A Cache-Stall Driven CPU Frequency Governor for Linux*

Erik Griese[1], Leon Matthes[1], Maximilian Stiede[1], Sven Köhler[2], Lukas Wenzel[2]
Hasso Plattner Institute, Potsdam
[1]{firstname.lastname}@student.hpi.de [2]{firstname.lastname}@hpi.de

ABSTRACT

Energy efficiency is a key concern in the operation of today's computer systems. An operating system needs to carefully control the CPU frequencies over several clock domains. The default Linux frequency governor *SchedUtil* [1] can be found to enforce the highest possible frequency for systems at full load. However, previous research has shown a strong relationship between data dependency and wasted energy [3], as ever faster clocked processing units tend to idle at high clock rates, while data is not yet available from the slower memory subsystem. Deep cache hierarchies and prefetchers mitigate these effects, but only for access patterns with sufficient locality and predictability.

To improve the energy efficiency, we propose StallGov, a Linux CPU frequency governor based on hardware performance measurement counters (PMCs). These counters are special purpose registers providing low-overhead runtime measurements of microarchitectural hardware events. StallGov estimates the data throughput of the cache hierarchy resulting from the access patterns of the currently running CPU process. In response, it makes dynamic voltage and frequency scaling decisions to minimize the energy consumption during cycles when the CPU experiences a memory stall. Earlier approaches to adapt the CPU clock speed for different workloads typically required *a priori* knowledge about the executed programs. We investigate which PMCs best serve as heuristics *at runtime* and evaluate the impact of our implementation on both energy demand and execution time.

Implementation. We wrote StallGov, a *CPUFreq* governor for Linux, tested with kernel version 5.15 and above on recent Intel and AMD CPUs. While our implementation is a self-contained module, we require a minor change to the vanilla Linux kernel: namely, the export of the `perf_read_local` function to allow access to hardware counters from anywhere in the kernel. For now, our prototype features a local copy of this function.

We provide *normal* (with an update thread) and *fast* frequency switching (without), with the prospect of *p-state switching*, although this would currently limit us to Intel CPUs at the time of writing. An *update hook*, called periodically by the kernel scheduler every 5 to 10 ms, polls the PMCs and uses a simple linear interpolation of the observed min/max values to scale the frequency. The advantage of linear scaling is that it allows for a very lightweight and simple implementation, compared to SchedUtil.

PMC Selection. To answer which hardware counters provide a heuristic for which tasks might benefit from a reduced clock speed, we first need to capture PMCs for later investigation and to determine the min/max values. To that end, we record PMC events at 5 ms resolution into a ring buffer, which is exposed through the Linux *debugfs* to and logged from userspace every 10 s.

On our three test systems (Intel Core 6700HQ, 10510U, and AMD Ryzen 9 5900X) we compared 12 combinations of 21 counters—captured with both, one core only, and all cores. The five NAS Parallel Benchmark suite kernels (NPB), version 3.4.1 with OpenMP, serve as workload. We found an increased core frequency no longer improves performance when the core waits for the shared L3 or RAM. Therefore, counters related to L2 cache misses provide a good distinction: the number of L2 stalls per cycle (only available on Intel) and the instructions per cycle (also available on AMD).

These findings are in line with other research, suggesting the same correlation [2]. We thus shift the categorization from compute-bound vs. memory-bound to *core*-bound vs. *uncore*-bound, as latter is conveniently a different clock domain on our test systems.

Evaluation. Using our StallGov prototype we compared the impact of our approach on energy (RAPL, `pkg-domain`) and total execution-time (wall-clock) with the default Linux frequency governor SchedUtil while running aforementioned NPB suite, with workload sizes chosen to take 60 seconds on our systems. Note, that StallGov does not choose a fixed frequency, but rather changes over the course of our experiments as the workload phases change. For a more complete picture of the possible headroom, we additionally ran each benchmark at a fixed frequency, ranging from the lowest to the highest supported CPU frequency (in steps of 100 MHz).

In total, we increase energy efficiency by 13.3 % compared to SchedUtil while being 6 % slower (geometric means, 3 runs per benchmark, additional warm-up). In terms of the Energy-Delay-Squared Product (ED^2P, a metric that counterbalances the quadratic influence of the voltage on the energy demand) StallGov outperforms SchedUtil for the `mg` and `ft` benchmarks (-18.9 % and -3.2 %), stays in the same range for `ep` and `is` (-0.1 % and +0.2 %), and worsens the ED^2P only for `cg` (+12.9 %), with $\sigma < 1$ % for all.

Future Work. We plan to investigate whether found clock-domain split at the L2 cache always serves as a good heuristic, or if other PMC combinations prove more suitable. At the same time, our simplistic linear interpolation can be extended. Better statistical methods and (e.g. statistical learning) may prove useful. In this context, we could incorporate usage statistics from SchedUtil.

In addition, we will evaluate the time overhead for each frequency switch, as well as the behavior with multiple, dynamically switching processes (and simultaneous multithreading), since we have only run single NPB workloads exclusively. However, it is reasonable to assume that our process-agnostic implementation yields good results as PMCs are collected per-core.

ACKNOWLEDGEMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 502228341 ("Memento") and a stipend granted by the IBM Research & Development Lab, Böblingen.

*This poster abstract was accepted at the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23), Poster Session.

REFERENCES

[1] Redha Gouicem, Damien Carver, Jean-Pierre Lozi, Julien Sopena, Baptiste Lepers, Willy Zwaenepoel, Nicolas Palix, Julia Lawall, and Gilles Muller. 2020. Fewer Cores, More Hertz: Leveraging High-Frequency Cores in the OS Scheduler for Improved Application Performance. In *Proc. of the 2020 USENIX Annual Technical Conference (USENIX ATC'20)*. USENIX Association, 435–448.

[2] Benedict Herzog, Stefan Reif, Fabian Hügel, Wolfgang Schröder-Preikschat, and Timo Hönig. 2022. Bears: Building Energy-Aware Reconfigurable Systems. In *Proc. of the XII Brazilian Symp. on Computing Systems Engineering (SBESC)*. 1–8.

[3] Andreas Weissel and Frank Bellosa. 2002. Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management. In *Proc. of the 2002 Int. Conference on Compilers, Architecture, and Synthesis for Embedded Systems* (Grenoble, France) *(CASES '02)*. New York, NY, USA, 238–246.

StallGov

A Cache-Stall Driven CPU Frequency Governor for Linux

Erik Griese, Leon Matthes, Maximilian Stiede, Sven Köhler, Lukas Wenzel

Motivation

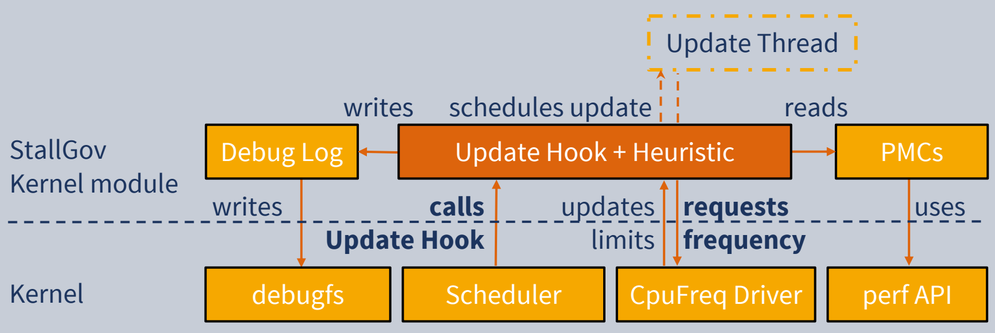
- For energy efficiency, processing units should not idle at high clock rates while waiting for data from the slower memory subsystem [3]
- Linux' default frequency governor SchedUtil often maximizes clock [1]
- Hardware performance counters can give insight on how much (+ where) the CPU is stalled and provide hints on a better clock rate and voltage
- Our implementation should work for unknown workloads *at run-time*

Implementation

- StallGov, a Linux kernel module (~1700 loc) using the CpuFreq API
- Implements *normal* and *fast* frequency switching
- Update hook is called regularly (5-10ms) by scheduler
- Queries hardware counters from (custom exported) kernel perf API
- Uses simple linear interpolation of observed min/max counter and allowed min/max clock frequency

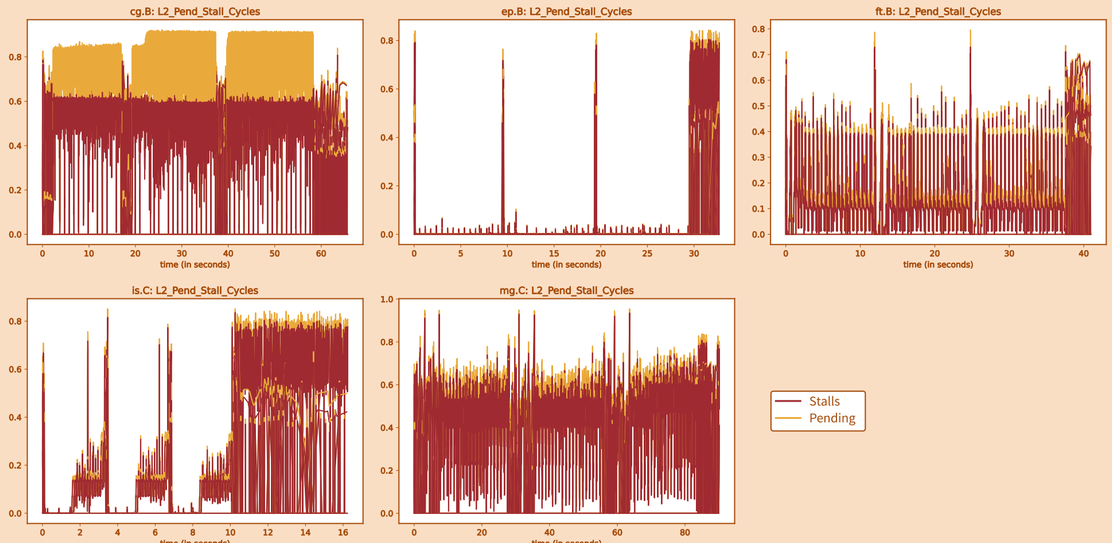
$$f_{current} = clamp\left(\frac{\beta_{current} - \beta_{min}}{\beta_{max} - \beta_{min}}, 0.0, 1.0\right) \cdot (f_{max} - f_{min}) + f_{min}$$

- Logs hardware counters in ring-buffer, exposed to userspace via debugfs



PMC Selection for Heuristic

- Compared 12 combinations of 21 hardware counters
- Recorded counters with threads pinned to single core, and run on every core
- Found that an increased core frequency no longer improves performance when the core waits for shared L3 cache or RAM
- Assumption: Core/uncore clock domain boundary is relevant feature
- Found two promising heuristics:
  - L2 cache stalls per cycle (only available on Intel)
  - Instructions per cycle (also available on AMD)
- Findings consistent with other research suggesting same correlation [2]
- Allows for *adaptive frequency selection* as workload changes phases



L2 Stalls per Cycle for the Five NPB Kernels

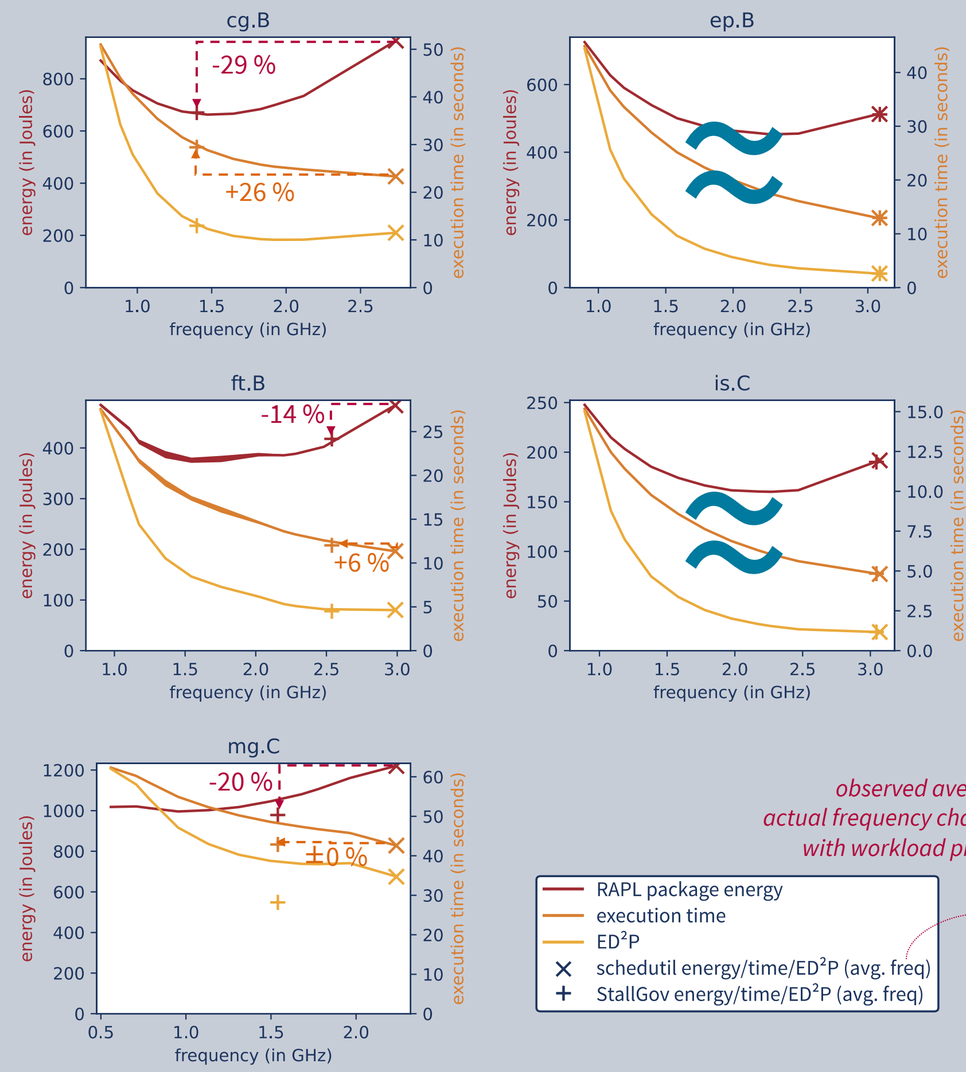
Test Workloads and Setup

- Using the NAS Parallel Benchmark suite, version 3.4.1 with OpenMP
- Evaluated on three machines, Linux 5.15+, problem sizes to run ~60s

Intel Core i7-6700HQ	Intel Core i7-10510U	AMD Ryzen 9 5900X
ACPI CpuFreq-driver	ACPI CpuFreq-driver	amd-pstate CpuFreq-driver
10ms update interval	10ms update interval	5ms update interval
0.8 - 2.6 (3.5 turbo) GHz	0.4 - 1.8 (4.9 turbo) GHz	0.55 - 3.7 (4.9 turbo) GHz
NAS {cg,fp,ft}.B, {is,mg}.C	NAS {cg,fp,ft}.B, {is,mg}.C	NAS {cg,fp,ft,is,mg}.C

Evaluation

- In total **13.3 % more energy-efficient** than SchedUtil, but **6 % slower**
- ED²P improvement for `mg`, `ft`, no changes with `ep`, `is`, worse for `cg`



Future Work

- Better Heuristics? Other PMCs? Non-linear interpolation?
- Include usage statistics from SchedUtil
- Investigate workloads other than the NPB suite
- Evaluate interference with multiple, dynamically switching processes and simultaneous multithreading

References

1. Redha Gouicem, Damien Carver, Jean-Pierre Lozi, Julien Sopena, Baptiste Lepers, Willy Zwaenepoel, Nicolas Palix, Julia Lawall, and Gilles Muller. 2020. Fewer Cores, More Hertz: Leveraging High-Frequency Cores in the OS Scheduler for Improved Application Performance. In *Proc. of the 2020 USENIX Annual Technical Conference (USENIX ATC'20).* USENIX Association, 435–448.
2. Benedict Herzog, Stefan Reif, Fabian Hügel, Wolfgang Schröder-Preikschat, and Timo Hönig. 2022. Bears: Building Energy-Aware Reconfigurable Systems. In *Proc. of the XII Brazilian Symp. on Computing Systems Engineering (SBESC)*. 1–8.
3. Andreas Weissel and Frank Bellosa. 2002. Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management. In *Proc. of the 2002 Int. Conference on Compilers, Architecture, and Synthesis for Embedded Systems* (Grenoble, France) *(CASES '02).* New York, NY, USA, 238–246.

https://github.com/osmhpi/stallgov

Hasso Plattner Institute, Potsdam
{firstname.lastname}@{student.}hpi.de

funded with a stipend granted by IBM Deutschland R&D Lab, Böblingen
SPP 2377
funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 502228341
Hasso Plattner Institut
Digital Engineering · Universität Potsdam