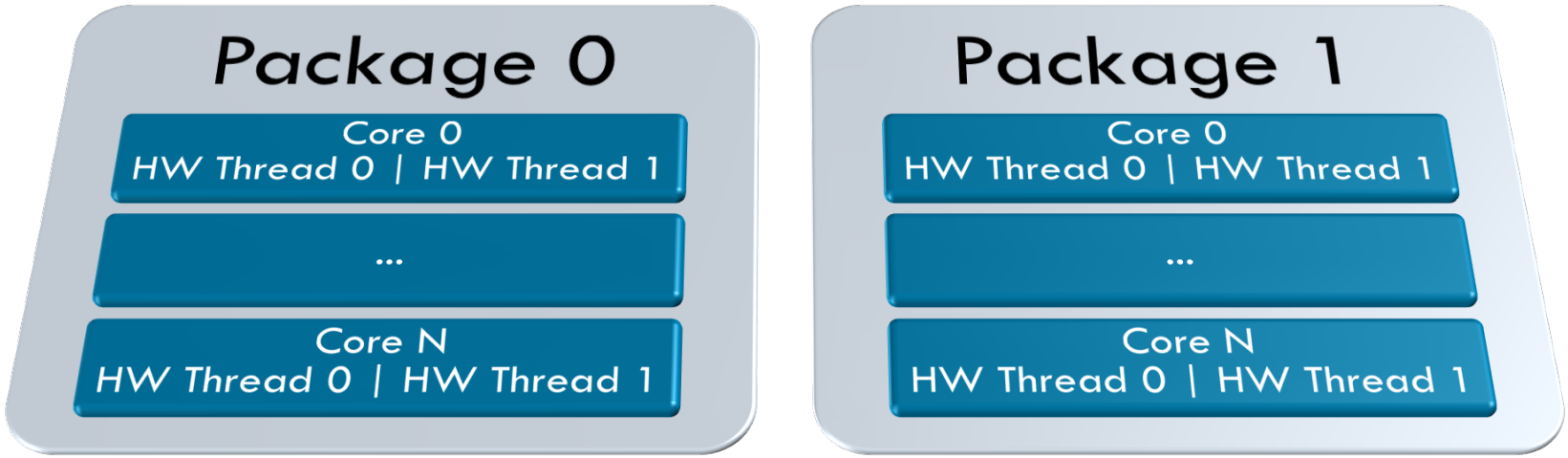
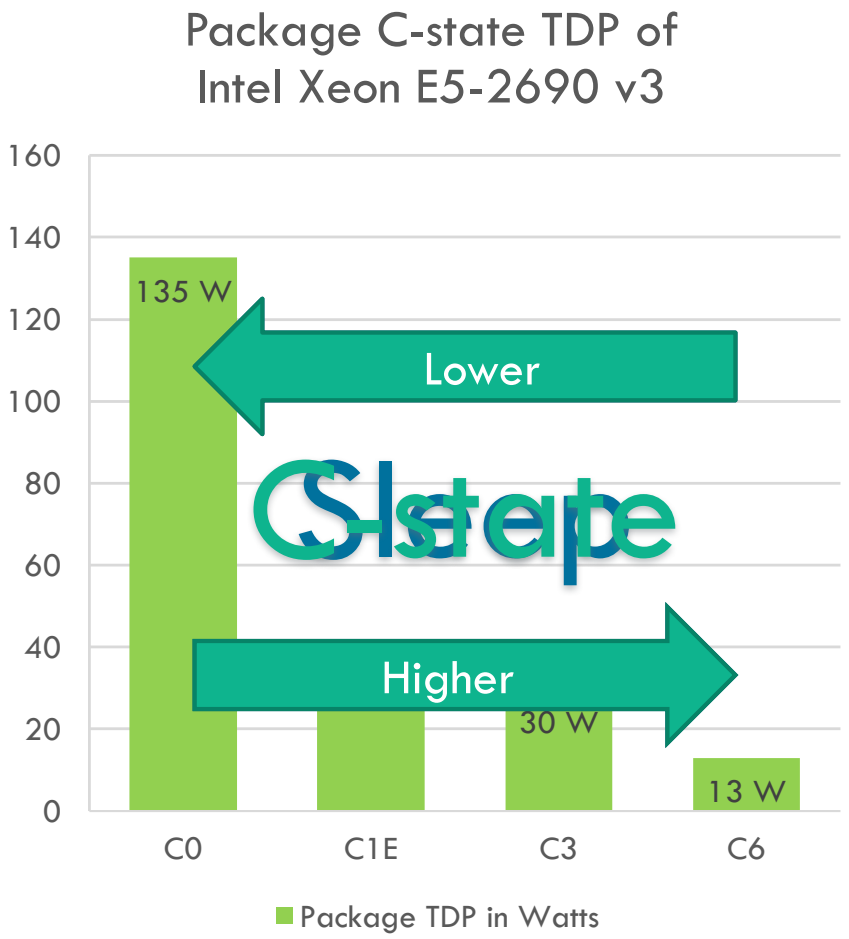


Image source: http://download.intel.com/pressroom/kits/45nm/penryn_dualcore_txt.jpg

Background – C-states

- Idle power conservation
- Increasing latency
- Controllable per CPU, but applied per core
- Package C-state determined by lowest core C-state
- Effective use is essential for low idle power



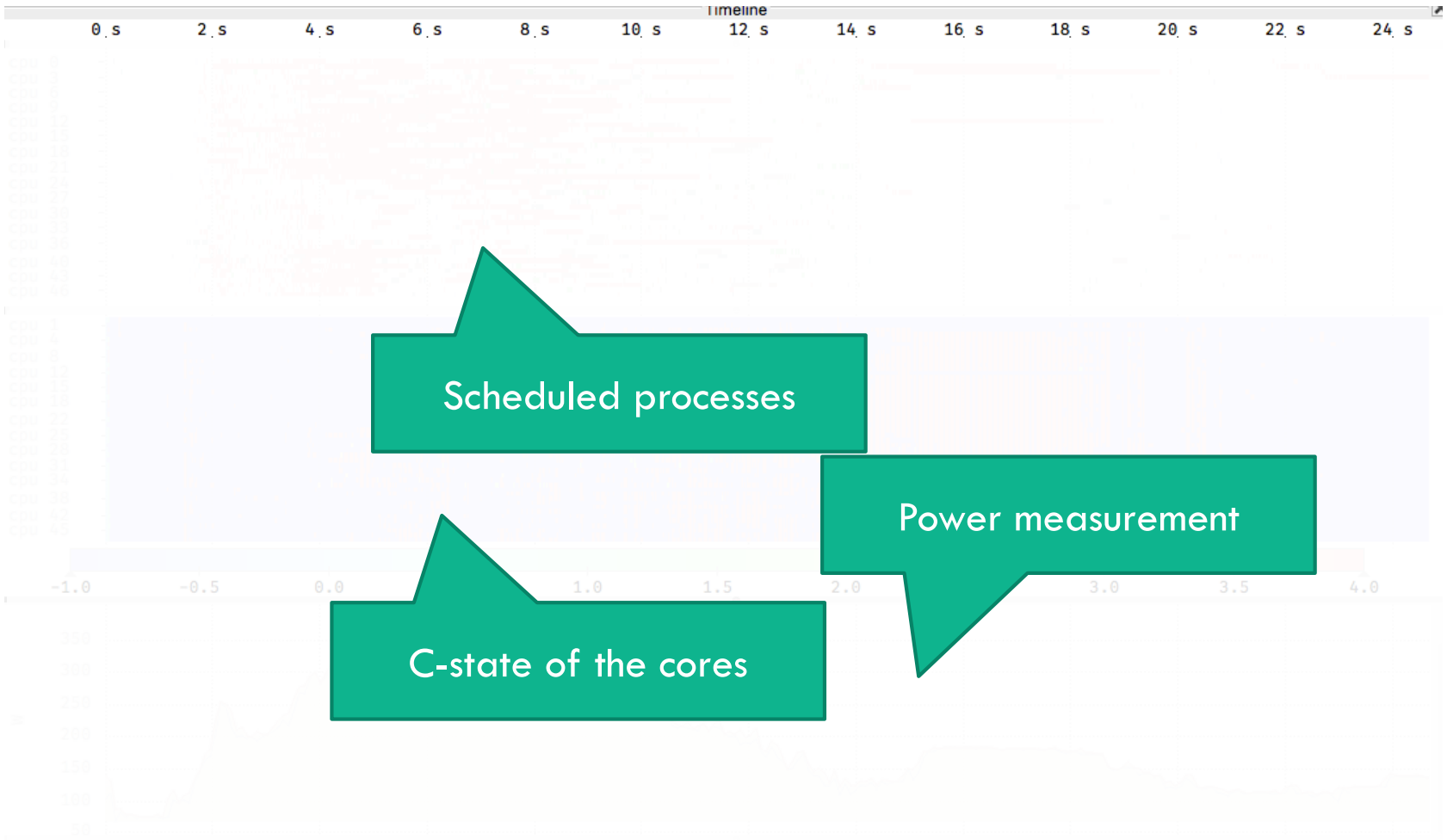
Background – Linux idle governor

- Selects C-state for CPU
- `ladder_governor` gradually changes C-state
- `menu_governor` is based on a heuristic
- Heuristic used to predict idle time
  - Next timer event with correction factor
  - Repeatable interval detector (up to 8 data points)
  - Latency requirement

Investigation – lo2s

- Uses Linux' perf infrastructure
- Create a trace combining
  - Active processes using the trace point sched_switch
  - Selected C-state using the cpu_idle trace point
  - External power measurements
  - C-state residency using x86_adapt
- Available at https://github.com/tud-zih-energy/lo2s

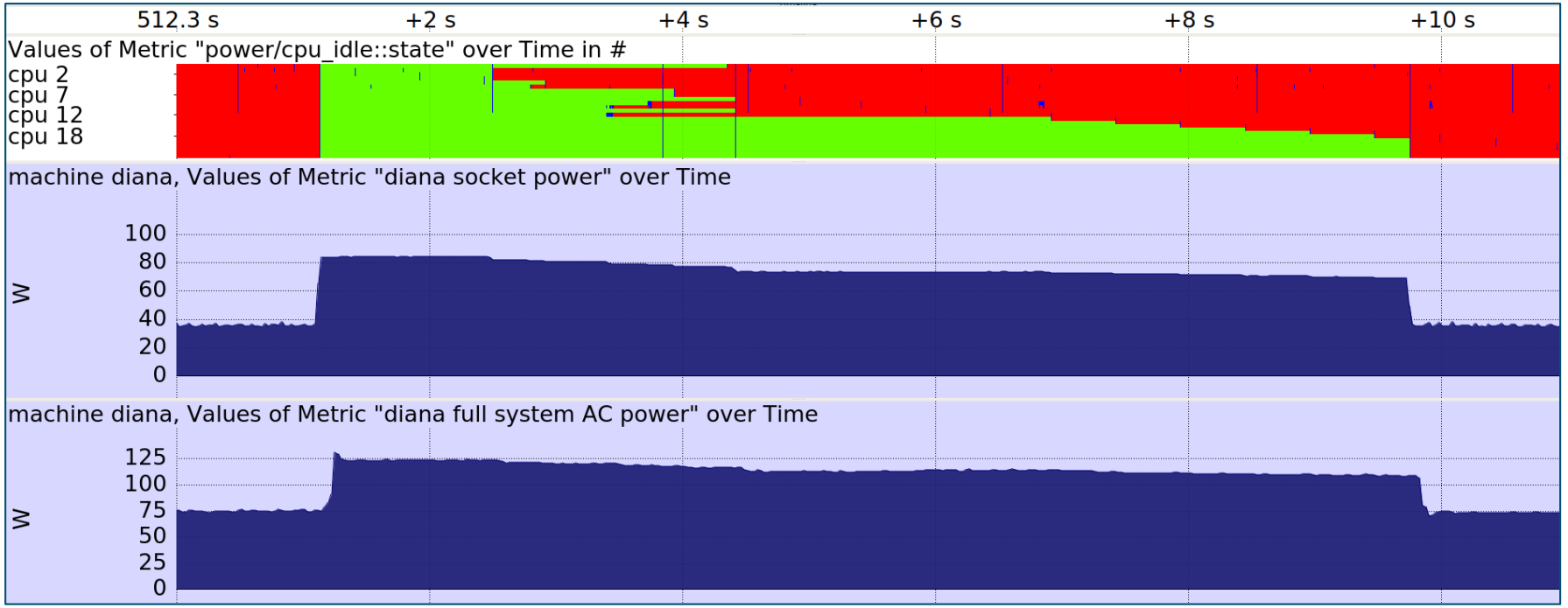
Investigation – lo2s



Vampir showing a lo2s trace of a parallel build using make

Investigation – Powernightmare

- Up to 3 wakeups needed for correction after a misprediction by the heuristic



Zoomed in portion of Powernightmare

Full begin of Powernightmare

Scheduler tasks, C-states and socket power

Triggering the issue

- Code to reliably trigger a Powernightmare

```
int main() {
#pragma omp parallel
{
    #pragma omp barrier
    while (1) {
        for (int i = 0; i < 8; i++) {
            #pragma omp barrier
            usleep (10);
        }
        sleep (10);
    }
}
}
```

Approaching the problem

- Changing task behavior
- Improving the idle time prediction
- Biasing the prediction error
- C-state selection by hardware
- Mitigating the impact

Impact mitigation approach

- Set a wakeup timer if huge difference between next known timer and predicted idle time

Prediction correct

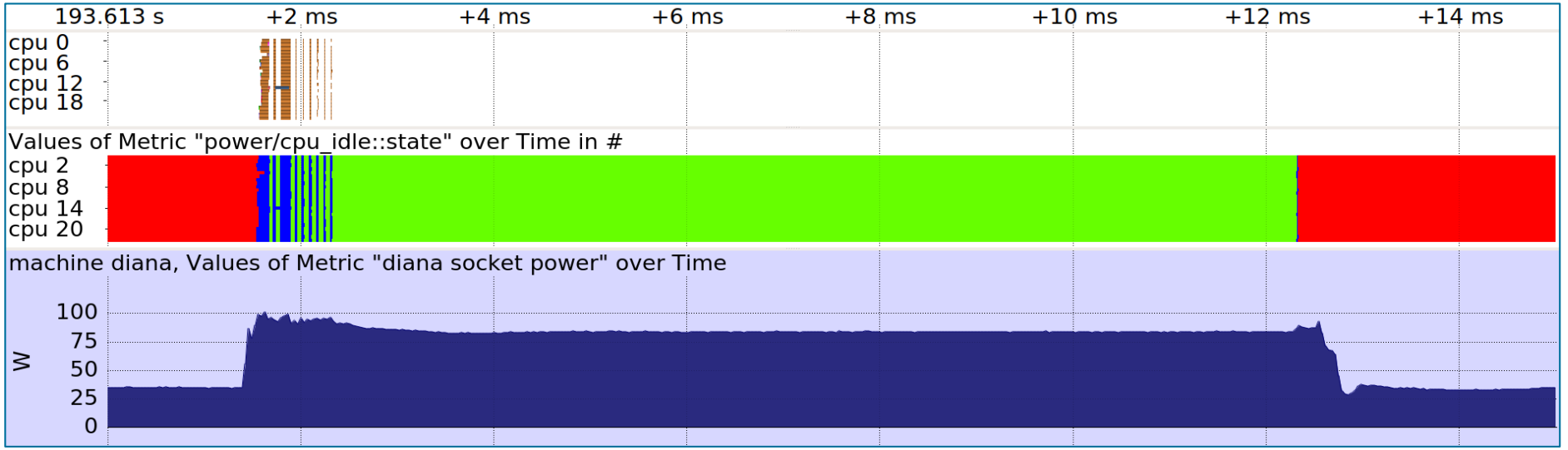
- Wakeup event in predicted time interval
- Cancel timer

Prediction incorrect

- Timer triggers wakeup
- Ignore recent residency
- Enter high C-state
- ✓ Misprediction corrected

Powernightmare with timer

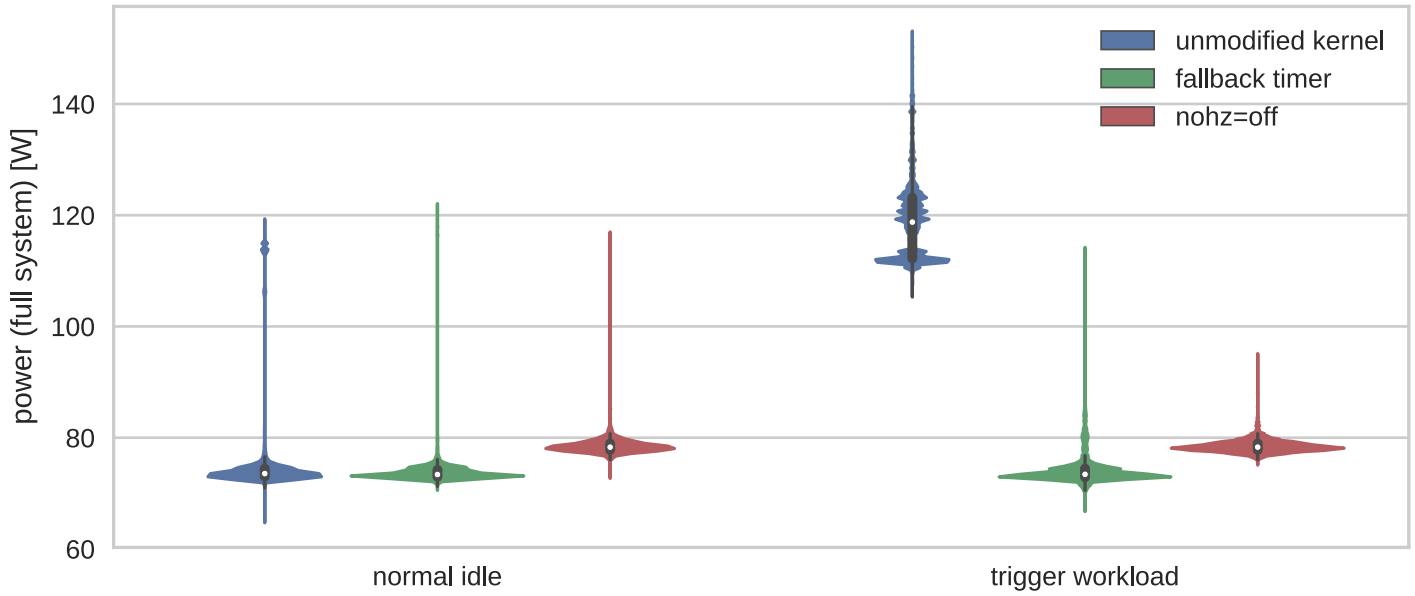
- Fallback timer corrects wrong C-state selection
- Only 10 ms of shallow sleep



Reduced impact of Powernightmare with active fallback timer

Verification

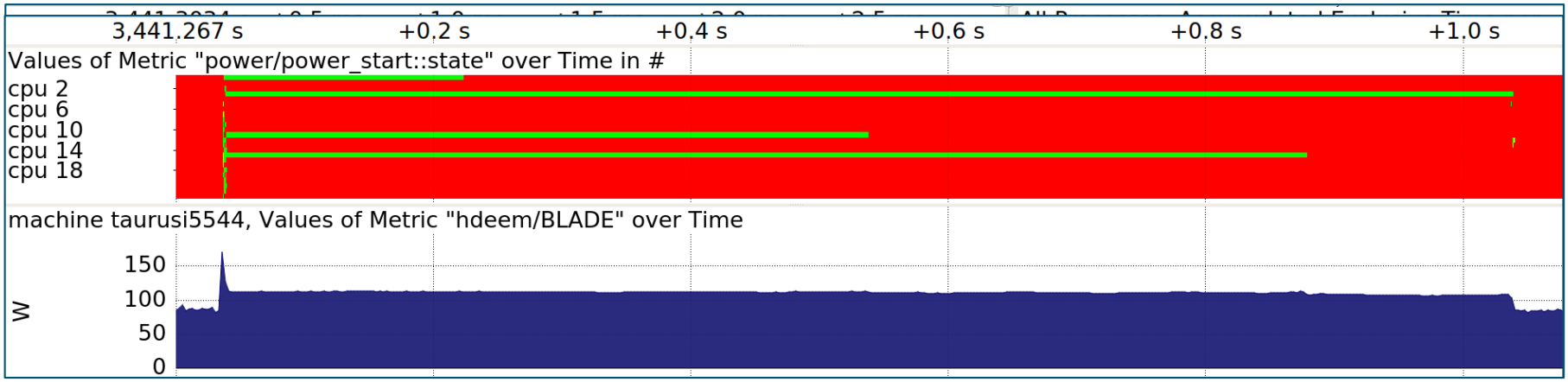
- Measurements taken over 20 minutes
- Trigger workload every 10 seconds



Power consumption during idle and trigger workload

Production servers?

- Found on node of production HPC system “taurus”
- Lustre related pattern every 25 seconds
- Triggers one second Powernightmare



Scheduling of user and kernel tasks

Summary

- Analyzed pattern of inefficient use of sleep states
- Developed a methodology and tools to observe
- Investigation shows misprediction in idle governor
- Proposed solution to mitigate effect
- Discussion with Linux community initiated
- Increasing probability with rising number of cores
- Effect not limited to HPC Systems


Any questions?