OS Scheduling with Nest: Keeping Tasks Close Together on Warm Cores

Julia Lawall, Himadri Chhaya-Shailesh (Inria), Jean-Pierre Lozi (Oracle Labs),
Baptiste Lepers, Willy Zwaenepoel (University of Sydney), Gilles Muller (Inria)

September 12, 2022

Per-core task scheduling in Linux

The goal of a task scheduler:

- Place tasks on cores on fork, wakeup, or load balancing.
- Choose a task on the core to run when the core becomes idle.

Per-core task scheduling in Linux

The goal of a task scheduler:

- Place tasks on cores on fork, wakeup, or load balancing.
- Choose a task on the core to run when the core becomes idle.

The challenge:

- Task placement that synergizes with hardware features.

A hardware feature: A core does one thing at a time

Work conservation: If a core is **overloaded**, no other core should be **idle**.

Studied in:

- The Linux scheduler: a decade of wasted cores. EuroSys 2016.
- Provable multicore schedulers with Ipanema: application to work conservation. EuroSys 2020.

Work conservation example

The machine

core 0	core 1	core 2	core 3
T1		T2	

Work conservation example

The machine

core 0	core 1	core 2	core 3
T1		T2	

Where to put waking task **T3**?

- According to **work conservation**, core 1 or core 3 is a better choice.

Another hardware feature: Dynamic Voltage and Frequency Scaling

With DVFS, cores can run at different frequencies:

- Higher frequency $\longrightarrow$
  - faster execution
  - more energy consuption
  - more heat generation
- Lower frequency $\longrightarrow$
  - slower execution
  - less energy consuption
  - less heat generation.

Another hardware feature: Dynamic Voltage and Frequency Scaling

With DVFS, cores can run at different frequencies:

- Higher frequency $\longrightarrow$
  - faster execution
  - more energy consuption
  - more heat generation
- Lower frequency $\longrightarrow$
  - slower execution
  - less energy consuption
  - less heat generation.

Principles (Intel, AMD servers):

- **More activity** on a core results in a higher frequency
  - Requests from the software (OS) and heuristics of the hardware.

Another hardware feature: Dynamic Voltage and Frequency Scaling

With DVFS, cores can run at different frequencies:

- Higher frequency $\longrightarrow$
  - faster execution
  - more energy consuption
  - more heat generation
- Lower frequency $\longrightarrow$
  - slower execution
  - less energy consuption
  - less heat generation.

Principles (Intel, AMD servers):

- **More activity** on a core results in a higher frequency
  - Requests from the software (OS) and heuristics of the hardware.
- **Turbo frequencies:** Fewer used cores allows highest frequencies, due to thermal constraints

What should be the impact of DVFS on scheduling?

core 0	core 1	core 2	core 3
T1		T2	

Where to put waking task **T3**?

What should be the impact of DVFS on scheduling?

	core 0	core 1	core 2	core 3
recent	T1	T0	T2	
current	T1		T2	

Where to put waking task **T3**?

What should be the impact of DVFS on scheduling?

	core 0	core 1	core 2	core 3
recent	T1	T0	T2	
current	T1		T2	

Where to put waking task **T3**?

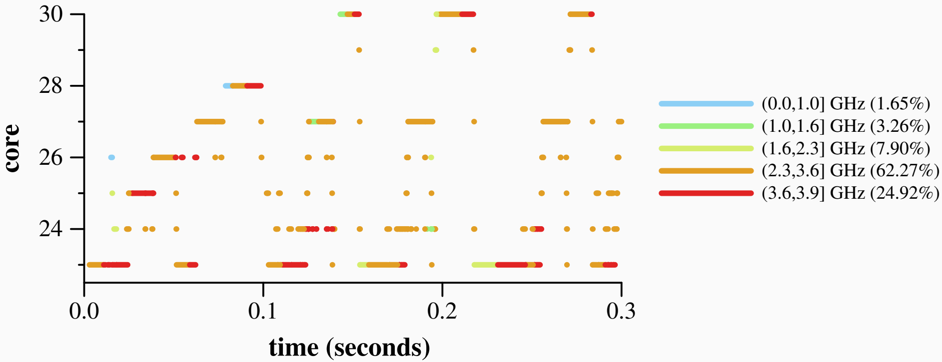
- **Core 1** could be a better choice:
  - Core 1 was recently active, so at a higher frequency.
  - Core 3 would suggest there are 4 active cores, giving a lower turbo frequency.

Nest scheduler

Goal: Task placement to exploit core frequencies.

- **Reuse cores:**
  - Maintain a **nest** of recently used cores.
- **Keep cores warm:**
  - Cores in the nest are likely to be reused,
    so spin briefly when they go idle, to keep the frequency high.

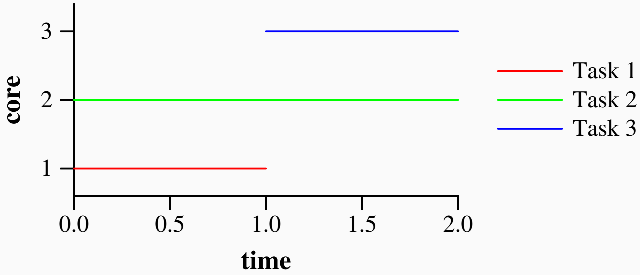
Task placement with Linux v5.9's CFS (configuration of LLVM)



2-socket 64-core Intel 5218

Task placement with Linux v5.9's CFS (configuration of LLVM), another perspective

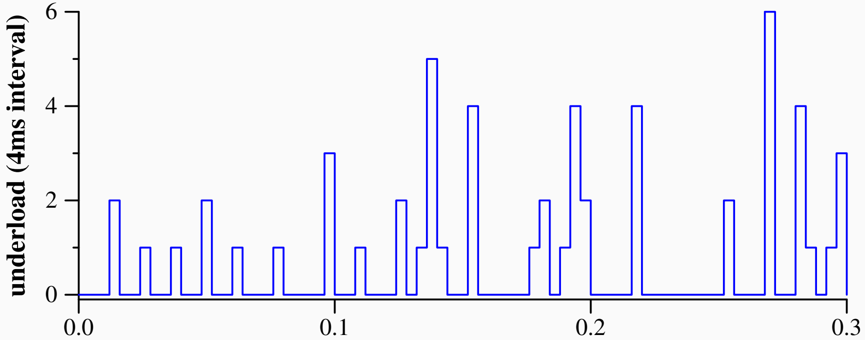
Underload: In an interval, the number of cores used beyond the degree of concurrency.



3 cores used, but only 2 needed $\Longrightarrow$ Underload of 1.

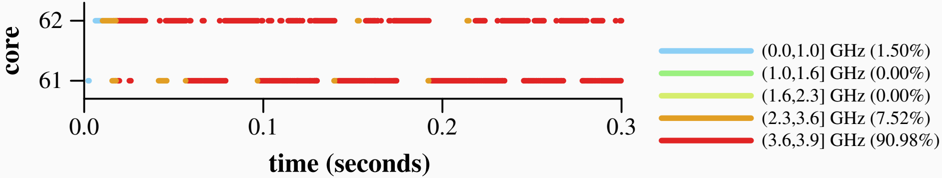
Task placement with Linux v5.9's CFS (configuration of LLVM), another perspective

Underload: In an interval, the number of cores used beyond the degree of concurrency.



2-socket 64-core Intel 5218

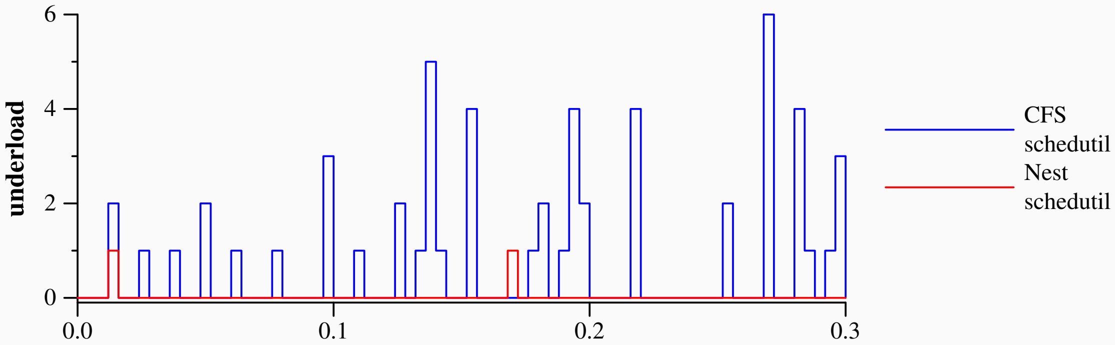
Task placement with Nest (configuration of LLVM)



2-socket 64-core Intel 5218

16% speedup overall

Underload: Nest vs CFS



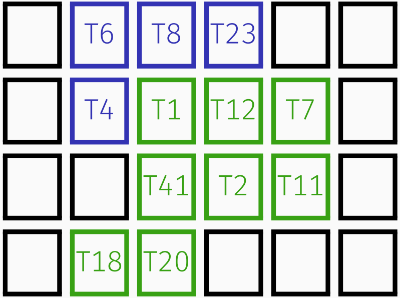
2-socket 64-core Intel 5218

16% speedup overall

Nest details: Reuse cores

Primary nest:

- Cores that are currently/recently used, and
- Expected to be useful in the near future.



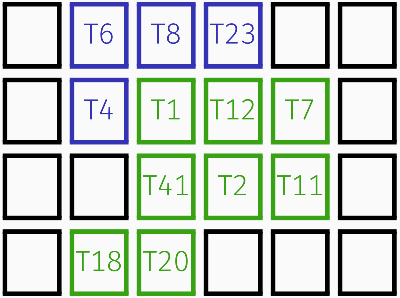
Nest details: Reuse cores

Primary nest:

- Cores that are currently/recently used, and
- Expected to be useful in the near future.

Reserve nest:

- Cores that were previously in the primary nest, but not used in a while, or
- Selected recently by CFS, but not yet deemed necessary in the primary nest.



Nest details: Reuse cores

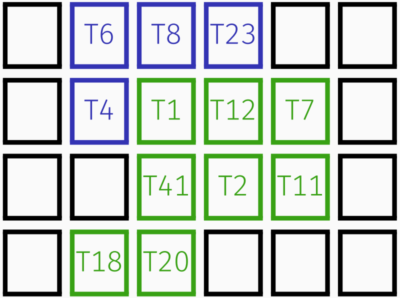
Primary nest:

- Cores that are currently/recently used, and
- Expected to be useful in the near future.

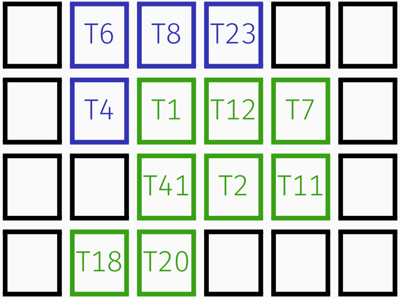
Reserve nest:

- Cores that were previously in the primary nest, but not used in a while, or
- Selected recently by CFS, but not yet deemed necessary in the primary nest.

The nests grow (and shrink, for the primary nest) automatically.



Nest details: Core selection example

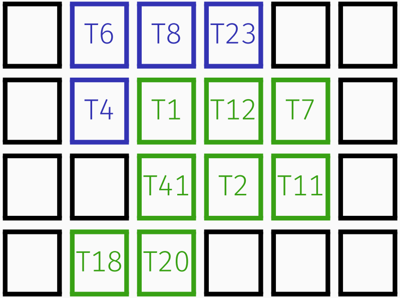


Task fork/wakeup: Task T80.

primary nest

reserve nest

Nest details: Core selection example



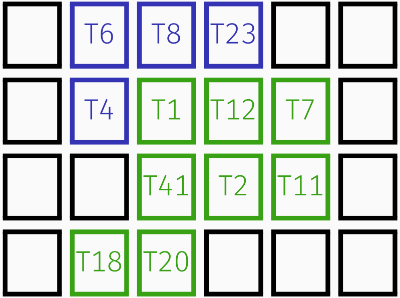
primary nest

reserve nest

Task fork/wakeup: Task T80.

- Look for an idle core in the primary nest:
  - Start at the parent/previous core to avoid collisions.

Nest details: Core selection example



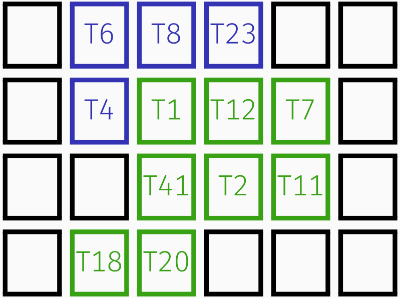
primary nest

reserve nest

Task fork/wakeup: Task T80.

- Look for an idle core in the primary nest: ✗
  - Start at the parent/previous core to avoid collisions.

Nest details: Core selection example



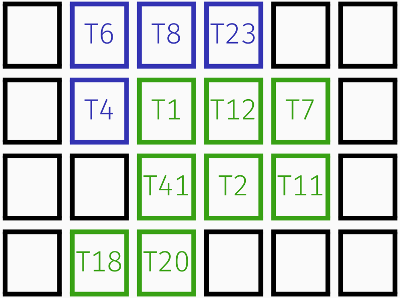
primary nest

reserve nest

Task fork/wakeup: Task T80.

- Look for an idle core in the primary nest: ✗
  - Start at the parent/previous core to avoid collisions.
- Look for an idle core in the reserve nest:
  - Always start at the same core, to avoid task dispersal.

Nest details: Core selection example



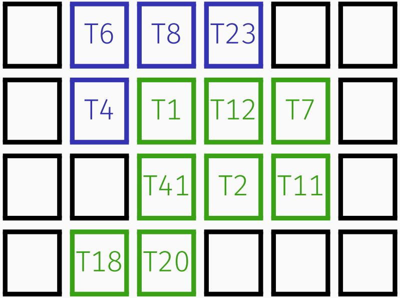
primary nest

reserve nest

Task fork/wakeup: Task T80.

- Look for an idle core in the primary nest: ✗
  - Start at the parent/previous core to avoid collisions.
- Look for an idle core in the reserve nest: ✗
  - Always start at the same core, to avoid task dispersal.

Nest details: Core selection example



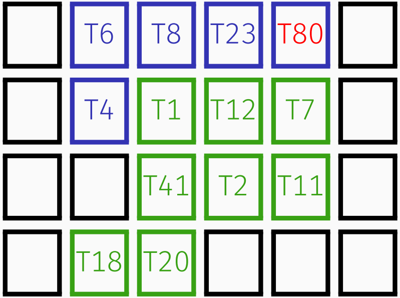
primary nest

reserve nest

Task fork/wakeup: Task T80.

- Look for an idle core in the primary nest: ✗
  - Start at the parent/previous core to avoid collisions.
- Look for an idle core in the reserve nest: ✗
  - Always start at the same core, to avoid task dispersal.
- In both cases, look in the parent/previous socket first, to improve locality.

Nest details: Core selection example



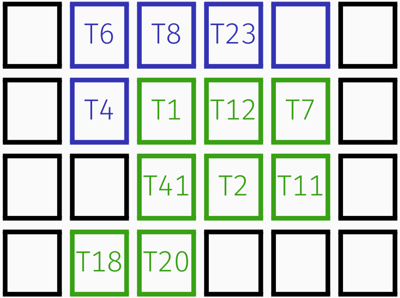
primary nest

reserve nest

Task fork/wakeup: Task T80.

- Look for an idle core in the primary nest:
  - Start at the parent/previous core to avoid collisions.
- Look for an idle core in the reserve nest:
  - Always start at the same core, to avoid task dispersal.
- In both cases, look in the parent/previous socket first, to improve locality.
- Finally, fall back to core picked by CFS; add it to the reserve nest, useful for transient tasks.

Nest details: nest management

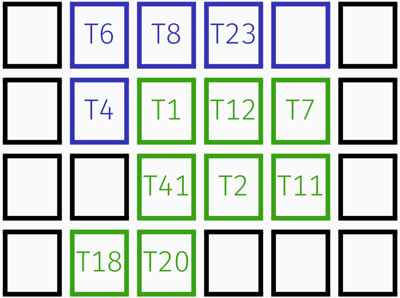


Another task fork/wakeup:

primary nest

reserve nest

Nest details: nest management



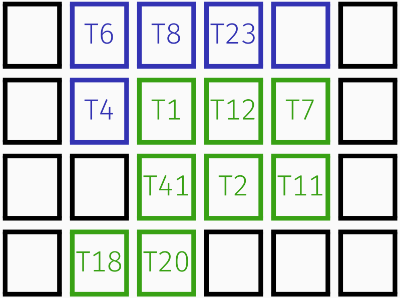
primary nest

reserve nest

Another task fork/wakeup:

- Look for an idle core in the primary nest:

Nest details: nest management



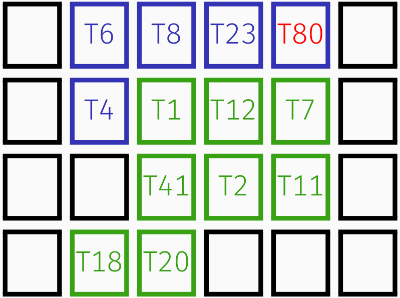
primary nest

reserve nest

Another task fork/wakeup:

- Look for an idle core in the primary nest: ✗

Nest details: nest management



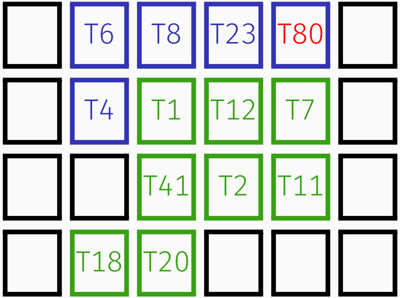
primary nest

reserve nest

Another task fork/wakeup:

- Look for an idle core in the primary nest: ✗
- Look for an idle core in the reserve nest:

Nest details: nest management



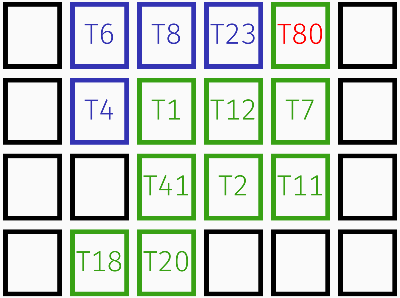
primary nest

reserve nest

Another task fork/wakeup:

- Look for an idle core in the primary nest: ✗
- Look for an idle core in the reserve nest: ✓

Nest details: nest management



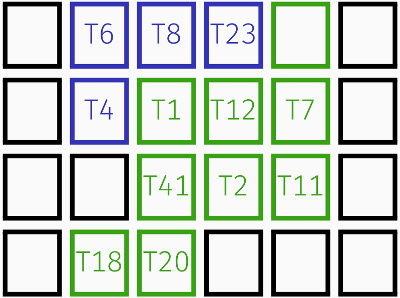
primary nest

reserve nest

Another task fork/wakeup:

- Look for an idle core in the primary nest: ✗
- Look for an idle core in the reserve nest: ✓
- The core enters the primary nest.

Nest details: nest management

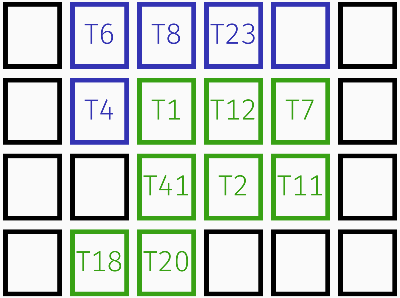


primary nest

reserve nest

Idle core in the primary nest:

Nest details: nest management



primary nest

reserve nest

Idle core in the primary nest:

- Moved back to the reserve nest:
  - Instantly on termination.
  - After some time, after a block.

Nest details: Keep cores warm

When a Nest task leaves a core, spin for a couple ticks:

- Long enough to keep the frequency high for the next task.
- Not too long to interfere with the turbo frequency choice

Nest details: Other issues

Attached cores:

- Task becomes attached to a core where it has run more than once, and tries to return there (previous-core history of depth 2)
- Mitigates the need to move in case of conflict with a kernel thread.

Impatient tasks:

- A thread that finds its previous core successively occupied falls back directly to CFS, as the nests are considered to be too small.

Nest details: Other issues

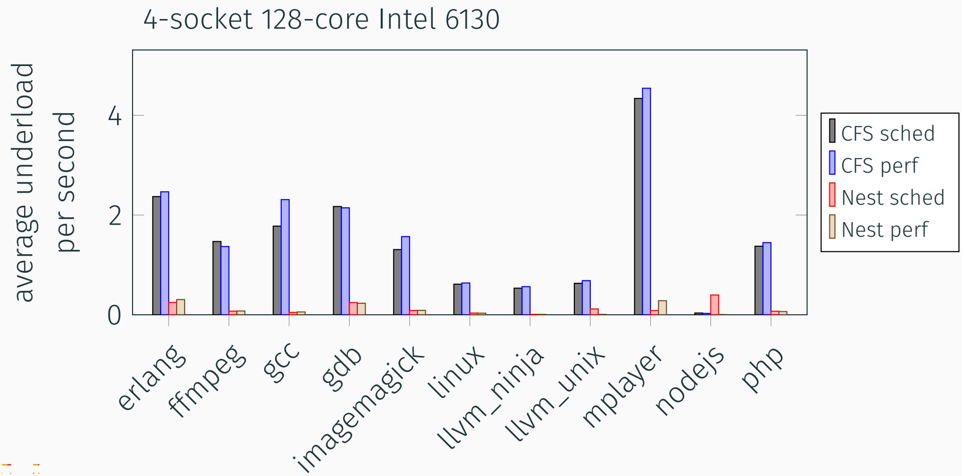
Wakeup work conservation:

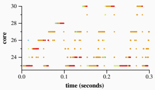
- Spreads tasks quickly across cores.
- Improves the accuracy of the nest size.

CAS to claim a core:

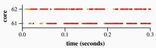
- Avoid collision on concurrent task placements.

Evaluation: Underload on software configuration



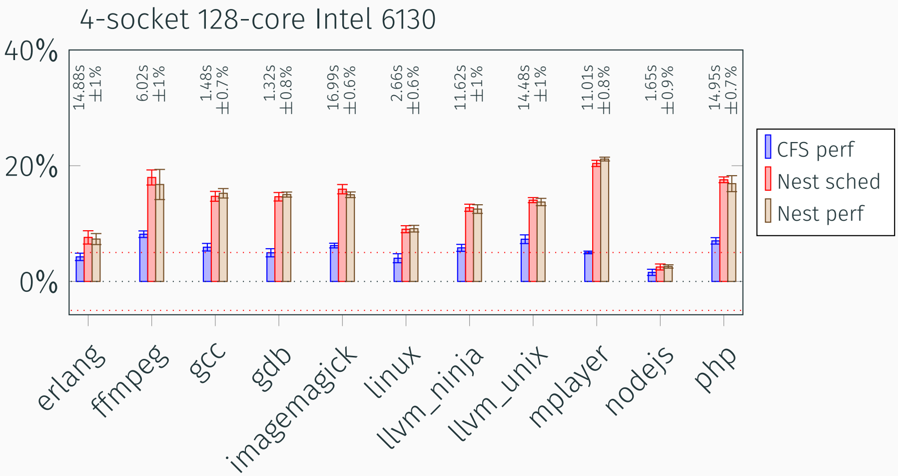


vs.



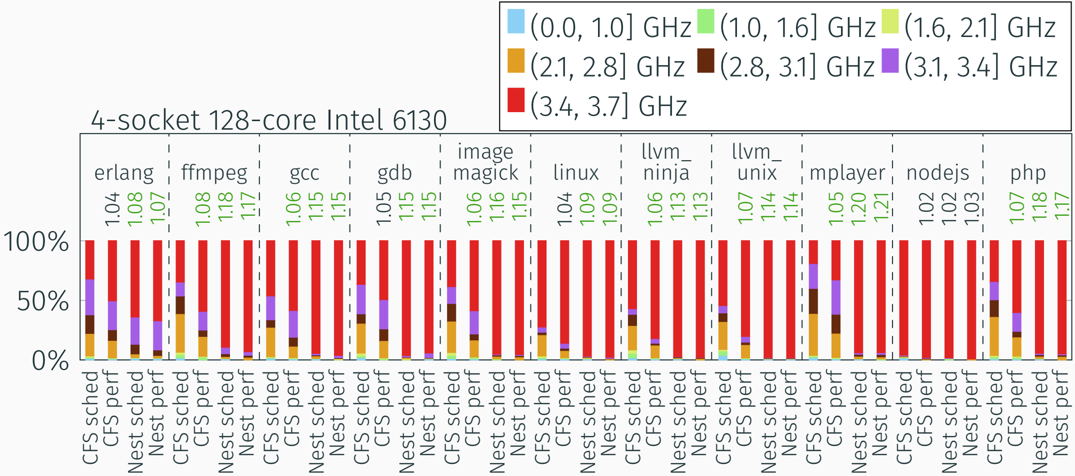
Linux v5.9

Evaluation: Performance improvement on software configuration



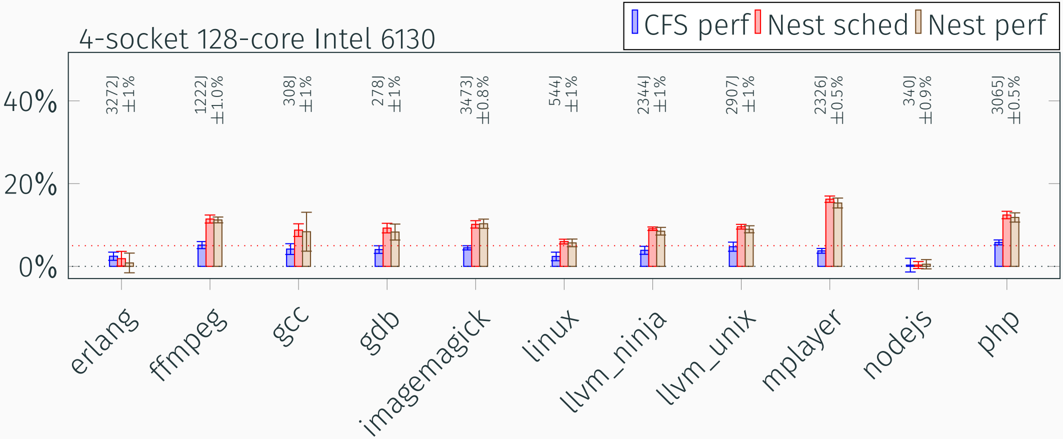
Linux v5.9

Evaluation: Core frequencies on software configuration



Linux v5.9

Evaluation: Energy consumption on software configuration (Linux v5.9)



Evaluation: Performance improvement on the Phoronix multicore suite (Linux v5.9)

Comparison to CFS schedutil:

CPU	scheduler	slower by $> 20\%$	slower by (5,20]%	same	faster by (5,20]%	faster by $> 20\%$
4 socket 6130	CFS-perf.	2 (1%)	7 (3%)	190 (87%)	9 (4%)	10 (5%)
	Nest-sched.	1 (0%)	19 (9%)	159 (73%)	21 (10%)	18 (8%)

More recent Linux versions

Change in schedutil in Linux 5.11:

- Before 5.11:

  ```
  intel_cpufreq_update_pstate(policy, target_pstate, true);
  ```

  Suggests a frequency.

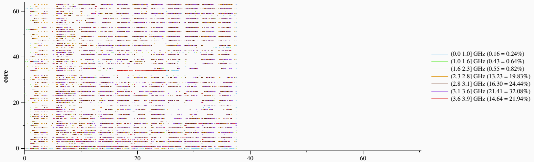
- Since 5.11:

  ```
  intel_cpufreq_hwp_update(cpu, min_pstate, max_pstate,
                           target_pstate, true);
  ```

  Imposes a frequency.

Impact on Nest

Baseline: Linux 5.9, with CFS (Intel 5128).

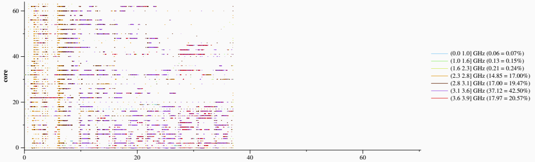


h2 troll-1 5.9 performance, duration: 37.48 seconds

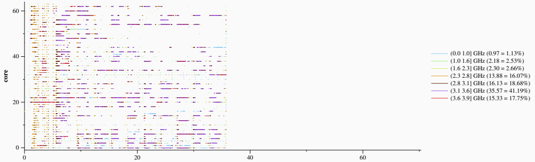
h2 troll-1 5.9 schedutil, duration: 43.58 seconds

Impact on Nest

Nest based on Linux 5.9 (Intel 5128).



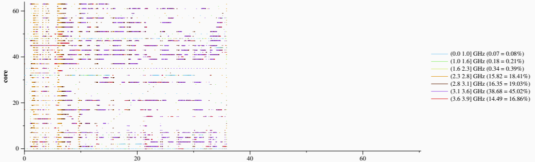
h2 troll-1 5.9Nest performance, duration: 36.95 seconds



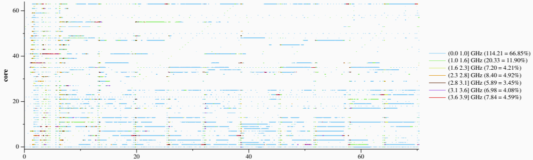
h2 troll-1 5.9Nest schedutil, duration: 35.71 seconds

Impact on Nest

Nest based on Linux 5.15 (Intel 5128).



h2 troll-1 5.15Nest performance, duration: 35.81 seconds



h2 troll-1 5.15Nest schedutil, duration: 70.30 seconds

Nest collects threads on cores,
but the frequency doesn’t rise with schedutil.

Conclusion

Nest task scheduler:

- Reuse cores.
  - Fewer used cores and increased utilization, leading to higher frequencies.
- Keep cores warm.
  - Maintain high frequencies over short idle periods.

Conclusion

Nest task scheduler:

- Reuse cores.
  - Fewer used cores and increased utilization, leading to higher frequencies.
- Keep cores warm.
  - Maintain high frequencies over short idle periods.

Performance impact (Linux v5.9):

- +10%–2× performance on light or moderate loads, on 1, 2, and 4 socket Intel servers (also an AMD desktop and an AMD server).
- Maintains performance for full loads and overloads (NAS benchmarks, some Phoronixes).
- Impact depends on the power management of the OS and target hardware.

Conclusion

Nest task scheduler:

- Reuse cores.
  - Fewer used cores and increased utilization, leading to higher frequencies.
- Keep cores warm.
  - Maintain high frequencies over short idle periods.

Performance impact (Linux v5.9):

- $+10\%$–$2\times$ performance on light or moderate loads, on 1, 2, and 4 socket Intel servers (also an AMD desktop and an AMD server).
- Maintains performance for full loads and overloads (NAS benchmarks, some Phoronixes).
- Impact depends on the power management of the OS and target hardware.

https://gitlab.inria.fr/nest-public/nest-artifact.git