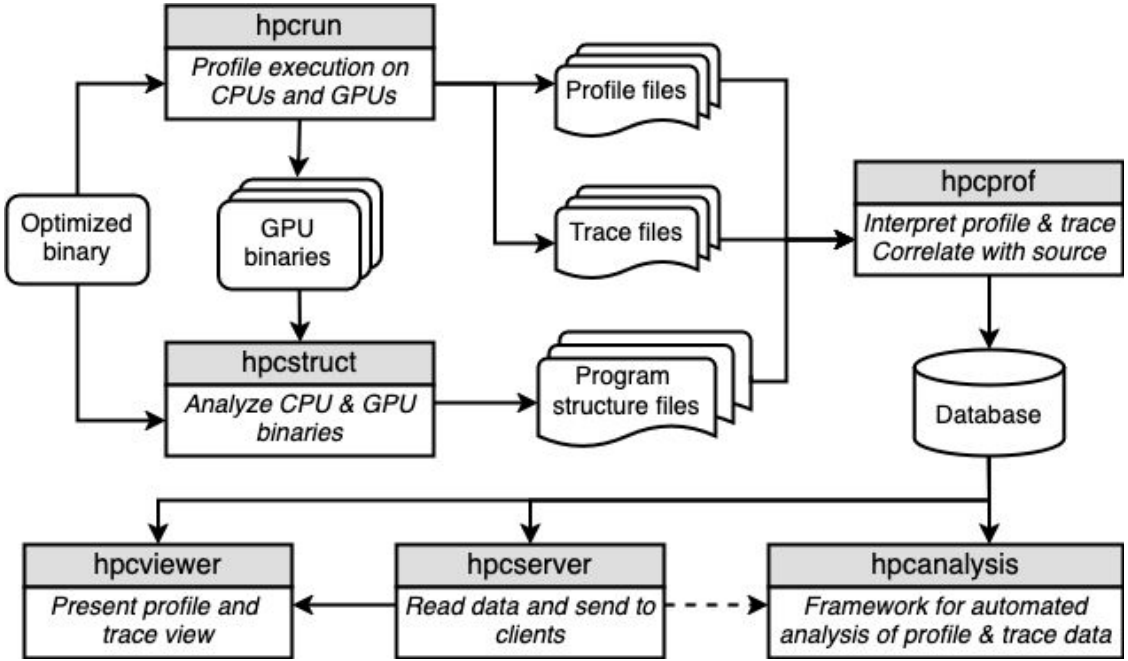
Top-Down Microarchitecture Analysis in HPCToolkit

Laksono Adhianto

Rice University

HPCToolkit Workflow

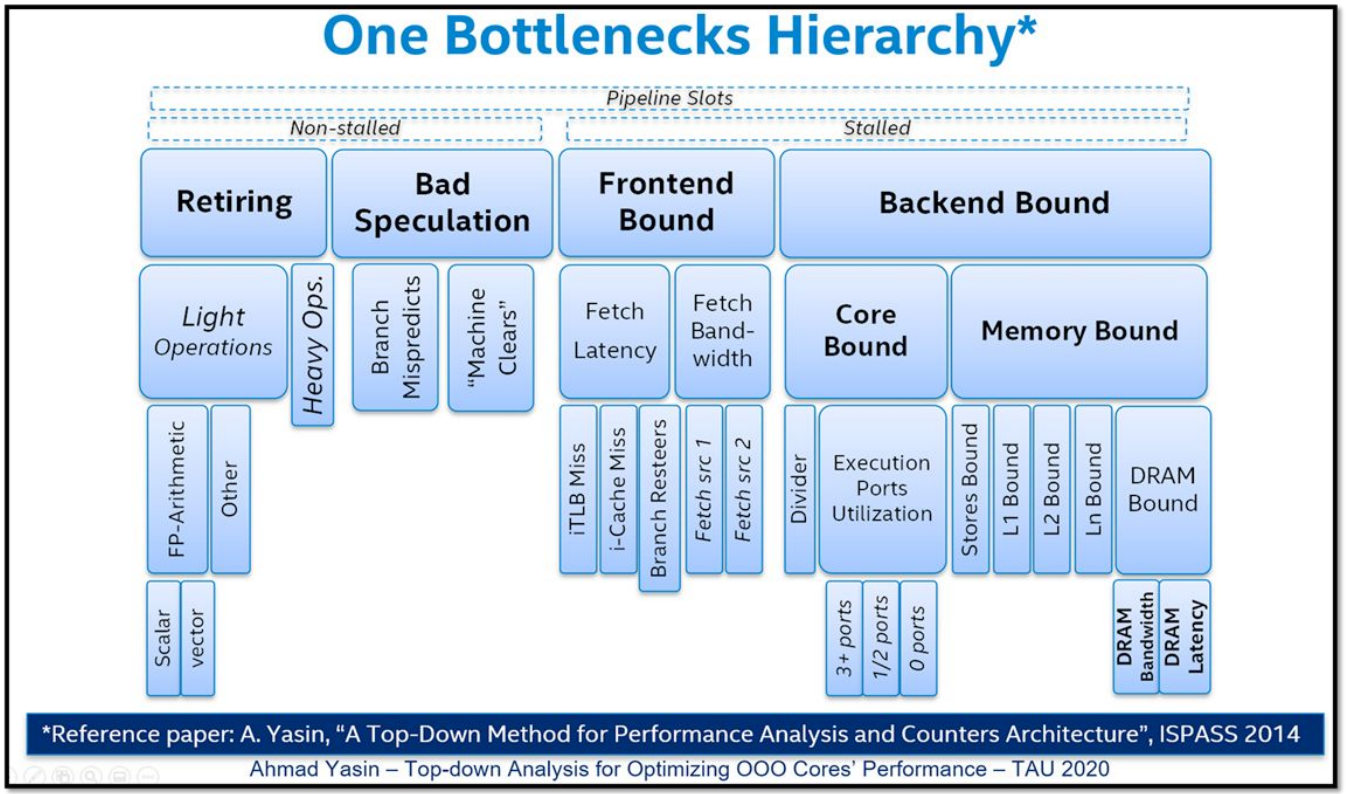


CPU Top-Down Methodology Analysis

- To identify software performance bottlenecks by categorizing CPU pipeline **slots** into four primary states: Retiring, Bad Speculation, Frontend Bound, and Backend Bound
  - If a CPU has an issue width of 6, then 1 Pipeline Cycle = 6 Slots
- Hardware support
  - Intel [1]: TMA with special register
  - Arm [2]
  - AMD
  - IBM POWER: CPI stack model [3]
  - GPUs: AMD, NVIDIA, Intel

[1] A. Yasin, “A Top-Down method for performance analysis and counters architecture,” 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, USA, 2014, pp. 35-44, doi: 10.1109/ISPASS.2014.6844459
[2] J. Mundichipparakkal, D. Greene, J. Randall, V. Coubard, M. Williams and R. D. Jong, "Multi-Level TDA for Heterogeneous Arm Processors Using a Standardized Telemetry Framework," 2026 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Seoul, Korea, Republic of, 2026, pp. 173-183, doi: 10.1109/ISPASS69572.2026.00026.
[3] M. Srinivas *et al*., "IBM POWER7 performance modeling, verification, and evaluation," in *IBM Journal of Research and Development*, vol. 55, no. 3, pp. 4:1-4:19, May-June 2011, doi: 10.1147/JRD.2011.2147170.

Intel Top-Down Methodology Analysis (**TMA**)



Intel PERF_METRICS MSR

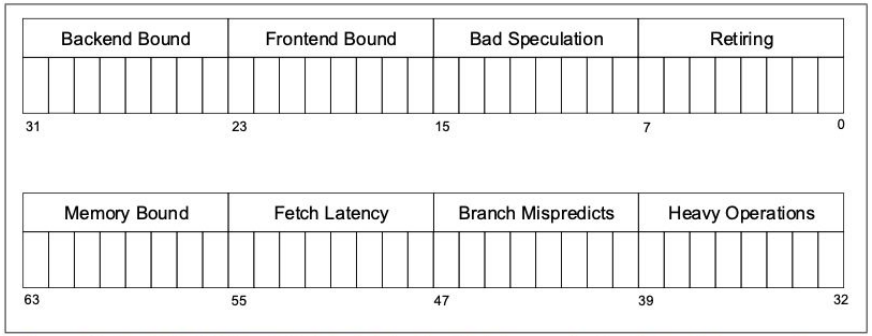


Figure 21-40. PERF_METRICS MSR Definition for 12th Generation Intel® Core™ Processor P-core

Core_Bound:	Backend_Bound - Memory_Bound
Fetch_Bandwidth:	Frontend_Bound - Fetch_Latency
Machine_Clears:	Bad_Speculation - Branch-Mispredict
Light_Operations:	Retiring - Heavy_Operation

- Intel **PERF_METRICS MSR**: a special register to provide percentages of slots for four TMA level 1 and four TMA level 2 metrics
- Each metric is represented as an 8-bit fraction of the total allocated slots
- Four additional TMA level 2 metrics (Core Bound, Fetch Bandwidth, Machine Clears and Light Operations) are derived from the metrics

Intel PERF_METRICS MSR

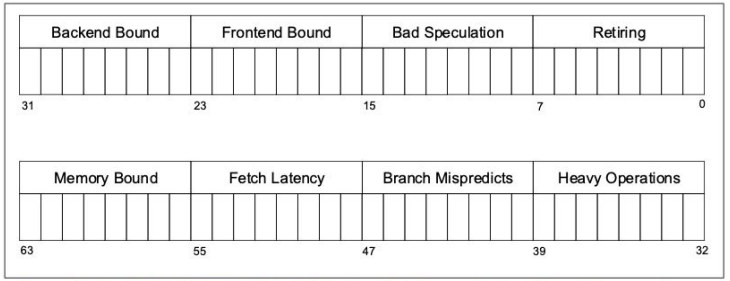


Figure 21-40. PERF_METRICS MSR Definition for 12th Generation Intel® Core™ Processor P-core

- Limited granularity: percentage granularity
- Accuracy: one metric → 8 bits → 1 / 255 = ±0.4%
  - absolute_value = field_value x TOPDOWN.SLOTS
- Not a precise event
- No sampling support
  - In Linux perf: supported in `perf stat` (counting mode), but not `perf record` (sampling mode)
- PERF_METRICS is designed around interval accounting
  - HPCToolkit's attribution model is designed around point-in-time events.
- Intel manual:
  - *software is recommended to **periodically** clear both registers in order to maintain accurate measurements for certain scenarios that involve sampling metrics at high rates*.
- Linux perf_events: resetting counter via `ioctl` is very costly

TMA in HPCToolkit

- Intel's TMA methodology was designed for light-weight coarse-grain measurement
- HPCToolkit collects TMA hardware counter values through sampling
  - Uses either `CYCLES` or a hardware counter overflow as a sample trigger
  - Attributes each sample to an instruction address (and the call path that reaches it) where an asynchronous trigger (counter overflow) fires
  - Attributes all metric counts from the same "counter group" to the sample point
    - "proxy sampling"
- HPCToolkit's post-mortem analysis elevates instruction-level metrics to source lines, loops, inlined code, and calling contexts

Proxy Sampling

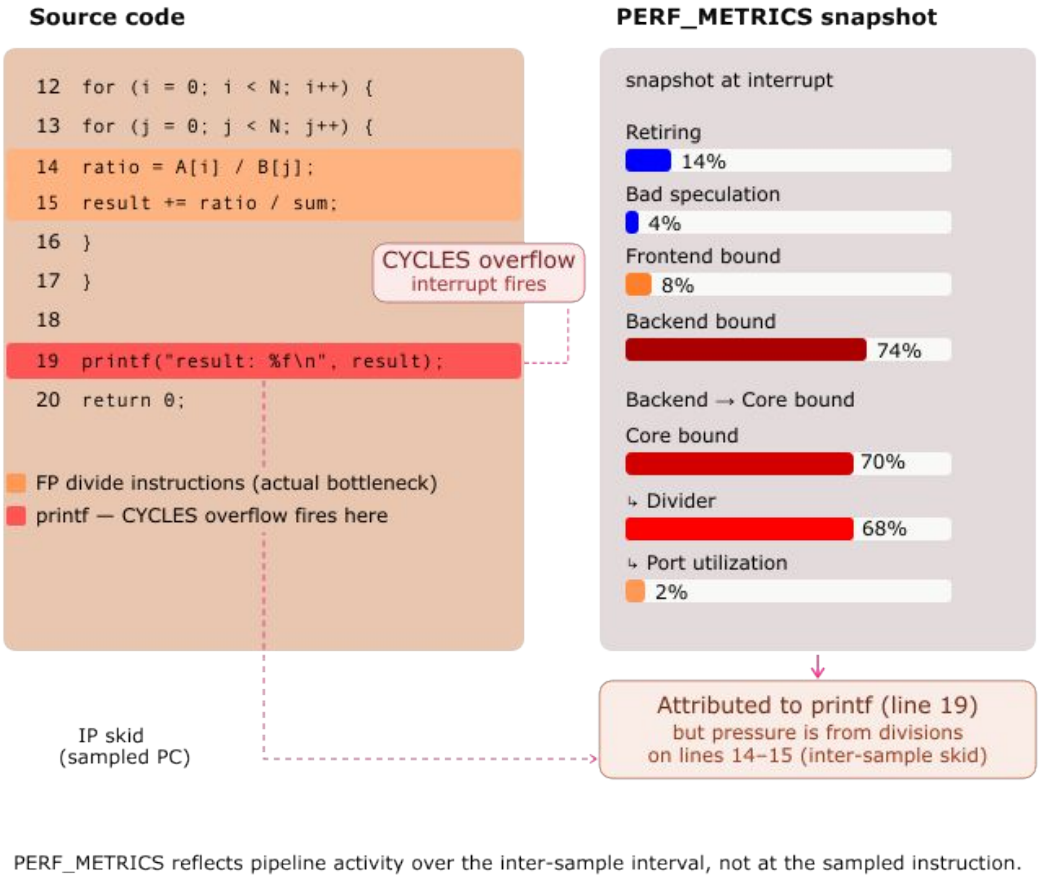
Lines 14–15: the FP divide instructions A[i]/B[j] and ratio/sum are the actual bottleneck

- Non-pipelined Divider costs many cycles
- PERF_METRICS snapshot shows 74%
- Divider unit constitutes 68%

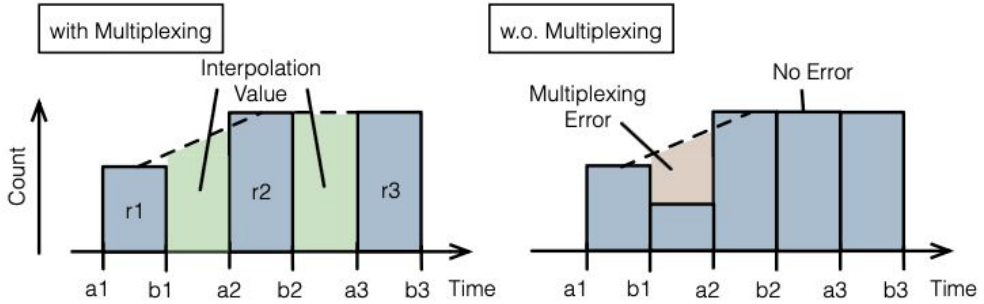
Line 19: the CYCLES counter happens to overflow while the CPU is executing the printf.

- This becomes the sampled PC — the instruction HPCToolkit records as the "location" of the sample.

HPCToolkit attributes the Core Bound / Divider pressure to printf on line 19, even though it has nothing to do with division.



Sampling Uncertainty



- The uncertainty comes from finite number of samples, stochastic sampling, multiplexing, timer skid / PEBS latency, call-path attribution
  - Multiplexing because so many events, so few registers
    - Even worse if SMT is enabled → less registers per thread
- Relative error $\mathbb{E} = \mathcal{S} \cdot \mathcal{M}(n,e)$
  - $\mathcal{S}$: Sampling uncertainty → Poisson samples: $1 / \sqrt{N}$
  - $\mathcal{M}$: Multiplexing uncertainty for a given CCT node n, event e
  - $\mathbb{E} < 0.3$: high confidence
  - $0.3 < \mathbb{E} < 0.6$: medium confidence
  - $0.6 < \mathbb{E}$: low confidence

Image taken from: Sudvarg, Li, et al. “Tintin: PMU Scheduling to Minimize Uncertainty.” 19th annual workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), July 2025

CPU Top-Down Analysis in HPCToolkit

- Currently only supports Intel Sapphire Rapids
- Huge overhead to collect multiple-level of TMA metrics
  - The overhead can be 50% or more
  - Allow users to collect only a specific TMA sub-tree
    - `topdown` – collect TMA level 1 and 2 with low overhead
    - `topdown:retiring` – collect TMA level 1-4 of retiring sub tree
    - `topdown:frontend` – collect TMA level 1-3 of frontend-bound sub tree
    - `topdown:backend` – collect TMA level 1-4 of backend-bound sub tree
    - `topdown:all` – collect all TMA level 1-4, very high overhead

Summary

- Intel PERF_METRICS MSR was designed for low-overhead coarse-grain measurement
  - It is great for an overview of performance bottlenecks in an application
  - No direct support for sampling and fine-grain measurement
- HPCToolkit uses sampling-based fine-grain measurement
  - Require some adjustment to support Intel top-down analysis
  - Huge overhead to collect **all** metrics
    - Solution: allow users to measure only a TMA sub tree
- Future work
  - Enhance proxy sampling by using the “most important” counter as the trigger event
  - Support for sampling confidence level

Future Work: Enhanced Proxy Sampling

- Instead of using `cycles` as the trigger event, use the “most important” event as the trigger.
- To compute L1 bound:

  `perf record -e '{slots,`**`exe_activity.bound_on_loads`**`,retiring,bad_speculation, …}:S' …`
  - `exe_activity.bound_on_loads` triggers PERF_METRICS snapshot
- To compute Store bound:

  `perf record -e '{slots,`**`exe_activity.bound_on_store`**`,retiring,bad_speculation, …}:S' …`
  - `exe_activity.bound_on_store` triggers PERF_METRICS snapshot
- With perf_events: kernel_value = (field_value / 255) x TOPDOWN.SLOTS
  - Since Kernel_value is an absolute value, it is additive and “deltable”