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:

## Work conservation example

### The machine

<table>
<thead>
<tr>
<th>core 0</th>
<th>core 1</th>
<th>core 2</th>
<th>core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td>T2</td>
<td></td>
</tr>
</tbody>
</table>

Where to put waking task T3?

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

The machine

<table>
<thead>
<tr>
<th>core 0</th>
<th>core 1</th>
<th>core 2</th>
<th>core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td>T2</td>
<td></td>
</tr>
</tbody>
</table>

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 $\rightarrow$
  - faster execution
  - more energy consumption
  - more heat generation

- Lower frequency $\rightarrow$
  - slower execution
  - less energy consumption
  - less heat generation.
Another hardware feature: Dynamic Voltage and Frequency Scaling

With DVFS, cores can run at different frequencies:

- **Higher frequency** →
  - faster execution
  - more energy consumption
  - more heat generation

- **Lower frequency** →
  - slower execution
  - less energy consumption
  - 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 →
  - faster execution
  - more energy consumption
  - more heat generation

- Lower frequency →
  - slower execution
  - less energy consumption
  - 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?

Where to put waking task T3?

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T0</td>
<td>T2</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td>T2</td>
<td></td>
</tr>
</tbody>
</table>
What should be the impact of DVFS on scheduling?

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

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

- 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.
Task fork/wakeup: Task T80.

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

primary nest

reserve nest
Nest details: Core selection example

Task fork/wakeup: Task T80.

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

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

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

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.
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.
Another task fork/wakeup:

- Look for an idle core in the primary nest:

primary nest
rescue nest
Another task fork/wakeup:

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

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

Another task fork/wakeup:

- Look for an idle core in the primary nest: X
- Look for an idle core in the reserve nest: ✓
Nest details: nest management

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.

primary nest
reserve nest
Nest details: nest management

Idle core in the primary nest:

- T6
- T8
- T23
- T4
- T1
- T12
- T7
- T41
- T2
- T11
- T18
- T20

primary nest
reserve nest
Nest details: nest management

Idle core in the primary nest:

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

4-socket 128-core Intel 6130

average underload per second

Linux v5.9
Evaluation: Performance improvement on software configuration

4-socket 128-core Intel 6130

<table>
<thead>
<tr>
<th>Software</th>
<th>CFS perf</th>
<th>Nest sched</th>
<th>Nest perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>erlang</td>
<td>14.88s ±1%</td>
<td>14.88s ±1%</td>
<td>14.88s ±1%</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>14.88s ±1%</td>
<td>14.88s ±1%</td>
<td>14.88s ±1%</td>
</tr>
<tr>
<td>gcc</td>
<td>16.99s ±1%</td>
<td>16.99s ±1%</td>
<td>16.99s ±1%</td>
</tr>
<tr>
<td>gdb</td>
<td>2.66s ±0.6%</td>
<td>2.66s ±0.6%</td>
<td>2.66s ±0.6%</td>
</tr>
<tr>
<td>imagemagick</td>
<td>11.62s ±1%</td>
<td>11.62s ±1%</td>
<td>11.62s ±1%</td>
</tr>
<tr>
<td>linux</td>
<td>1.48s ±0.7%</td>
<td>1.48s ±0.7%</td>
<td>1.48s ±0.7%</td>
</tr>
<tr>
<td>llvm_ninja</td>
<td>1.48s ±0.7%</td>
<td>1.48s ±0.7%</td>
<td>1.48s ±0.7%</td>
</tr>
<tr>
<td>llvm_unix</td>
<td>1.48s ±0.7%</td>
<td>1.48s ±0.7%</td>
<td>1.48s ±0.7%</td>
</tr>
<tr>
<td>mplayer</td>
<td>11.01s ±0.8%</td>
<td>11.01s ±0.8%</td>
<td>11.01s ±0.8%</td>
</tr>
<tr>
<td>nodejs</td>
<td>14.48s ±1%</td>
<td>14.48s ±1%</td>
<td>14.48s ±1%</td>
</tr>
<tr>
<td>php</td>
<td>1.32s ±0.8%</td>
<td>1.32s ±0.8%</td>
<td>1.32s ±0.8%</td>
</tr>
</tbody>
</table>

Linux v5.9
### Evaluation: Core frequencies on software configuration

<table>
<thead>
<tr>
<th>Software</th>
<th>0%</th>
<th>50%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>erlang</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ffmpeg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gdb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>image magick</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linux</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>llvm ninja</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>llvm unix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mplayer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodejs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>php</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 4-socket 128-core Intel 6130

<table>
<thead>
<tr>
<th>Core Frequency Range</th>
<th>CFS sched perf</th>
<th>Nest sched perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0, 1.0 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0, 1.6 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6, 2.1 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1, 2.8 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.8, 3.1 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1, 3.4 GHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4, 3.7 GHz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Linux v5.9

- 4-socket 128-core Intel 6130
- Core frequencies on software configuration

- Core frequencies plotted for various software applications:
  - Erlang
  - FFMPEG
  - GCC
  - GDB
  - Image Magick
  - Linux
  - LLVM Ninja
  - LLVM Unix
  - Mplayer
  - NodeJS
  - PHP

#### Core Frequency Ranges:

- 0.0, 1.0 GHz
- 1.0, 1.6 GHz
- 1.6, 2.1 GHz
- 2.1, 2.8 GHz
- 2.8, 3.1 GHz
- 3.1, 3.4 GHz
- 3.4, 3.7 GHz
Evaluation: Energy consumption on software configuration (Linux v5.9)

<table>
<thead>
<tr>
<th>Software</th>
<th>CFS perf</th>
<th>Nest sched</th>
<th>Nest perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>erlang</td>
<td>3272J ± 1%</td>
<td>1222J ± 1%</td>
<td>308J ± 1%</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>278J ± 1%</td>
<td>2344J ± 1%</td>
<td>2907J ± 1%</td>
</tr>
<tr>
<td>gcc</td>
<td>3473J ± 0.8%</td>
<td>2344J ± 1%</td>
<td>2326J ± 0.3%</td>
</tr>
<tr>
<td>gdb</td>
<td>544J ± 1%</td>
<td>2907J ± 1%</td>
<td>340J ± 0.9%</td>
</tr>
<tr>
<td>imagemagick</td>
<td>278J ± 1%</td>
<td>2344J ± 1%</td>
<td>2344J ± 1%</td>
</tr>
<tr>
<td>linux</td>
<td>308J ± 1%</td>
<td>2344J ± 1%</td>
<td>2344J ± 1%</td>
</tr>
<tr>
<td>llvm_ninja</td>
<td>3473J ± 0.8%</td>
<td>2344J ± 1%</td>
<td>2344J ± 1%</td>
</tr>
<tr>
<td>llvm_unix</td>
<td>278J ± 1%</td>
<td>2344J ± 1%</td>
<td>2344J ± 1%</td>
</tr>
<tr>
<td>mplayer</td>
<td>2907J ± 1%</td>
<td>2344J ± 1%</td>
<td>2344J ± 1%</td>
</tr>
<tr>
<td>nodejs</td>
<td>308J ± 1%</td>
<td>2344J ± 1%</td>
<td>2344J ± 1%</td>
</tr>
<tr>
<td>php</td>
<td>278J ± 1%</td>
<td>2344J ± 1%</td>
<td>2344J ± 1%</td>
</tr>
</tbody>
</table>

4-socket 128-core Intel 6130
Comparison to CFS schedutil:

<table>
<thead>
<tr>
<th>CPU scheduler</th>
<th>Slower by &gt; 20% (5,20]%</th>
<th>Same (5,20]%</th>
<th>Faster by &gt; 20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 socket 6130 CFS-perf.</td>
<td>2 (1%)</td>
<td>190 (87%)</td>
<td>9 (4%)</td>
</tr>
<tr>
<td>NEST-sched.</td>
<td>1 (0%)</td>
<td>159 (73%)</td>
<td>21 (10%)</td>
</tr>
</tbody>
</table>


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).
Impact on Nest

Nest based on Linux 5.9 (Intel 5128).

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

Nest based on Linux 5.15 (Intel 5128).

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