Improving CPU energy efficiency during I/O bottlenecks.

Francisco Jerez
LPC 2020

August 28, 2020

What's the purpose of this optimization?

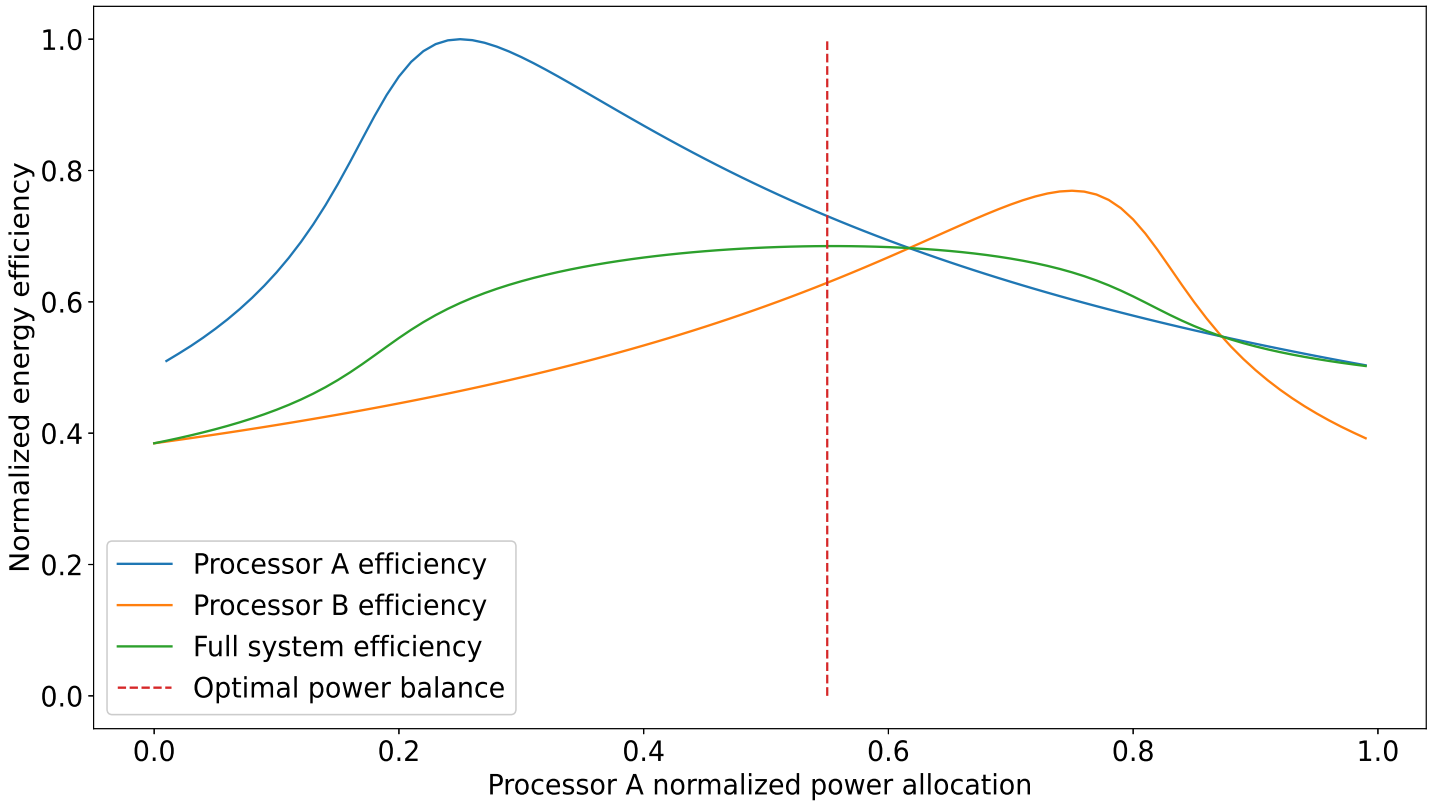
Primarily, reduce energy usage of the CPU where performance doesn't scale with CPU frequency due to an I/O bottleneck, and as a side effect:

- Improve parallelism/throughput in power-sharing scenarios,
- improve battery life,
- reduce carbon/monetary cost of running the system.

Relation to power-sharing scenarios.

As long as the performance of two devices A and B is constrained by a shared energy budget, optimal performance is achieved at the energy allocation with optimal efficiency. E.g.:

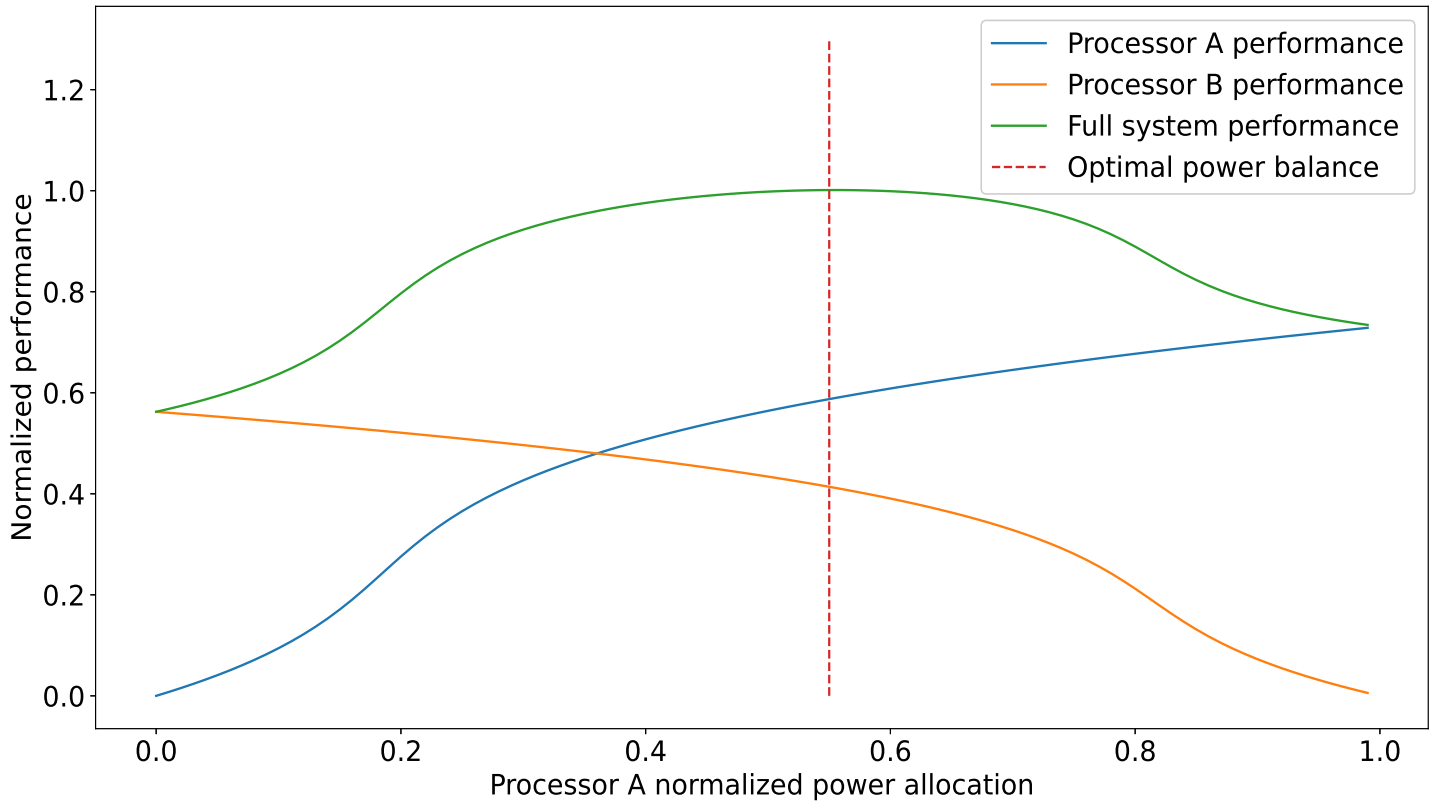
- If A and B are processors running independent tasks:



Relation to power-sharing scenarios.

As long as the performance of two devices A and B is constrained by a shared energy budget, optimal performance is achieved at the energy allocation with optimal efficiency. E.g.:

- If A and B are processors running independent tasks:



Relation to power-sharing scenarios.

As long as the performance of two devices A and B is constrained by a shared energy budget, optimal performance is achieved at the energy allocation with optimal efficiency. E.g.:

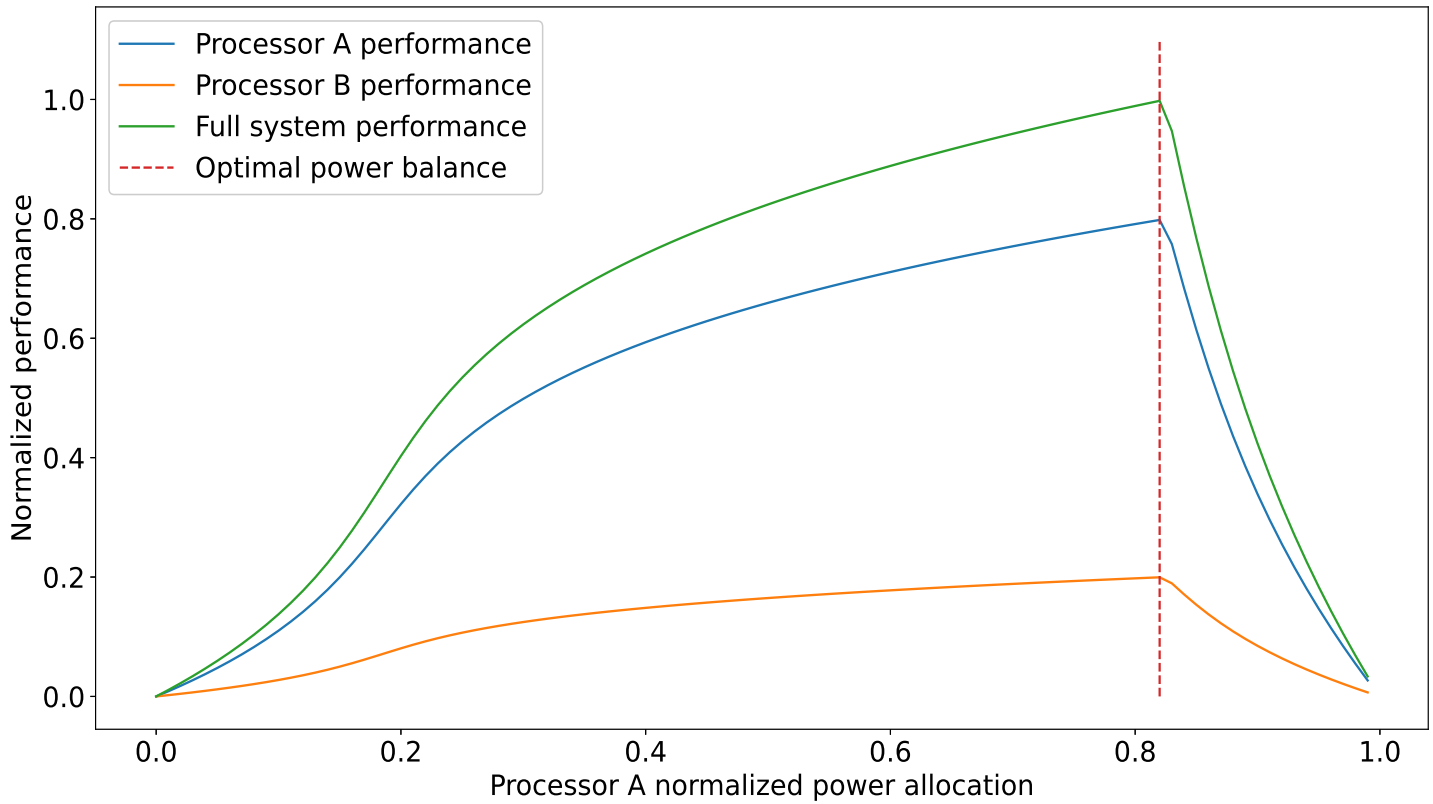
- ~~If A and B are processors running independent tasks:~~

Not the immediate purpose of this optimization. It isn't intended to orchestrate between independent energy-consuming actors, better use IPA for that, however a notion of the share of power required by each actor is necessary for IPA to work effectively, which this optimization can be helpful with.

Relation to power-sharing scenarios.

As long as the performance of two devices A and B is constrained by a shared energy budget, optimal performance is achieved at the energy allocation with optimal efficiency. E.g.:

- If A and B are processors running mutually dependent tasks such that a unit of B's work requires the completion of k units of A's work:



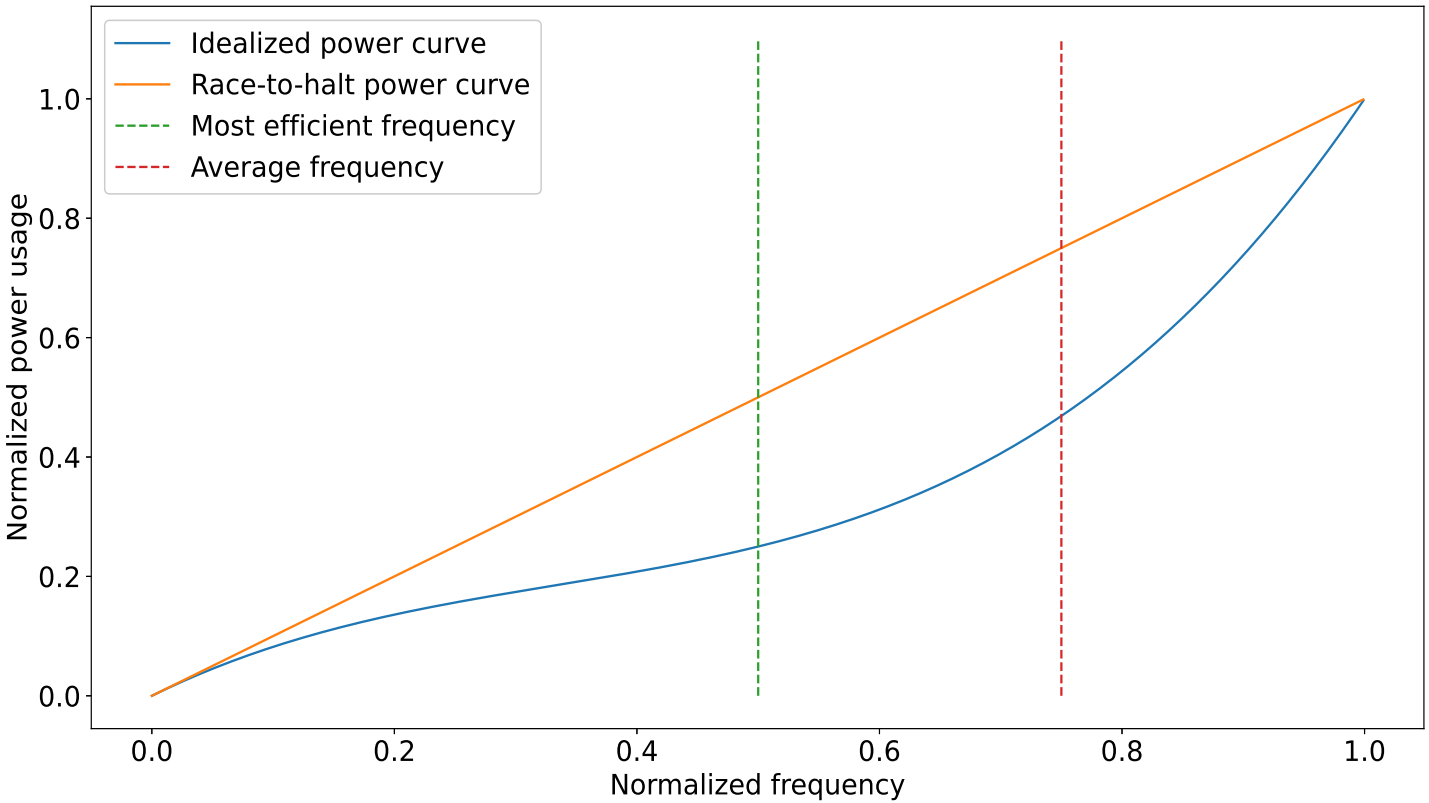
Relation to power-sharing scenarios.

As long as the performance of two devices A and B is constrained by a shared energy budget, optimal performance is achieved at the energy allocation with optimal efficiency. E.g.:

- If A and B are processors running mutually dependent tasks such that a unit of B's work requires the completion of k units of A's work:

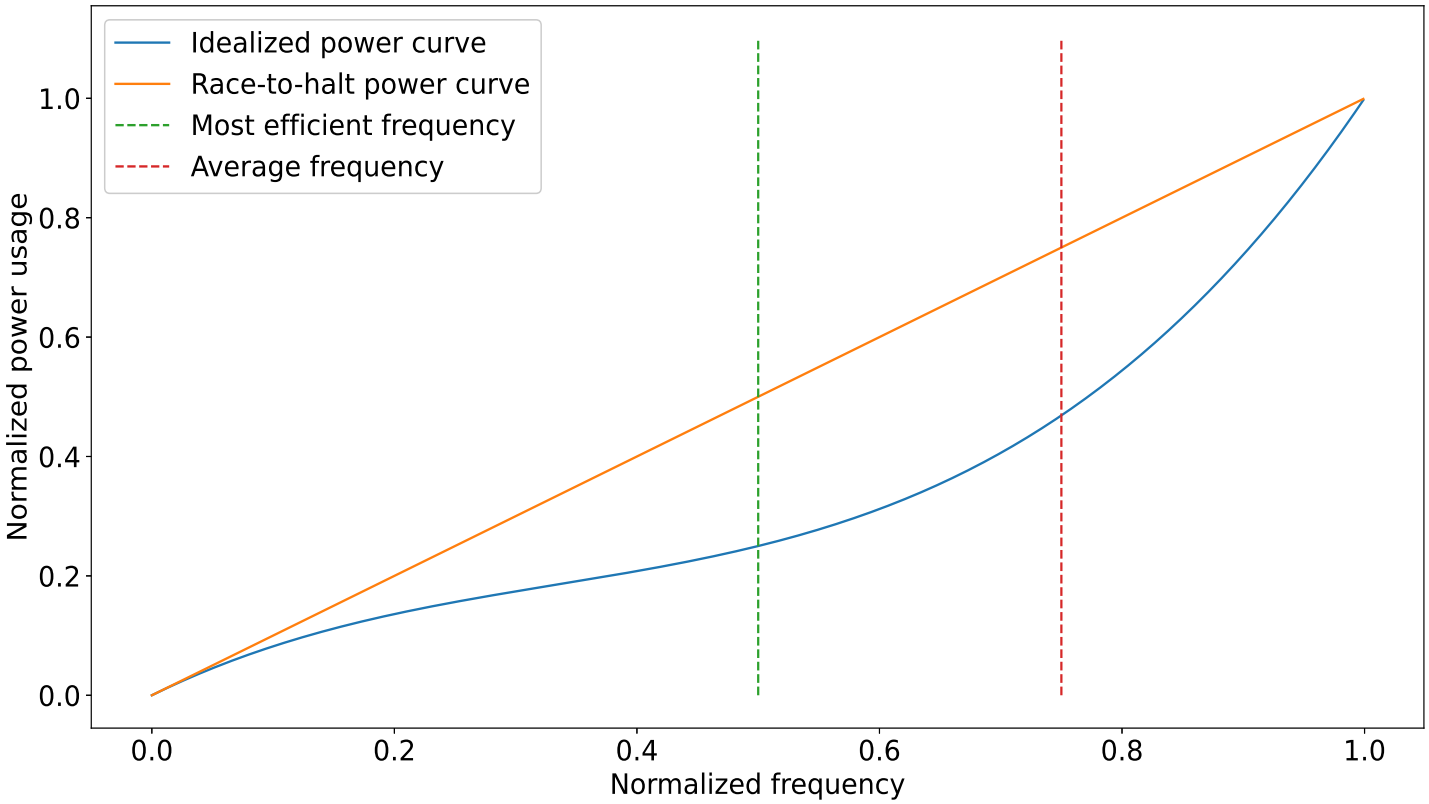
The long-term performance of both processors is correlated. As soon as processor A has a bottleneck in processor B its performance becomes decreasing with increasing power. Therefore if processor A limits its power usage to the minimum while remaining in the same bottleneck region, power allocation will approach the optimal balance line, without there being any central energy-allocation entity.

Principle of operation.



Relies on the knowledge of a single point of the power curve: the frequency of maximum efficiency. The power curve can generally be assumed to be convex beyond that point.

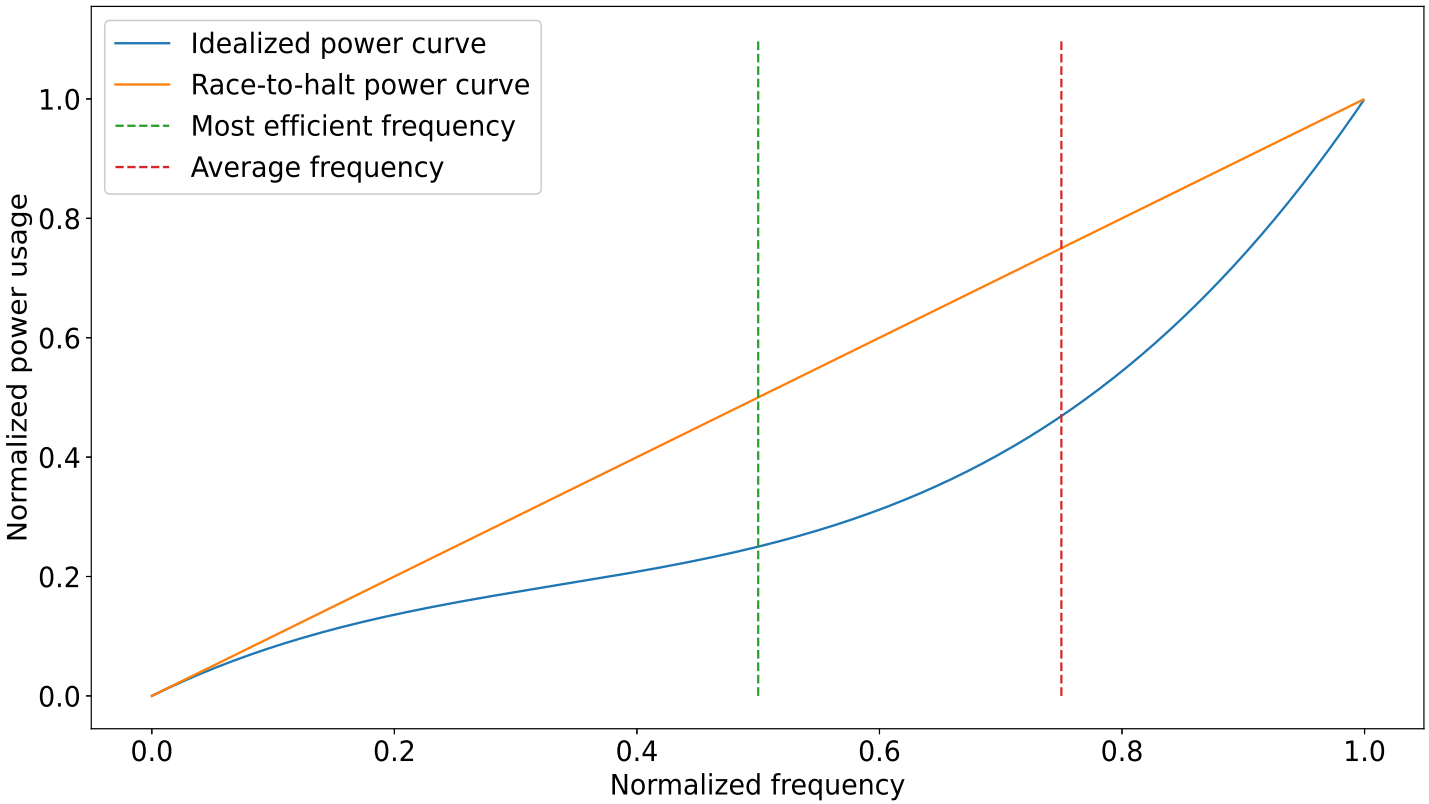
Principle of operation.



The average power consumption $\bar{P}$ of any workload running at a given average frequency $\bar{f}$ is just a linear combination of points of the power curve, but thanks to the convexity of $P(f)$:

$$\bar{P} = \frac{\Sigma_i P(f_i)\Delta t_i}{\Sigma_i \Delta t_i} \geq P\left(\frac{\Sigma_i f_i \Delta t_i}{\Sigma_i \Delta t_i}\right) = P(\bar{f})$$

Principle of operation.



So the energy usage of the processor going through a sequence of frequency states f_i is guaranteed to be greater than running the same workload at the (performance-equivalent) fixed frequency $\bar{f}$. All we need is a crystal ball in order to figure out $\bar{f}$. Easy, right?

Information required for optimality.

Under the convexity assumption, energy-optimal control requires knowledge of no more than:

- The future load of the application.
- The latency constraints of the application.

Cf. the bandwidth and deadline parameters of the DL scheduler.
No details about the power curve (beyond its convexity region) or energy budget are required for optimality.

Steady-state approximation.

- Heuristic assumption: Approximate the future load of the application based on its immediate past. Nothing new in the CPUFREQ subsystem, underlies nearly every non-trivial governor in tree.
- However the definition of “immediate past” involves a trade-off between response latency and accuracy we cannot effectively make for the application.

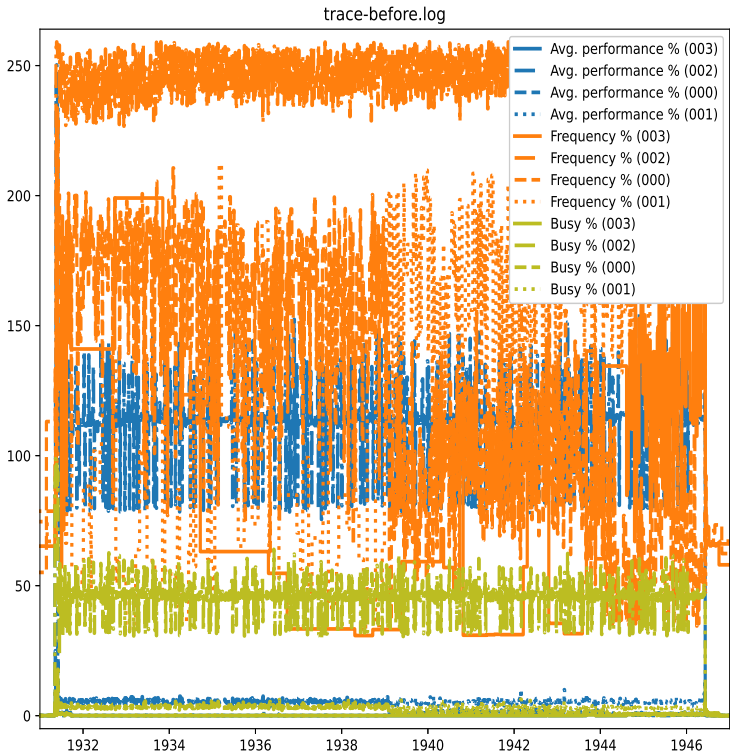
Shifting from frequency to latency constraints.

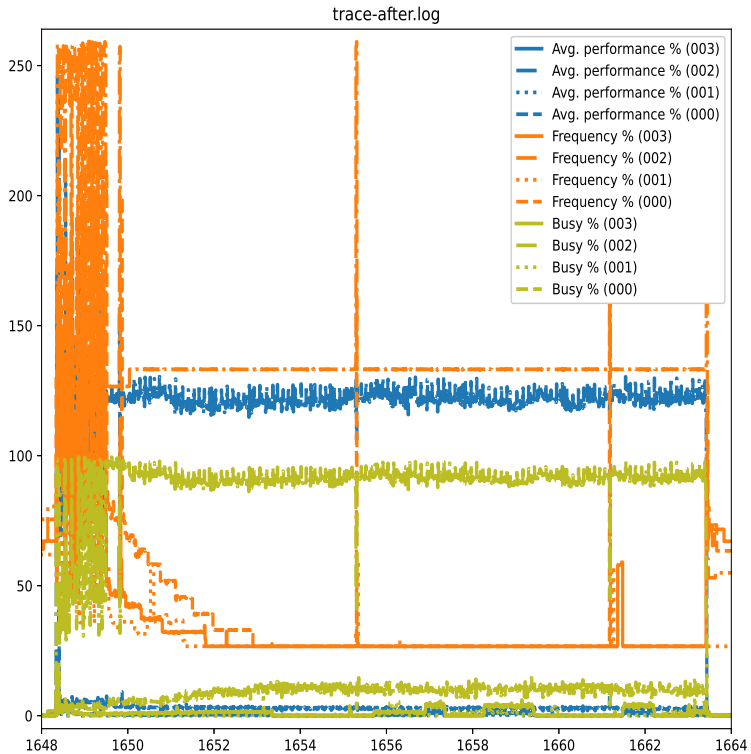
- Relieves applications of the difficult task of estimating their frequency requirements.
- Allows applications to decide what the right trade-off is between latency and energy efficiency. Provides CPUFREQ a hint of the time window the application can afford to have its CPU time rearranged by energy-efficiency optimizations.
- Unlike frequency QoS constraints, high-latency applications shouldn't be able to degrade the performance of low-latency applications running on the same processor.
- Involves a single component monitoring the performance of the CPU instead of O(n) on the number of applications attempting to improve their energy efficiency.

Can we avoid fixing every userspace application out there?

- Frequently devices have natural latency constraints (e.g. imposed by monitor refresh rate, audio sample rate, network latency). What if device drivers could infer reasonable latency bounds for their userspace clients?
- Main risk: Overestimating the latency requirements of applications that don't provide an explicit constraint.
- Restriction to avoid such risk: Apply device-derived latency constraint only if the application has a bottleneck in the device. Doesn't prevent optimal balancing as described in slide 7.

Effect on a live graphics workload.





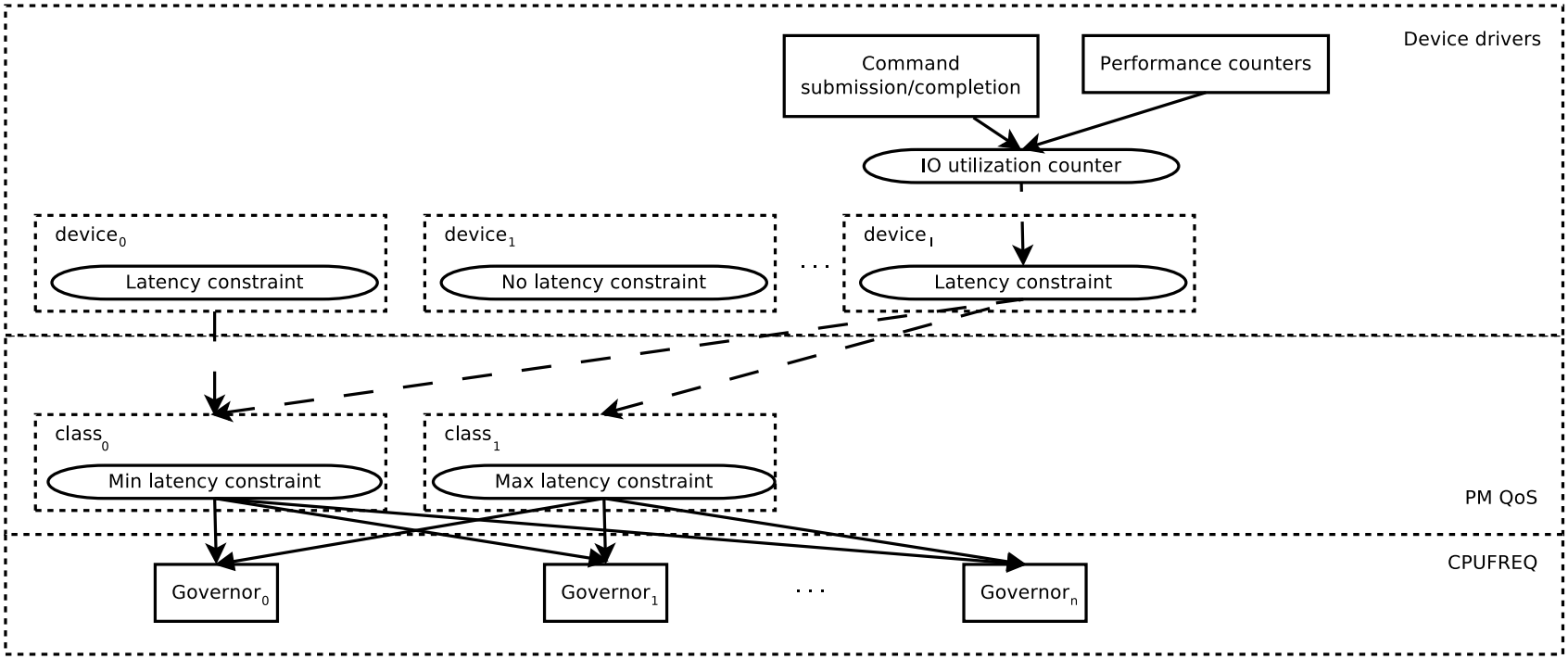
Plumbing latency constraints from clients.

Potential alternatives:

- Custom global CPUFREQ interface (as in v1).
- Global PM QoS class or global constraint interface like `cpu_latency_qos_*` introduced by Rafael (as in v2).
- Per-CPU PM QoS. Caveat: $O(n)$ on number of CPUs, or cat-and-mouse game.
- Scheduling-based with new utilization clamp.
- Scheduling-based with custom interface.

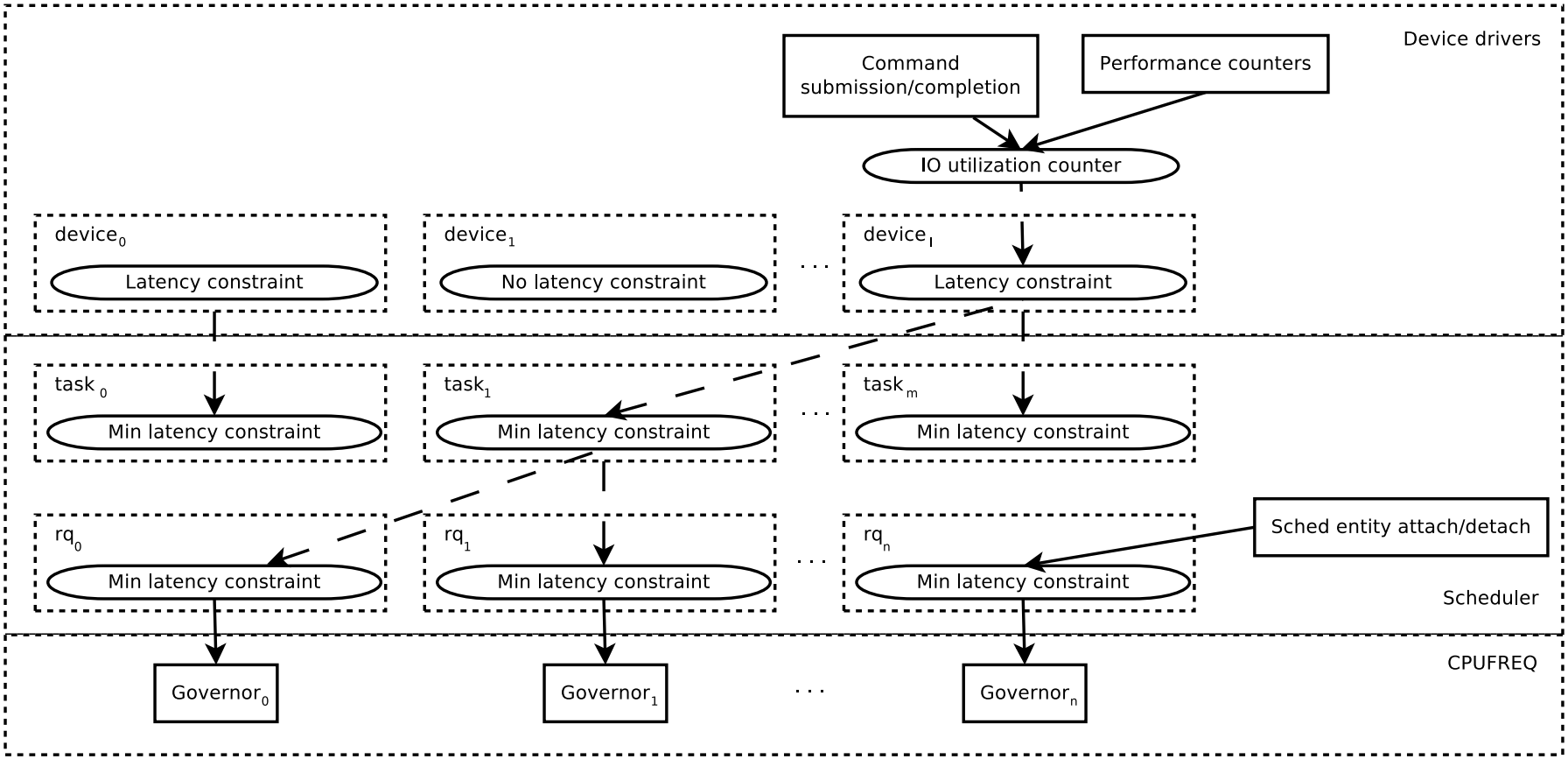
Plumbing latency constraints from clients.

- PM QoS-based solution:

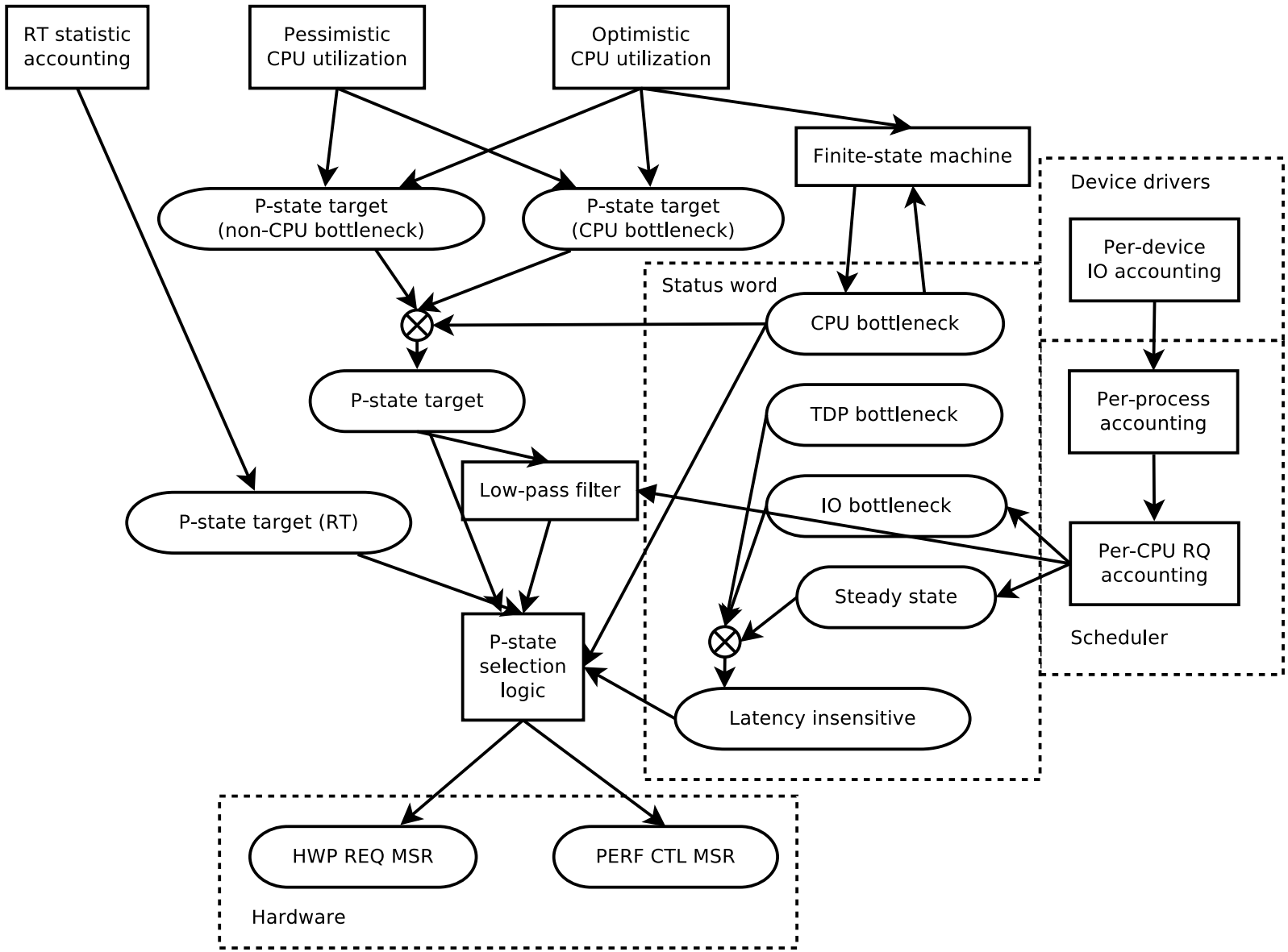


Plumbing latency constraints from clients.

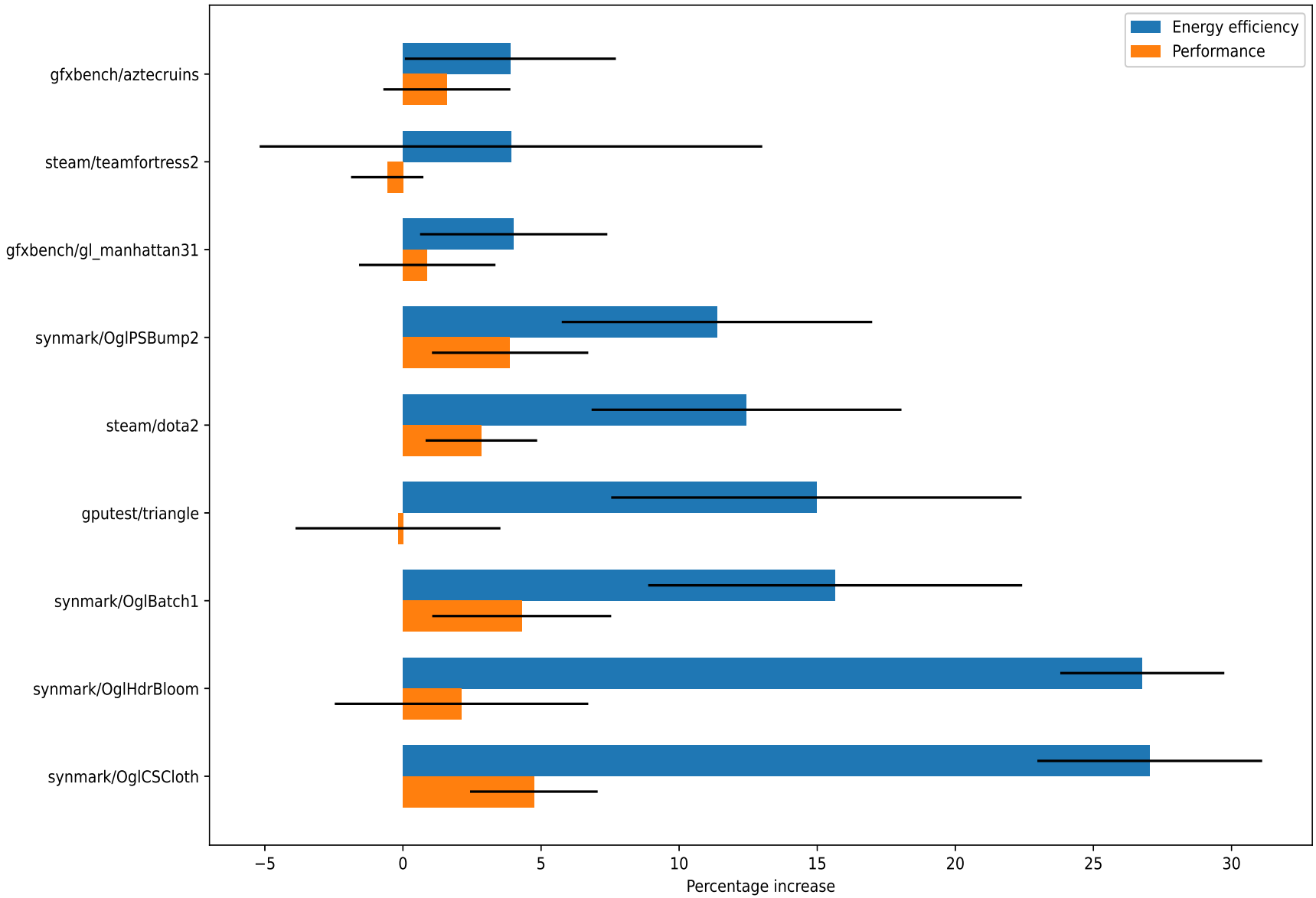
- Scheduling-based solution:



Putting it all together.



Benchmark results.



Status.

`https://github.com/curro/linux/tree/intel_pstate-vlp-v2.99`

- `intel_pstate`-based implementation of controller pending review since several months.
- Some optimizations improving energy efficiency beyond v2.99 coming up for the i915 code.
- Algorithm could be ported to other governors.
- Documentation could use some improvement.