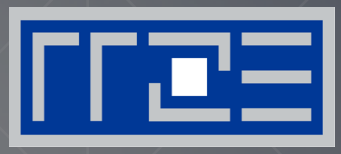
ERLANGEN REGIONAL COMPUTING CENTER



Jobspezifisches Monitoring
Anforderungen und Erwartungen

Jan Eitzinger

Übergeordnetes Ziel

Verbesserung der effizienten Nutzung von HPC Systemen!

Wie kann man Effizienz messen?

Unsere Sicht

Wie gut nutzt ein Applikationscode die angebotenen Resourcen einer Hardware Platform aus?

HPC Systeme bieten folgende Ressourcen

- Instruktionsdurchsatz
- Datentransfers (Bandbreite)
  - Caches, Hauptspeicher
  - Kommunikationsnetzwerk
  - File IO

Meist wird eine Applikation durch eine der obigen Resourcen beschränkt sein, man spricht von **Bottlenecks**.

Schwierigkeit

Eine belastbare Aussage über die Effizienz kann nur für ein spezifisches Programm auf einer spezifischen Hardwareplatform getroffen werden.

Dazu sind unter anderem notwendig

- Algorithmische Analyse
- Statische Code Analyse
- Mikrobenchmarking
- Applikationsbenchmarking
- Hardware Performance Monitoring (HPM) Messungen

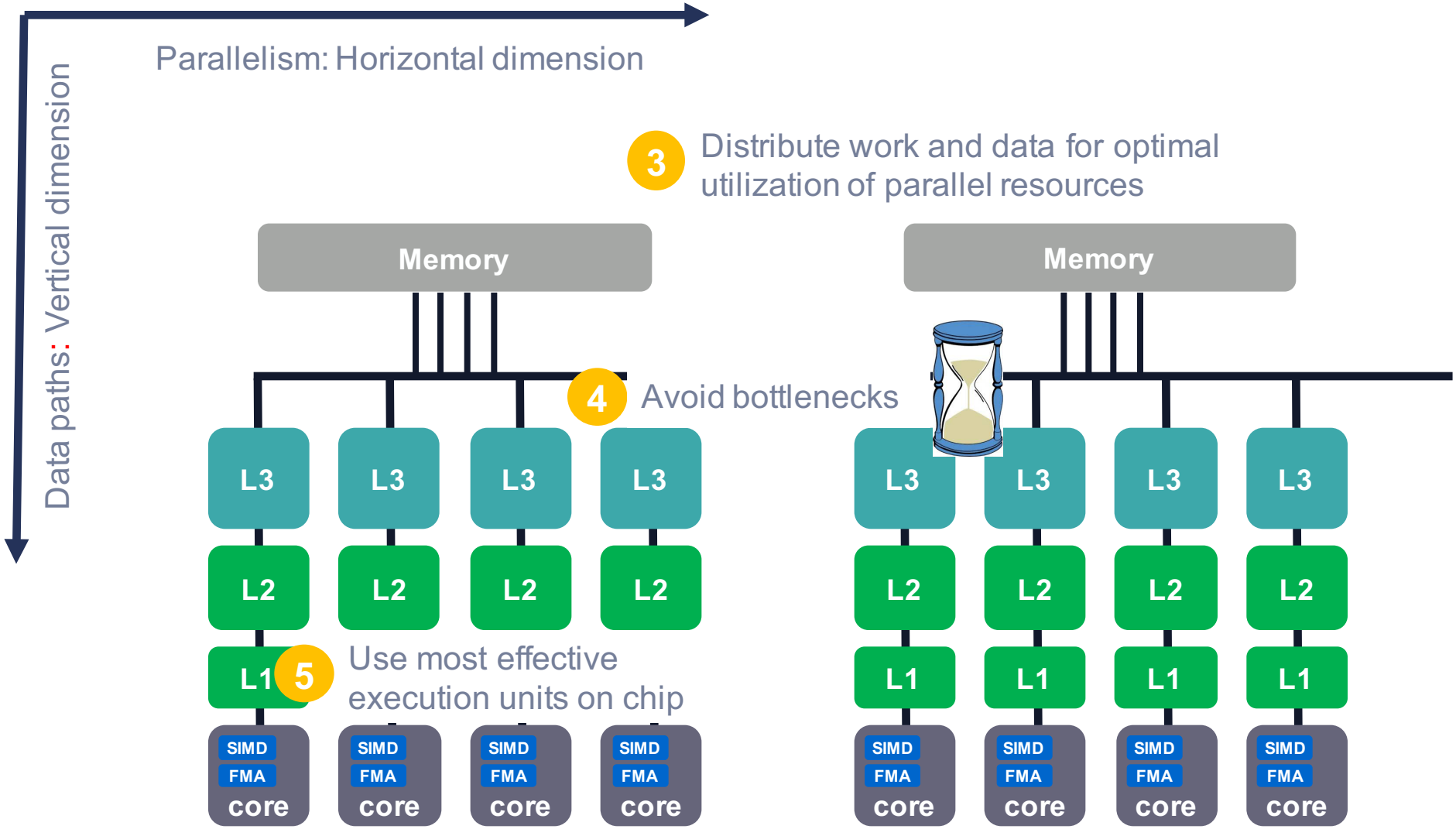
Ein automatisches HPM basiertes Job Performance Monitoring kann nur Extremfälle sicher erkennen oder allenfalls erste Hinweise über die Effizienz einer Anwendung geben.

Performance Engineering Tasks: Software

Optimizing software for a specific hardware requires to align several orthogonal targets and can only be successful employing an iterative process.

Software side: Reduce algorithmic and processor **work**

Performance Engineering Tasks: Hardware



Thinking in bottlenecks

- A bottleneck is a performance limiting setting
- Microarchitectures expose numerous bottlenecks

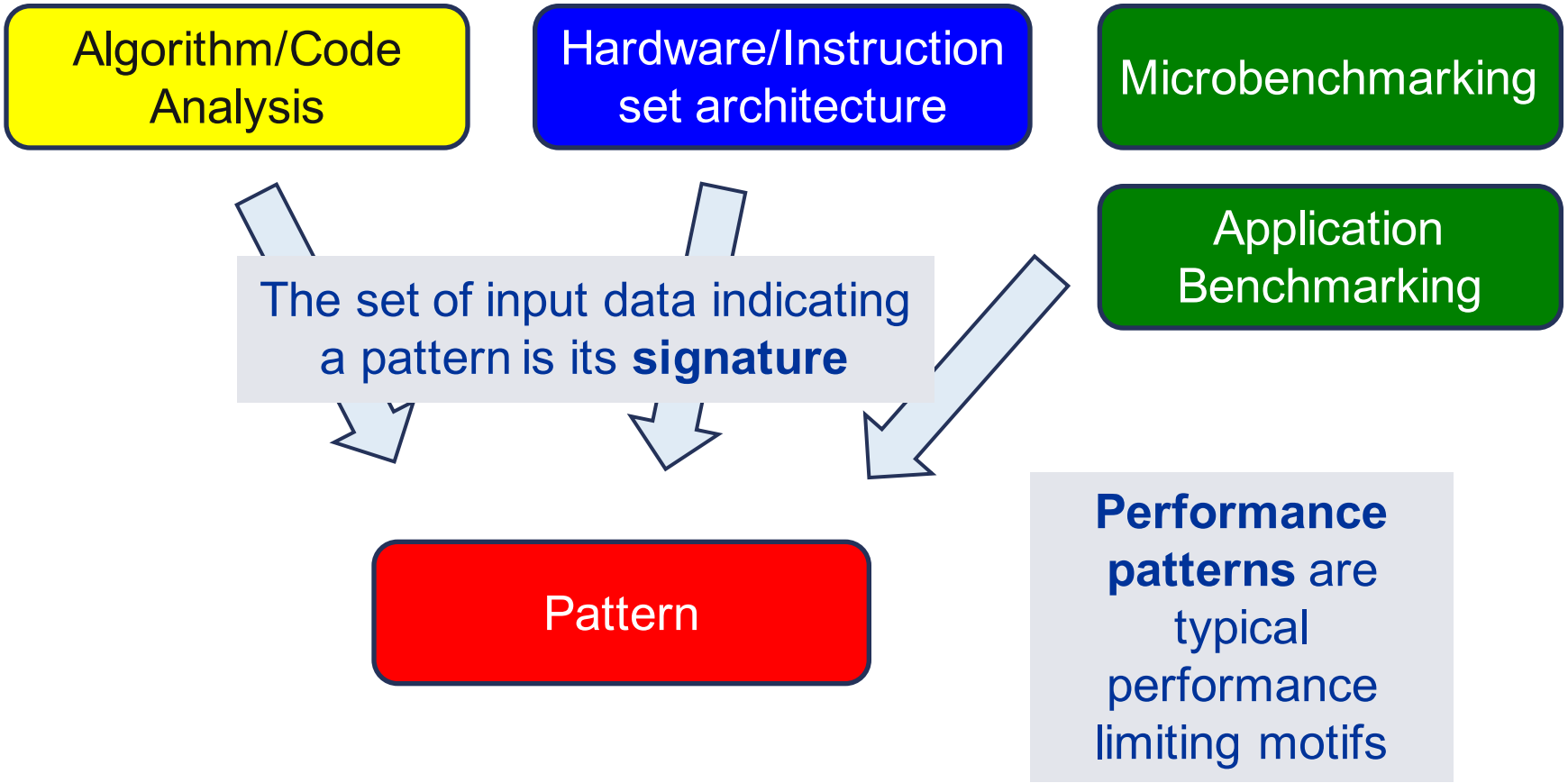
Observation 1:

Most applications face a single bottleneck at a time!

Observation 2:

There is a limited number of relevant bottlenecks!

Performance Engineering Process: Analysis



Step 1 **Analysis:** Understanding observed performance

Or Formulate a performance expectation

Performance Muster Klassifizierung

1. Maximum resource utilization
   (computing at a bottleneck)

2. Optimal use of parallel resources

3. Hazards
   (something “goes wrong”)

4. Use of ineffective instructions

5. Work related
   (too much work or too inefficiently done)

Patterns (I): Bottlenecks & parallelism

Pattern	Performance behavior	Metric signature, LIKWID 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)
ccNUMA utilization	Bad or no scaling across NUMA domains, performance improves with interleaved page placement	Unbalanced bandwidth on memory interfaces / High remote traffic (MEM)
Load imbalance / serial fraction	Saturating/sub-linear speedup	Different amount of “work” on the cores (FLOPS_*); note that instruction count is not reliable!

Patterns (II): Hazards

Pattern	Performance behavior	Metric signature, LIKWID performance group(s)
False sharing of cache lines	Large discrepancy from performance model in parallel case, bad scalability	Frequent (remote) CL evicts (CACHE)
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)
Micro-architectural anomalies	Large 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
Latency-bound data access	Simple bandwidth performance model much too optimistic	Low BW utilization / Low cache hit ratio, frequent CL evicts or replacements (CACHE, DATA, MEM)

Patterns (III): Work-related

Pattern		Performance behavior	Metric signature, LIKWID performance group(s)
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)
Excess data volume		Simple bandwidth performance model much too optimistic	Low BW utilization / Low cache hit ratio, frequent CL evicts or replacements (CACHE, DATA, MEM)
Code composition	**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)

Patterns conclusion

- Pattern signature = performance behavior + hardware metrics
  - Hardware metrics alone are almost useless without a pattern

Monitoring: Hardware metrics only!

- Patterns are applied hotspot (loop) by hotspot

Monitoring: Granularity may be too coarse!

- Patterns map to typical execution bottlenecks

Consequences for Monitoring

- **Resource utilization** is the only measure for efficiency
- If an observed performance is **good** or **bad** can only be decided with an in-depth analysis
- To provide a diagnosis of non-optimal resource utilization **performance patterns** are a useful systematic
- A certain performance behavior is only valid for homogeneous code, usually loop bodies

Tasks for Performance Monitoring

- Map resource categories and performance patterns on validated event sets
- Provide heuristics which performance bounds are acceptable
- Coarse measurement intervals will limit accuracy of analysis

Überblick: Technische Komponenten

Technische Anforderungen an Monitoring Infrastruktur

Konzentration auf **Overhead** und **Performance**

- Overhead kritisch bei **Datenerfassung** und **Datensammlung**

- Faktoren für **Performance**
  - Skalierbarkeit bei Datensammlung und Speicherung
  - Skalierbarkeit der Webschnittstelle

Beispielrechnung Datenvolumen

Knotenanzahl	1000 (100 – 5000)	Faktor 5
Threads pro Knoten	40 (20 – 240)	Faktor 6
Anzahl Metriken	20 (10 – 40)	Faktor 2
Messfrequenz	60s (10s – 10m)	Faktor 6
Datentyp float	4 byte	

Datenvolumen pro Tag / Woche bei Erfassung auf
Knotengranularität: 115 MB/ 806 MB

Datenvolumen pro Tag / Woche bei Erfassung auf
Threadgranularität: 4.6 GB / 32 GB