FEPA – A framework for systematic energy and performance analysis of extreme-scale applications in HPC computing centers

J. Treibig

RRZE

a joint project with:

lrz NEC

Bundesministerium
für Bildung
und Forschung

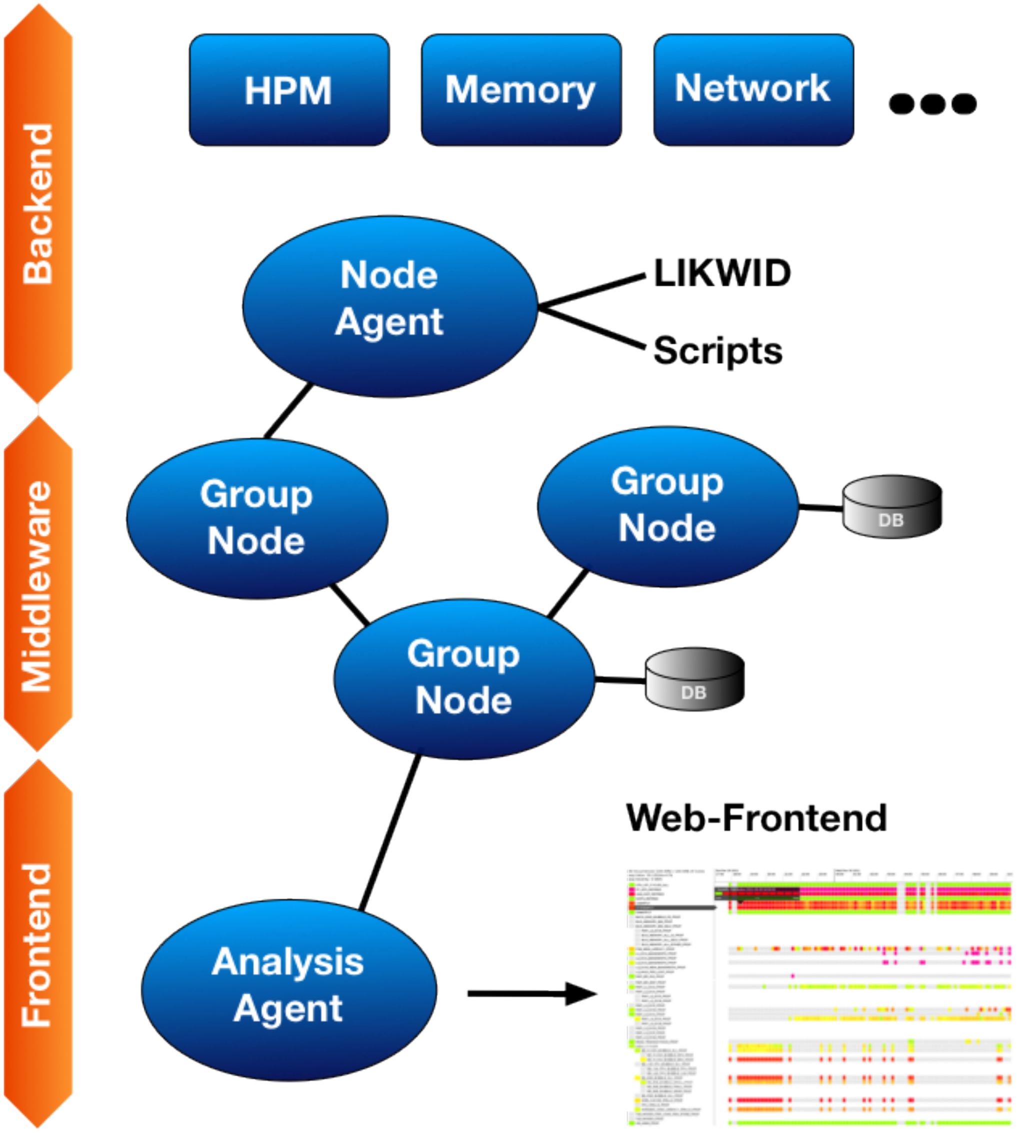
Goals

- Provide **tooling infrastructure** which allows to globally profile application performance in large supercomputing centers

- Embed application profiling in a **pattern-driven performance engineering** process aiming for maximum resource utilization

- Provide knowledge which enables to significantly improve the **efficient use** of HPC compute resources across all application domains

Technical Project Overview



Builds upon the results of previous BMBF projects:

- ISAR (LRZ)
- TIMaCS (NEC)

Opportunity to establish the LIKWID Open Source project as an alternative to established solutions

All components will be Open Source and can also be used stand alone

Philosophy

- **Motivated by a resource driven view**
- **Provide a structured iterative process based on:**
  - Performance patterns
  - A diagnostic performance model
- **Performance patterns are typical performance limiting bottlenecks**
- **Patterns are indicated by signatures which can consist of:**
  - HPM data
  - Scaling behavior
  - Other data
- **Uses one of the most powerful tools available:**

Your brain !

You are a investigator making sense of what's going on.
And there is no alternative to that.

Performance Pattern Classification

1. Maximum Resource utilization
2. Hazards
3. Work related (Application or Processor)

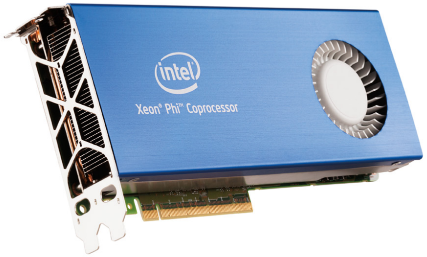
The system offers two basic resources:

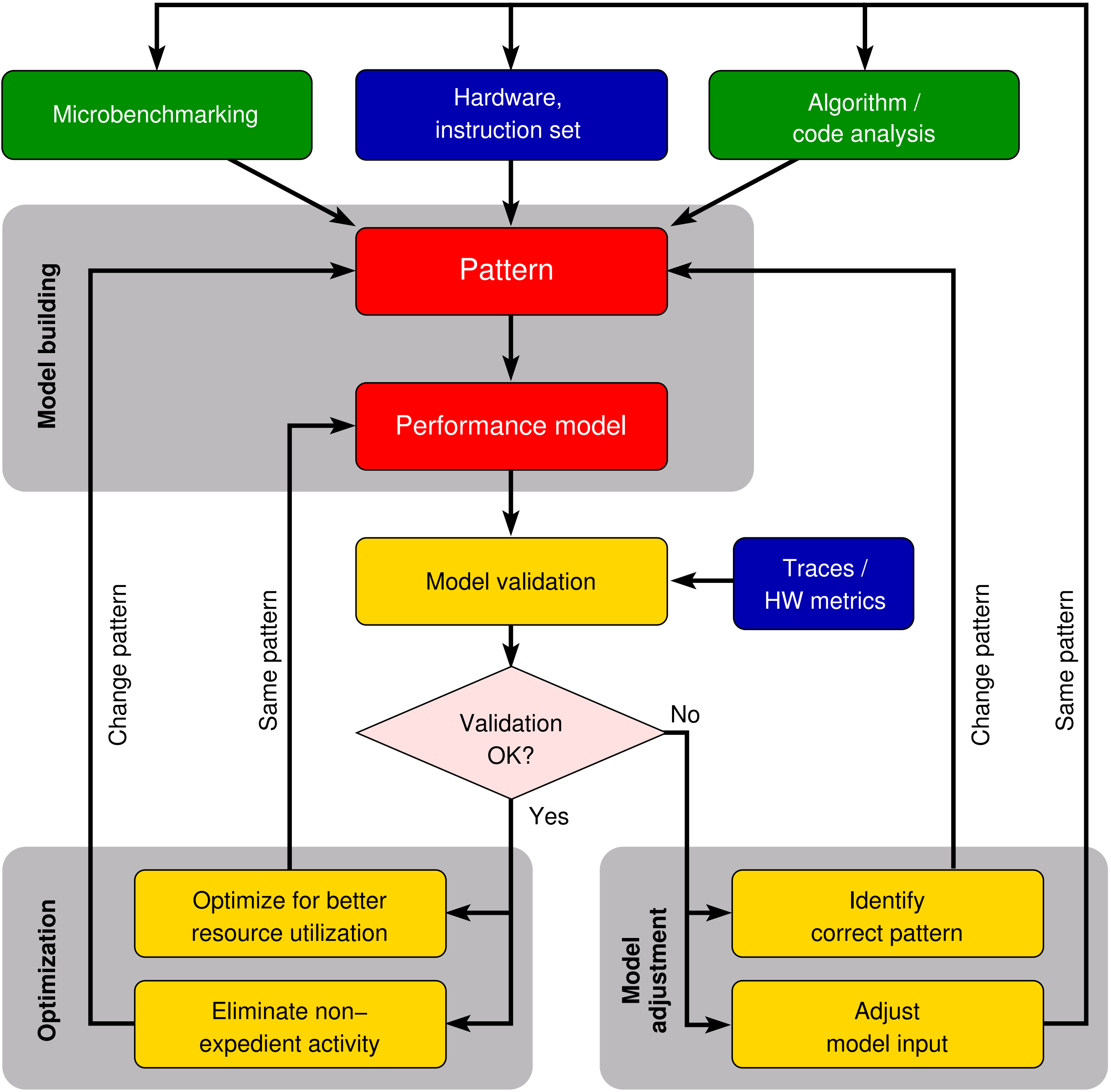
- **Execution of instructions** (primary)
- **Transferring data** (secondary)

A good architectures allows you to fully exploit the design capabilities without road blocks or detours.

SSE, AVX, AVX2

Alignment/Gather





Notions of work

- Application work
- Processor work

Pattern: qualitative

↕

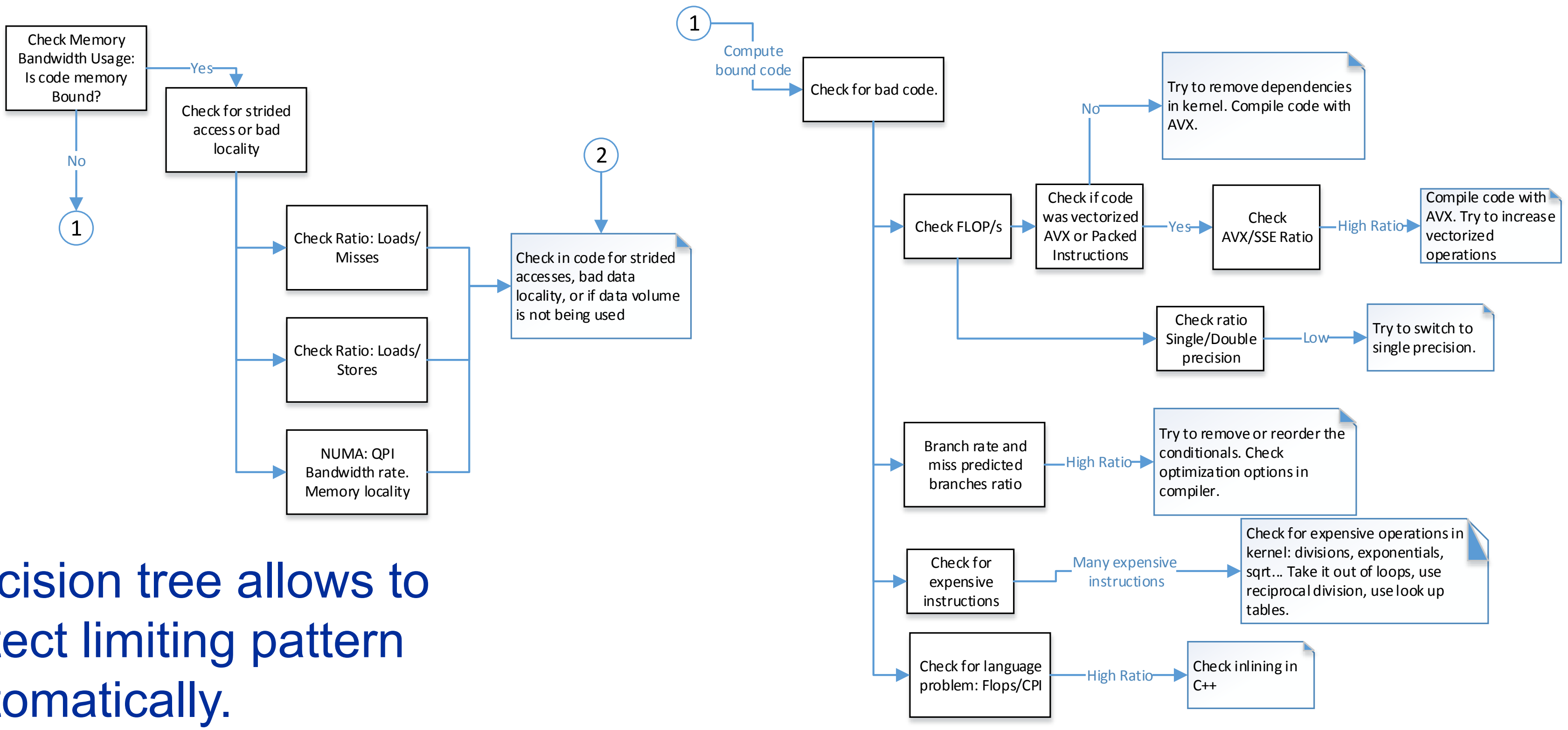
Model: quantitative

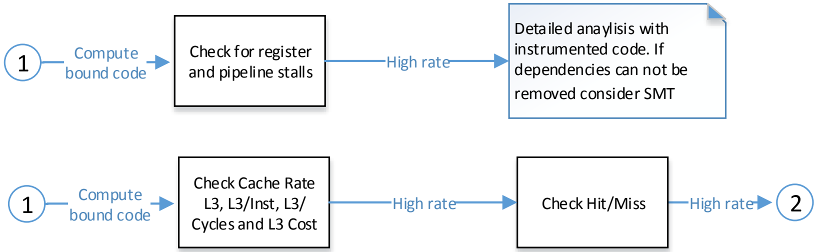
Find the relevant limiting bottleneck!

Overview Performance Patterns

Pattern		Behavior
Bandwidth saturation		saturating speedup across cores sharing a data path
Limited Instruction throughput	Pipeline saturation	throughput at design limit
	Pipelining hazards	in-core throughput far from design limit, performance insensitive to data size
	Control flow issues	
Inefficient data access	Strided Access	simple BW models far too optimistic
	Erratic Access	
Microarchitectural anomalies		large discrepancy from simple performance models
False cacheline sharing		very low speedup, or slowdown / discrepancy from model only in parallel case
Bad ccNUMA page placement		bad/no scaling across locality domains, better performance w/ interleaved placement
Load imbalance		saturating/sub-linear speedup
Synchronization overhead		speedup going down as more cores are added / no speedup with small problem sizes
Code composition issues	Instruction overhead	low application performance, good scaling across cores, performance insensitive to problem size
	Expensive instructions	
	Ineffective instructions	

Connection to application monitoring


Check Memory Bandwidth Usage: Is code memory Bound?
Yes
No
1
Check for strided access or bad locality
Check Ratio: Loads/ Misses
Check Ratio: Loads/ Stores
NUMA: QPI Bandwidth rate. Memory locality
2
Check in code for strided accesses, bad data locality, or if data volume is not being used
1
Compute bound code
Check for bad code.
Check FLOP/s
Check if code was vectorized AVX or Packed Instructions
No
Try to remove dependencies in kernel. Compile code with AVX.
Yes
Check AVX/SSE Ratio
High Ratio
Compile code with AVX. Try to increase vectorized operations
Check ratio Single/Double precision
Low
Try to switch to single precision.
Branch rate and miss predicted branches ratio
High Ratio
Try to remove or reorder the conditionals. Check optimization options in compiler.
Check for expensive instructions
Many expensive instructions
Check for expensive operations in kernel: divisions, exponentials, sqrt… Take it out of loops, use reciprocal division, use look up tables.
Check for language problem: Flops/CPI
High Ratio
Check inlining in C++


1
Compute bound code
Check for register and pipeline stalls
High rate
Detailed anaylisis with instrumented code. If dependencies can not be removed consider SMT
1
Compute bound code
Check Cache Rate L3, L3/Inst, L3/ Cycles and L3 Cost
High rate
Check Hit/Miss
High rate
2

Decision tree allows to detect limiting pattern automatically.

Estimate urgency.

Identify dangerous applications.

Conclusion

- FEPA will provide a low overhead framework which allows to measure system wide application performance/energy data
- The effective interpretation of the raw profiling data is enabled by introducing performance patterns
- The effectiveness of the approach will be evaluated at several Gauss member HPC centers

HPC is computing at a bottleneck

Thank you for your attention!

Any Questions?

Visit us tomorrow at the Poster reception, 5:15PM :

Pattern-Driven Node-Level Performance Engineering