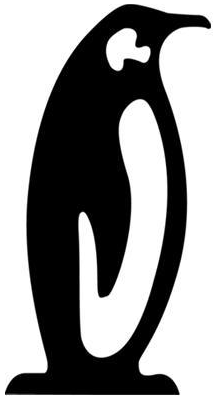


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Taming the Chiplet

High Performance CCX scheduling via BPF

Aniket Gattani
Josh Don
Peilin Ye

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Agenda

- Background / EEVDF Attempts
- BPF CCX-Aware Scheduling Policy
- Parameter Tuning
- Testing & Results
- Q&A

Background / EEVDF Attempts

What are we trying to achieve?

- Dynamic, soft affinity
  - Dynamic => no pre-programmed affinity mask (ie. pick a single LLC domain, doesn't matter which one)
  - Soft => adhere on a best-effort basis, but allow spillover under contention

- Maintain thread grouping properties
  - Keep threads of a process/cgroup together on a set of cores

- Load balancing improvements
  - Keep cores balanced without excessive overhead, and improve overall work conservation
  - Tight packing of SMT siblings vs expansion to new cores
  - Increase entropy in idle cpu search to avoid crowding low-numbered cores

- Tie the scheduling policy closely to the application
  - Expose tunables to make feedback driven optimization effective

Achieving this in EEVDF

- Prototype attempt:
  - Hijack the wakeup path to achieve desired distribution
  - Trigger active balancing via tick to migrate threads running outside their preferred region back to their home when capacity becomes available
  - Works, but:
    - Need to also add code to prevent the periodic/idle load balancer from working against us
    - We lose all the other wakeup heuristics
    - Requires careful synchronization via atomics within e.g. a given cgroup
    - Non-trivial computation in the wakeup path
    - Difficult to tune
    - **We can do much better if we tailor to the unique case of VM scheduling (more on that later)**

Achieving this in EEVDF

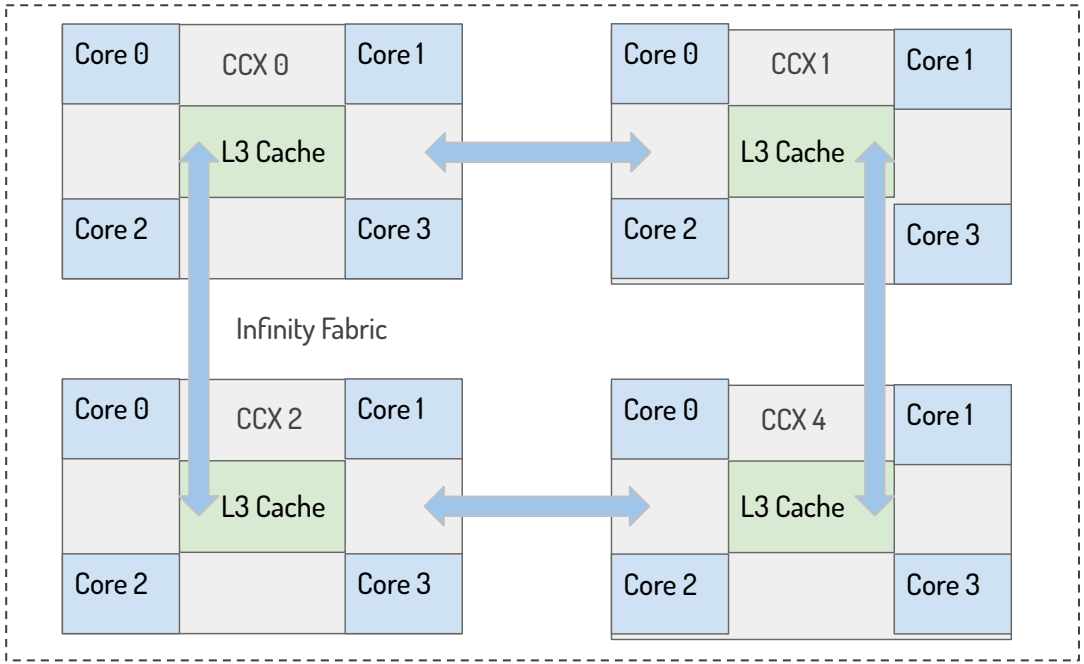
- Cache Aware Scheduling
  - Active series on the mailing list to keep groups of threads on a single LLC domain
  - Builds on top of the existing load balancing mechanisms
  - Dynamically push threads to a more active LLC (from their group's perspective)
  - Has some support for LLC overload (avoid pushing too much load to the "preferred" LLC)
  - Works, but:
    - Needs some additional semi-expensive computation in balancing (e.g. to sift through the task list to find good targets to pull)
    - Relies on point-to-point load balancing; not as much top-down orchestration
    - Overload spillover is not tightly constrained (ie. to as few unique non-preferred LLC as possible)

BPF CCX-Aware Scheduling Policy

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

AMD Zen5 Topology

Topology of a single socket



Cache 2 Cache latencies

Max latencies	Genoa (ns)	Turin (ns)
Intra CCX	25.3	25.4
Same Socket	112	143
Cross Socket	199	232

Disclaimer: Not accurate please refer to AMD's Microarch Specs.

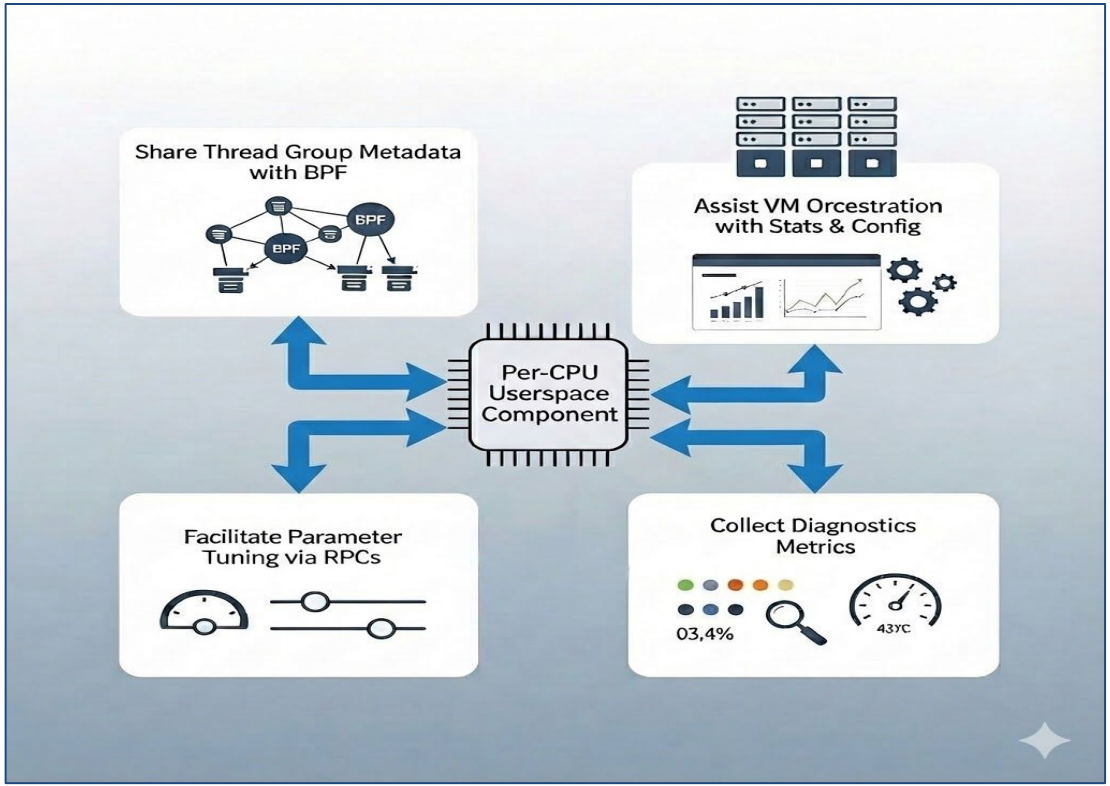
Workload Characteristics

- CPU Overcommitment: (more vcpus than physical cpus). Upto 2x.
- Using AMD's Zen5 Turin processors. 256 cores (across 2 sockets with SMT upto 2).
- General Purpose Workloads for VM scheduling. Balance performance and locality at the cost of wait times..
- VMs can be of multiple defined shapes and can comprise of multiple thread groups.
- Each thread group size has to be smaller than a CCX (LLC).
- Cross NUMA migration is disabled.

Orthogonality from EEVDF

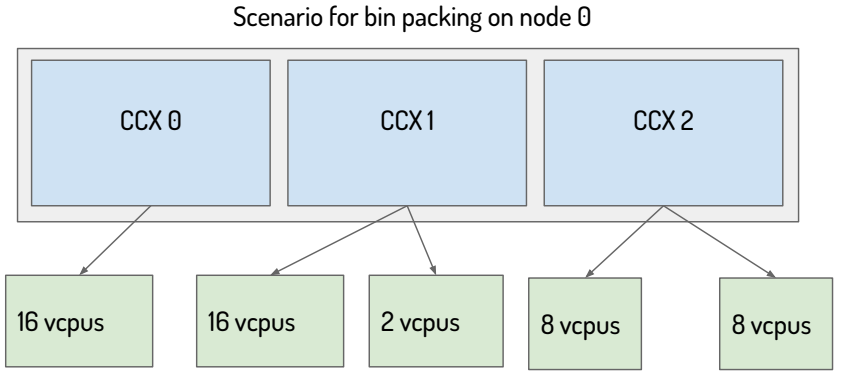
- Runqueue is **CCX instead of per-CPU**. Work conservation across a CCX.
- Async bin packing algorithm, in userspace, for soft affining thread groups to their home CCX.
- Load balancing tries to migrate to different CCXes instead of specific CPUs.
- We have finite overcommit ratio simplifying bin-packing and load balancing.
- Significant userspace component..
- Rich set of stats and metrics are exported via RPCs. Easy to add tunables. Enables closed loop analysis, optimization and post mortem.

Userspace Agent



Bin Packing Considerations

- Different shaped thread groups.
- Priority of different vLLCs based on the size and also cpu util.
- Should support targeted soft affinity to specific pLLCs
- Prevent skewness in assignments as much as possible
- Keep the assignments semi-stable.
- Bin packing interval should be controllable via tuning knobs.



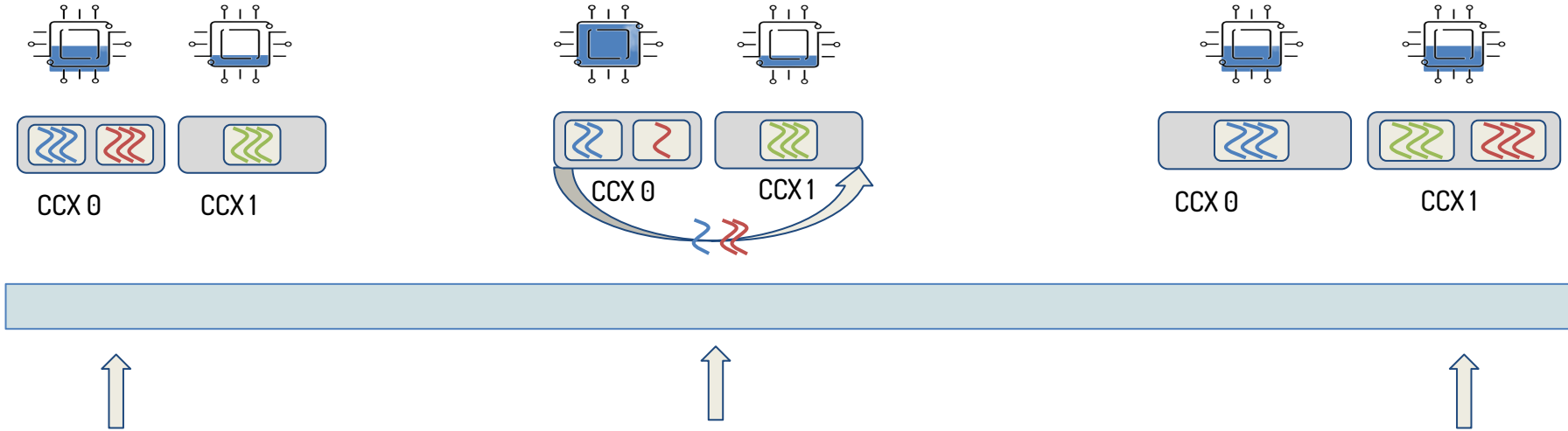
Bin Packing Algorithm

$$\texttt{score(pllc, vllc)} = f(\texttt{vllc.num\_cores}, \texttt{pllc}) \cdot \texttt{cores\_coeff} + h(\texttt{vllc.load}, \texttt{pllc}) \cdot \texttt{load\_coeff} - g(\texttt{vllc.prev\_home}, \texttt{pllc}) \cdot \texttt{prev\_home\_coeff}$$

$$\texttt{best\_pllc} = \underset{0 \leq \texttt{pllc} < \texttt{num\_pllcs}}{\operatorname{argmin}} (\texttt{score(pllc, vllc)})$$

- Sort the vLLCs according to their load and size. Assign prev home if possible else score different pLLCs against the vLLC.
- Scoring function considers different parameters
- Find the pLLC with least score

Overload & Bin-packing in Tandem



Bin packing epoch

Instantaneous burst in cpu util triggering Overload on CCX 0

Bin packing epoch

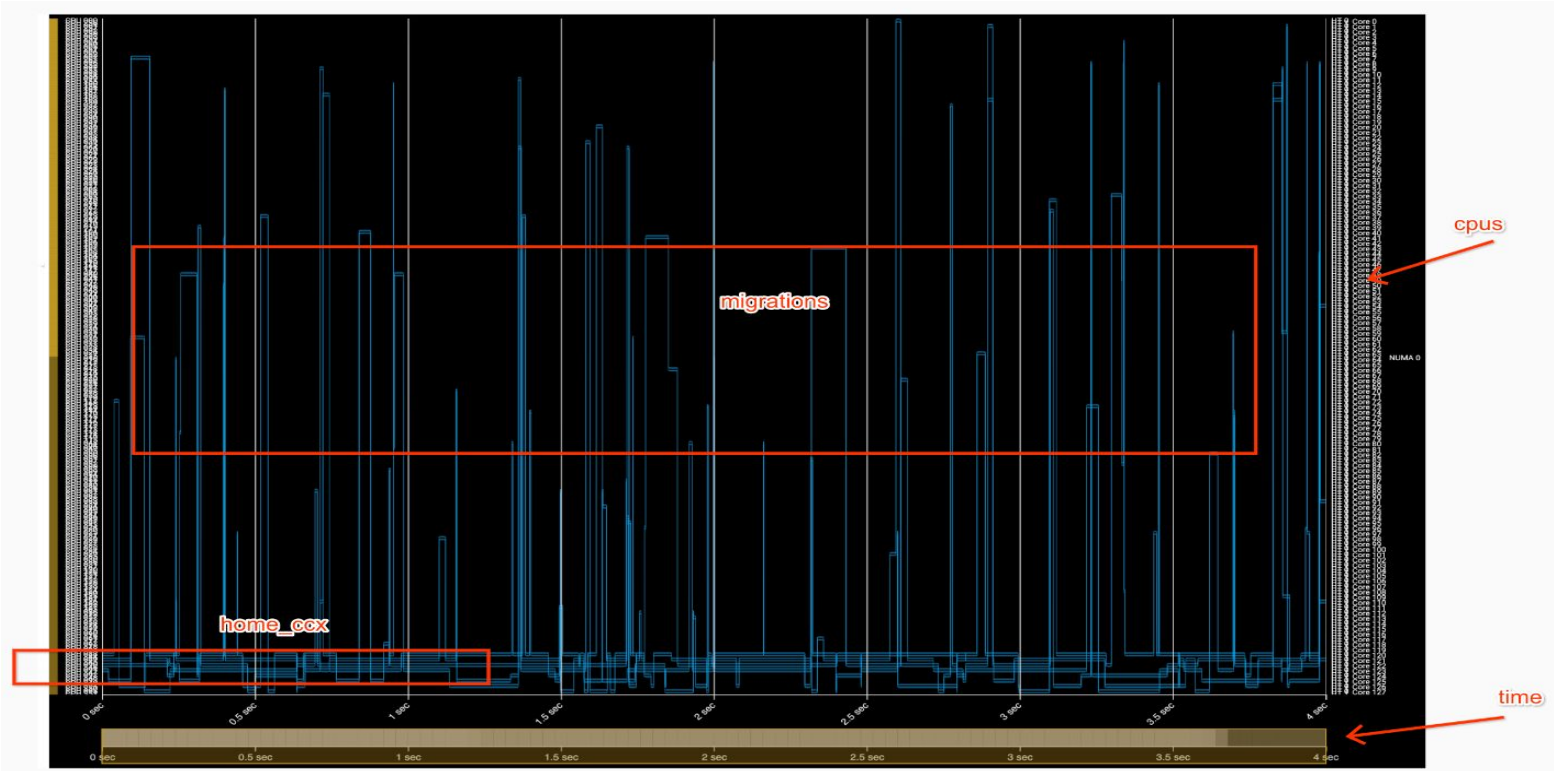
Runqueue

Thread Group

Time

Avg CPU Util of CCX

Overload in action



Overload Mechanism

- Overload is a shared-fate for the runqueue. All waiting vCPUs get migrated.
- Aggressiveness of the overload is controlled via a tuning knob.
- Previously, we implemented a min-wait time threshold check on every PNT edge to trigger overload.
    - Problems:
        - Hard to get accurate. Too small threshold would cause locality loss. Small spikes in cpu-utilization could trigger overload.
        - Too big would cause high vcpu-wait times
        .
        - Frequent wakers / sleepers encounter small periods of wait times
    - Workaround:
        - Sliding window algorithm over a period of time instead of a single event. Provides better tolerance.

Parameter Tuning

Parameter Tuning

- Brute forcing just explodes the parameter space.
- Takes too long. Each benchmark can run for several days.
- Default value space is hard to pick. Possibly ML tuning can help.
- Could depend a lot on what kind of workloads were used.

RT Bench Total lost p99.9 (Memory Overcommit)			
	bin_packing_interval_ms		
overload_wait_thresh_ms	1000	5000	10000
54	198.5	194.5	201
108	203.5	209	216.5
162	201	206	208

SpecjBB Critical_Jops (CPU Overcommit)			
	bin_packing_load_coeff		
bin_packing_prev_home_coeff	100	200	400
100	7974.00	8035.00	8023.00
200	8508.97	7880.00	7994.00
400	8364.00	7936.00	8614.00

Testing & Results

BPF vs EEVDF

Isolated Testing

Benchmark	Metric	BPF	EEVDF	Perf Uplift
SIR_17	Geomean Score	35.9	35.8	+0.27%
SpecjBB	Critical JOPS	21166.5	20992.5	+0.8%
Rt_Bench	Total_Lost_P99.9	121.5	136	+10%
	Total_Lost_P99.999	249.5	533.5	+53.24%**

Stream Antagonist putting memory pressure

Benchmark	Perf Reference	Metric	BPF	EEVDF	Perf Uplift
SIR_17	Higher is Better	Geomean Score	32.1	22.45	+43%
SpecjBB	Higher is Better	Critical JOPS	16493.5	7903	+108.6%
RT_Bench	Lower is Better	Total_Lost_P99.9	203.5	313.5	+35.1%
		Total_Lost_P99.999	3540.5	5591.5	+36.7%

BPF vs EEVDF

Antagonist contending for cpu. Overcommit factor = 1.7x

Benchmark	Perf Reference	Metric	BPF	EEVDF	Perf Uplift
SIR_17	Higher is Better	Geomean Score	26.4	25.20	4.76%
	Lower is Better	Wait-Time (ms/s)	29.63	55.37	86.87%
	Higher is Better	Locality %	94.96	46.34	48.62%
SpecjBB	Higher is Better	Critical JOPS	8508.97	6284.00	35.41%
	Higher is Better	Max JOPS	30511	23853.00	27.91%
	Lower is Better	Wait-Time (ms/s)	40.25	64.94	61.34%
	Higher is Better	Locality %	88.85	39.46	49.39%

Q&A

Arigatō

- *From the gh0St team*