Performance Engineering in CSE: A Bird's-Eye View

Georg Hager, Jan Laukemann

Erlangen National High Performance Computing Center (NHR@FAU)

Minisymposium MS167

SIAM CSE23, Amsterdam, The Netherlands

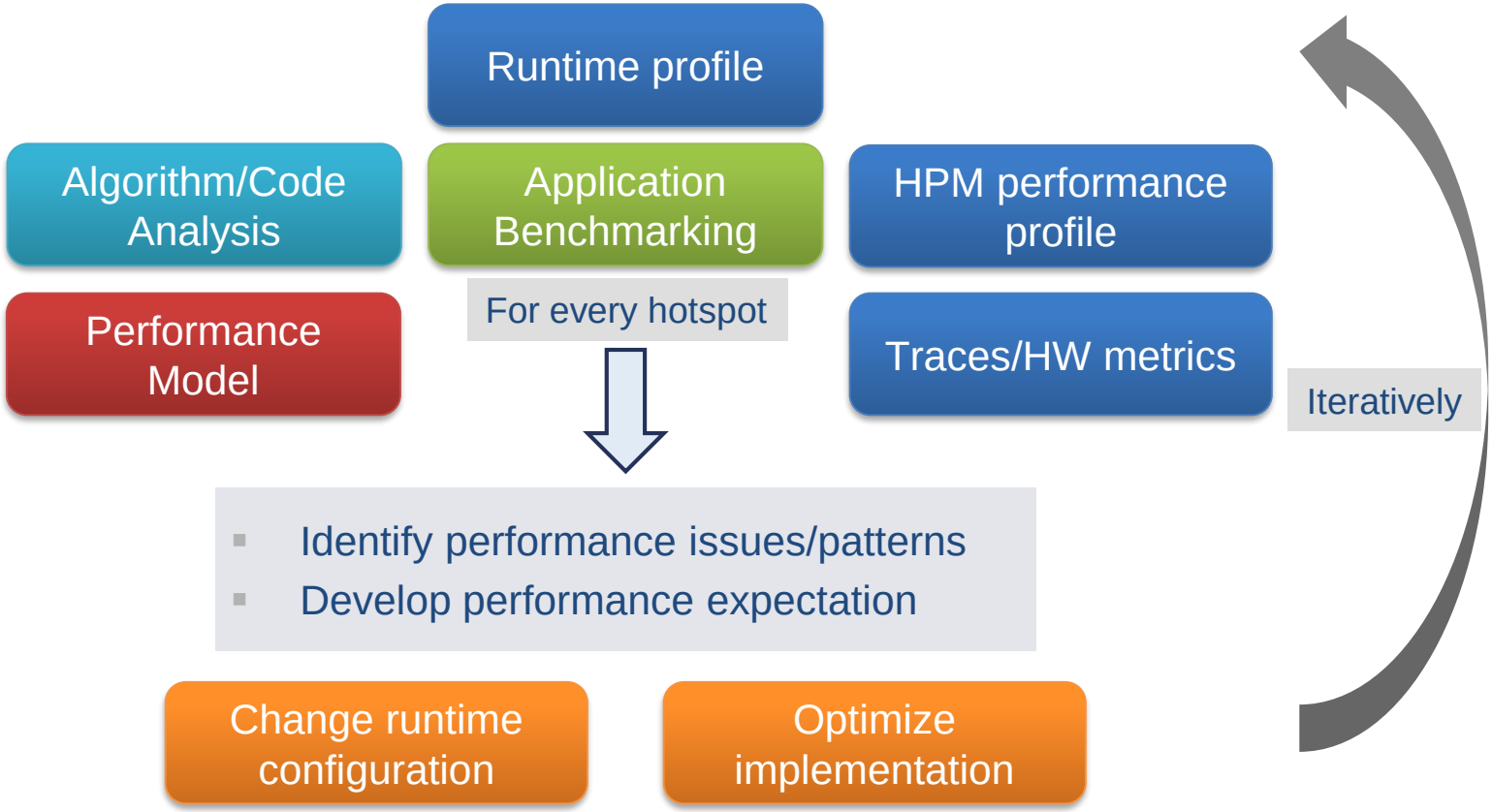
2023-03-01

Agenda

- Performance Engineering process
  - Tools
  - Metrics
  - Patterns

- An inspiring example
  - a.k.a. “The most outrageously expensive way to compute prime numbers”

The PE process



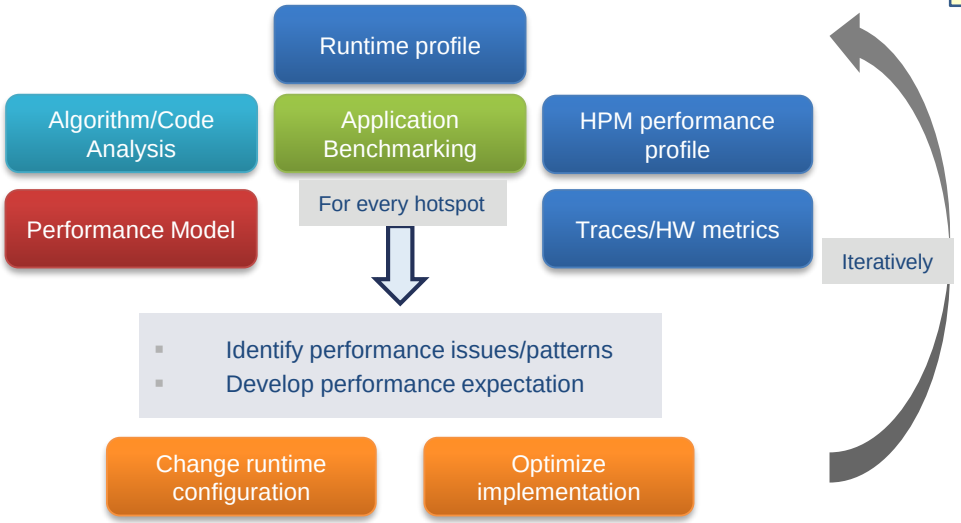
Tools

Profilers
(hotspot detection)

Modeling tools
(XYZ prediction)

HPM tools
(HW/SW interaction)

Tracing tools
(timeline exploration)



Metrics

Performance / runtime

Scalability / speedup

Resource utilization

Runtime profile

Algorithm/Code Analysis

Application Benchmarking

HPM performance profile

Performance Model

For every hotspot

Traces/HW metrics

Iteratively

Variability / behavior

- Identify performance issues/patterns
- Develop performance expectation

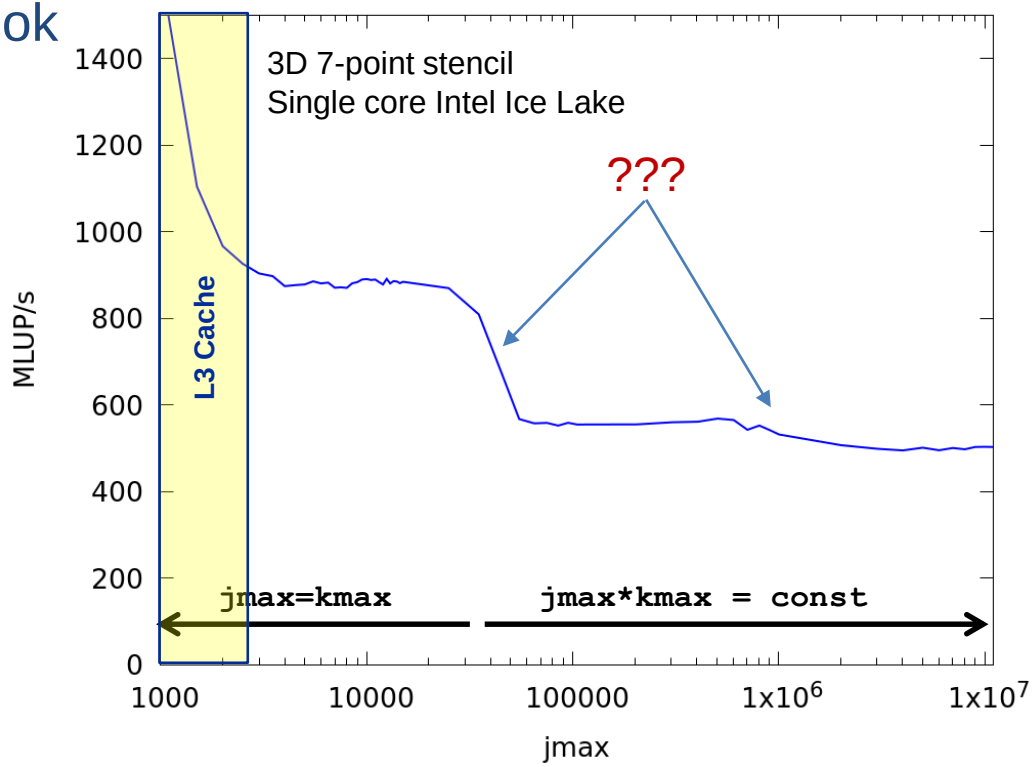
Change runtime configuration

Optimize implementation

“Behavior”?

“How does Y change if I change X?”
is an extremely powerful way to look at performance

- Problem size
- Domain shape
- Domain-process mapping
- # cores/threads/processes
- Affinity settings
- …



Performance patterns

Pattern		Performance behavior	Metric signature & LIKWID [7] performance group(s)
Bandwidth saturation		Saturating speedup across cores sharing a data path	Bandwidth meets BW of suitable streaming benchmark (MEM, L3)
ALU saturation		Throughput at design limit(s)	Good (low) CPI, integral ratio of cycles to specific instruction count(s) (FLOPS_*, DATA, CPI)
Inefficient data access	**Excess data volume**	Simple bandwidth performance model too optimistic	Low BW utilization / Low cache hit ratio, frequent CL evicts or replacements (CACHE, DATA, MEM)
	Latency-bound access		
Micro-architectural anomalies		Significant discrepancy from simple performance model based on LD/ST and arithmetic throughput	Relevant events are very hardware-specific, e.g., memory aliasing stalls, conflict misses, unaligned LD/ST, requeue events, WA evasion
False sharing of cache lines		Large discrepancy from performance model in parallel case, bad scalability	Frequent (remote) CL evicts (CACHE)
Bad ccNUMA page placement		Bad or no scaling across NUMA domains, performance improves with interleaved page placement	Unbalanced bandwidth on memory interfaces / High remote traffic (MEM)

Performance patterns

Pattern		Performance behavior	Metric signature & LIKWID [7] performance group(s)
Pipelining issues		In-core throughput far from design limit, performance insensitive to data set size	(Large) integral ratio of cycles to specific instruction count(s), bad (high) CPI (FLOPS_*, DATA, CPI)
Control flow issues		See above	High branch rate and branch miss ratio (BRANCH)
Load imbalance / serial fraction		Saturating/sub-linear speedup	Different amount of “work” on the cores (FLOPS_*); note that instruction count is not reliable!
Synchronization overhead		Speedup going down as more cores are added / No speedup with small problem sizes / Cores busy but low FP performance	Large non-FP instruction count (growing with number of cores used) / Low CPI (FLOPS_*, CPI)
Instruction overhead		Low application performance, good scaling across cores, performance insensitive to problem size	Low CPI near theoretical limit / Large non-FP instruction count (constant vs. number of cores) (FLOPS_*, DATA, CPI)
Code compo-sition	**Expensive instructions**	Similar to instruction overhead	Many cycles per instruction (CPI) if the problem is large-latency arithmetic
	Ineffective instructions		Scalar instructions dominating in data-parallel loops (FLOPS_*, CPI)

The prime number riddle

A motivating example in PE

SPEChpc 2021 Cloverleaf performance (519_clvleaf_t)

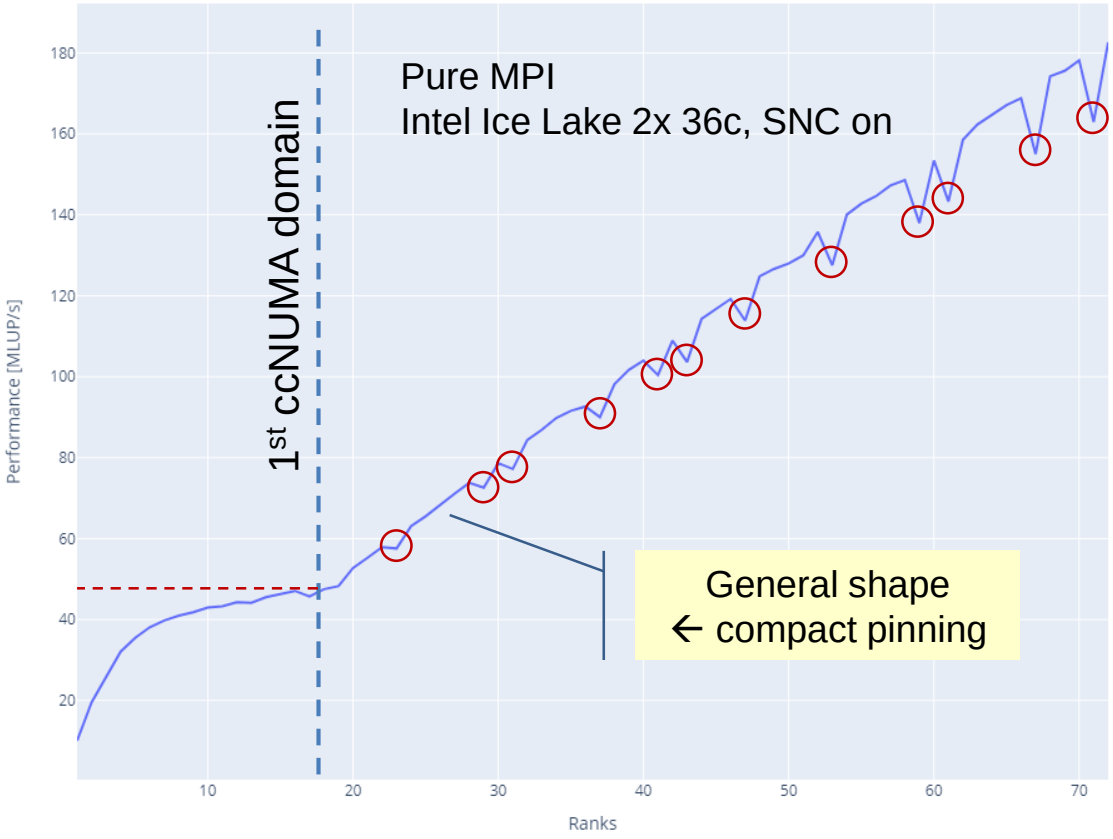
Lowest order: typical memory-bound code

- Saturation on 1st ccNUMA domain
- linear scaling beyond

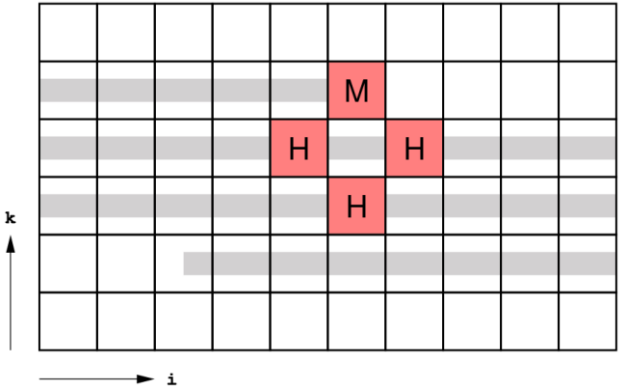
However…

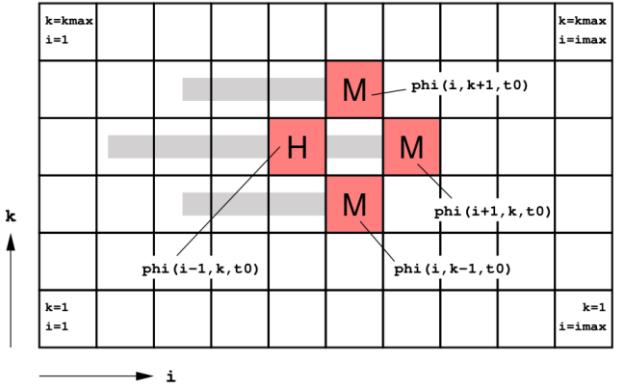
- No “hard” saturation
- Dips at #processes = prime???

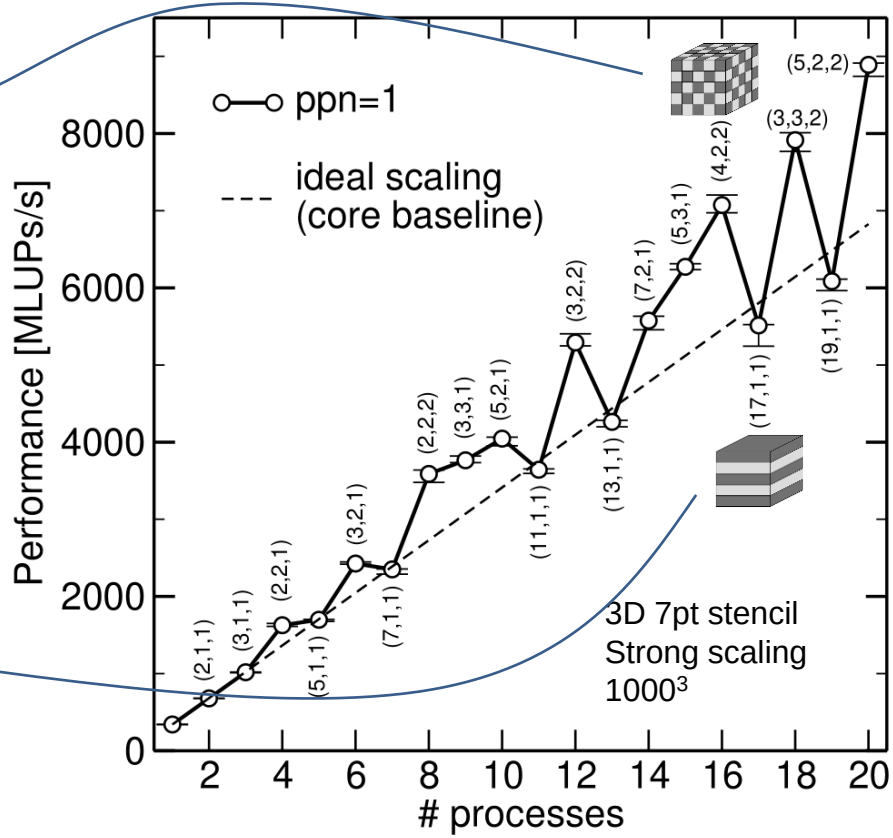
Wait. Prime???



Layer conditions and domain decomposition







However…

- Overall problem size is ~15300^2
- Hotspots show simple 2D stencil patterns
- E.g., one `advec_mom_kernel` loop nest:

$$15300 \times 2\ (\text{arrays}) \times 2\ (\text{layers}) \times 8\ \text{bytes} \approx 490\ \text{kB}$$

- … and Cloverleaf cuts the inner dimension anyway ☹

`advec_mom_kernel()`, line 222

```
DO k=y_min,y_max+1
  DO j=x_min-1,x_max+2
    ! Staggered cell mass post advection
    node_mass_post(j,k)=0.25_8*(density1(j  ,k-1)*post_vol(j  ,k-1) &
      +density1(j  ,k  )*post_vol(j  ,k  )                          &
      +density1(j-1,k-1)*post_vol(j-1,k-1)                          &
      +density1(j-1,k  )*post_vol(j-1,k  ))
    node_mass_pre(j,k)=node_mass_post(j,k)-node_flux(j-1,k)+node_flux(j,k)
  ENDDO
ENDDO
```

Still… memory traffic?

Memory data traffic at hotspots shows distinct patterns

- Overall drop along 1st and 2nd ccNUMA domain (> 30%)
- More traffic @ prime number of processes

Enter SpecI2M!

Intel SpecI2M – write-allocate evasion

- Hot Chips ‘20 Conference: https://www.hotchips.org/assets/program/conference/day1/HotChips2020_Server_Processors_Intel_Irma_ICX-CPU-final3.pdf

SpecI2M optimization: Convert RFO to specI2M when memory subsystem is heavily loaded

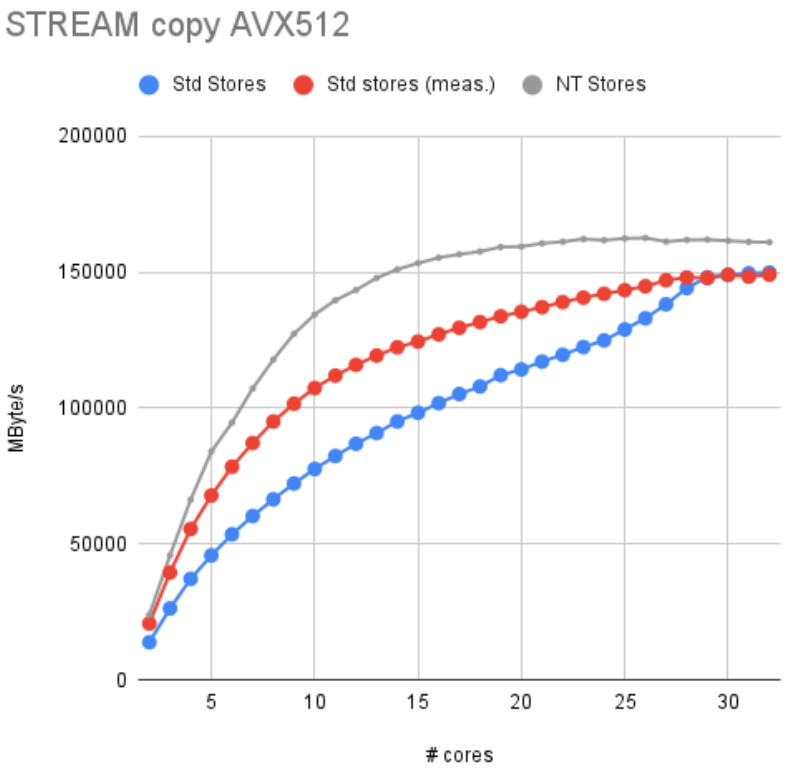
- Reduces mem bandwidth demand on streaming WLs that do full cache line writes (25% efficiency increase)

- John McCalpin: https://community.intel.com/t5/Software-Tuning-Performance/ICX-What-is-SpecI2M-request-and-how-it-differs-from-RFO/td-p/1204258

SpecI2M write-allocate evasion

Conditions

- Gradually kicks in on the way to saturation
  - Explains slow drop in read traffic towards saturation
- Long loops with no significant gaps (e.g., halo layers)
  - Explains traffic spikes at prime #procs



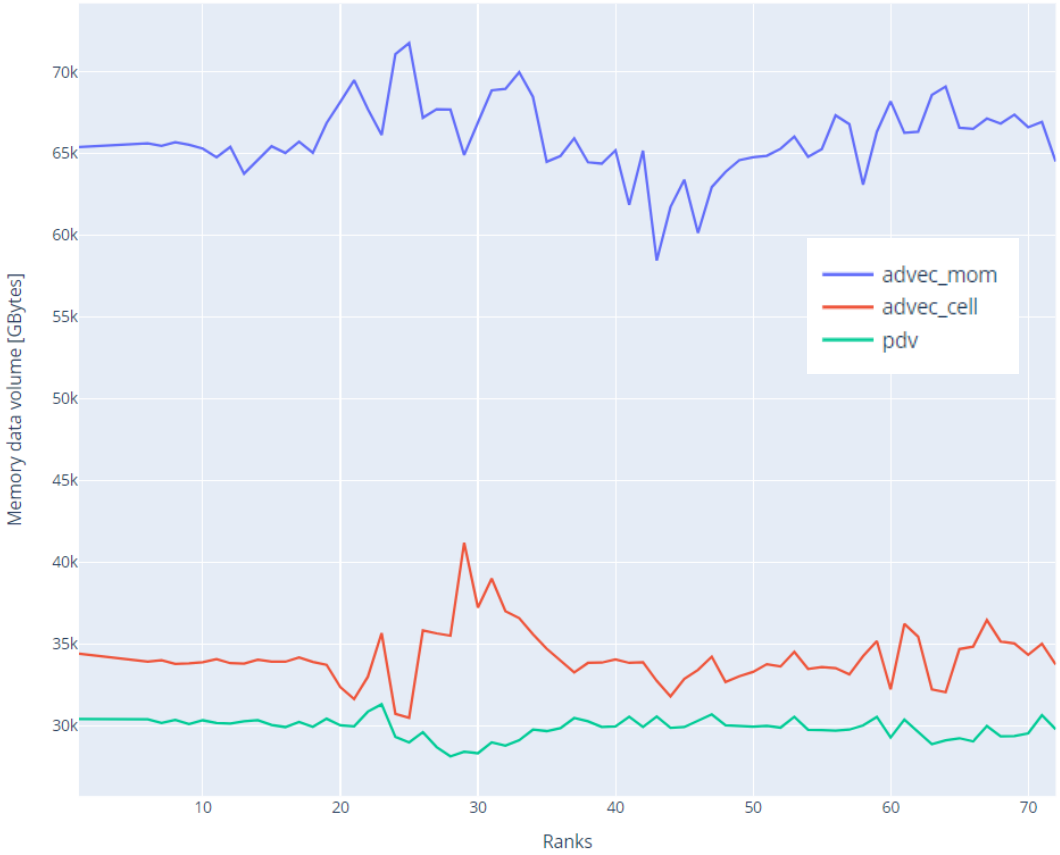
Cross-check: deactivate SpecI2M

MSR bit disclosed by Intel under NDA

- Prime number pattern gone
- Drop along 1st socket gone
- General traffic volume in accordance with analytic model

What about NT stores?

- Ask me in private



NHR FAU

Friedrich-Alexander-Universität
Erlangen-Nürnberg

“If you just dig deep enough, things get juicy”

Thank you.