Understanding Application Performance with the POP Metrics

Fabian Orland

IT Center, RWTH Aachen University

NHR4CES NHR for Computational Engineering Science

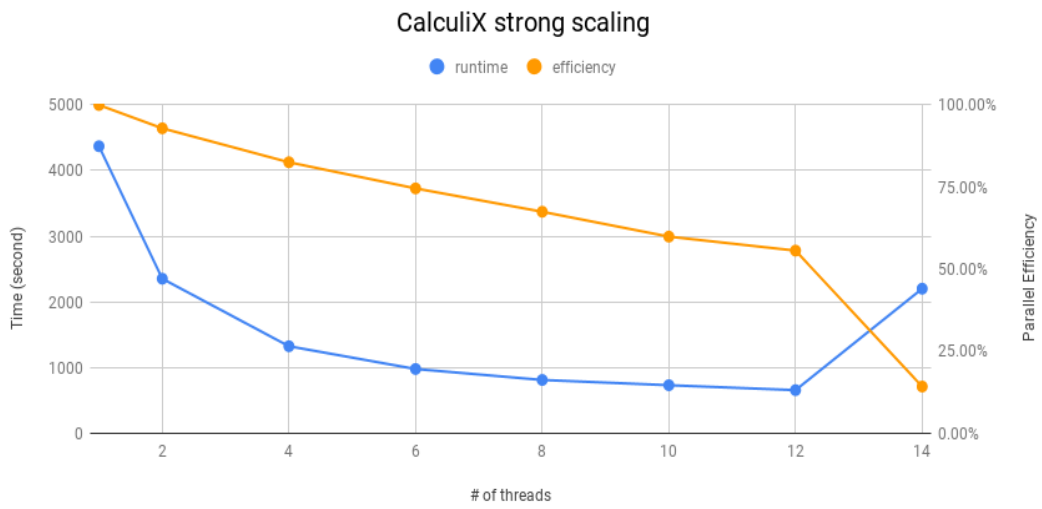
IT Center

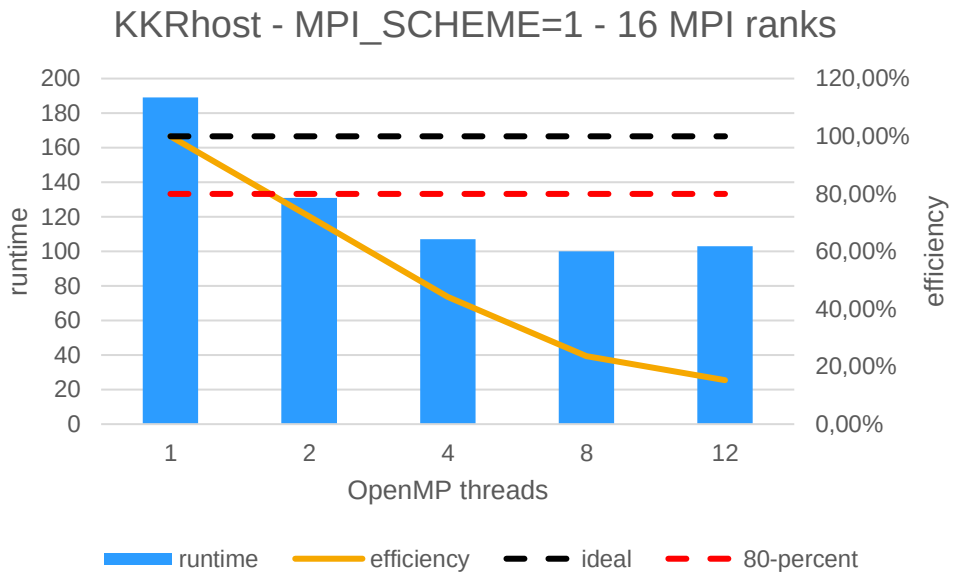
RWTH AACHEN UNIVERSITY

Overview

- Motivation
- POP Metrics
  - Standard POP metrics
  - Hybrid POP metrics
    - Additive hybrid metrics
    - Multiplicative hybrid metrics
    - Critical Path based metrics
- Case Studies / Success Stories
  - CalculiX
  - juKKR
- Summary

Motivation





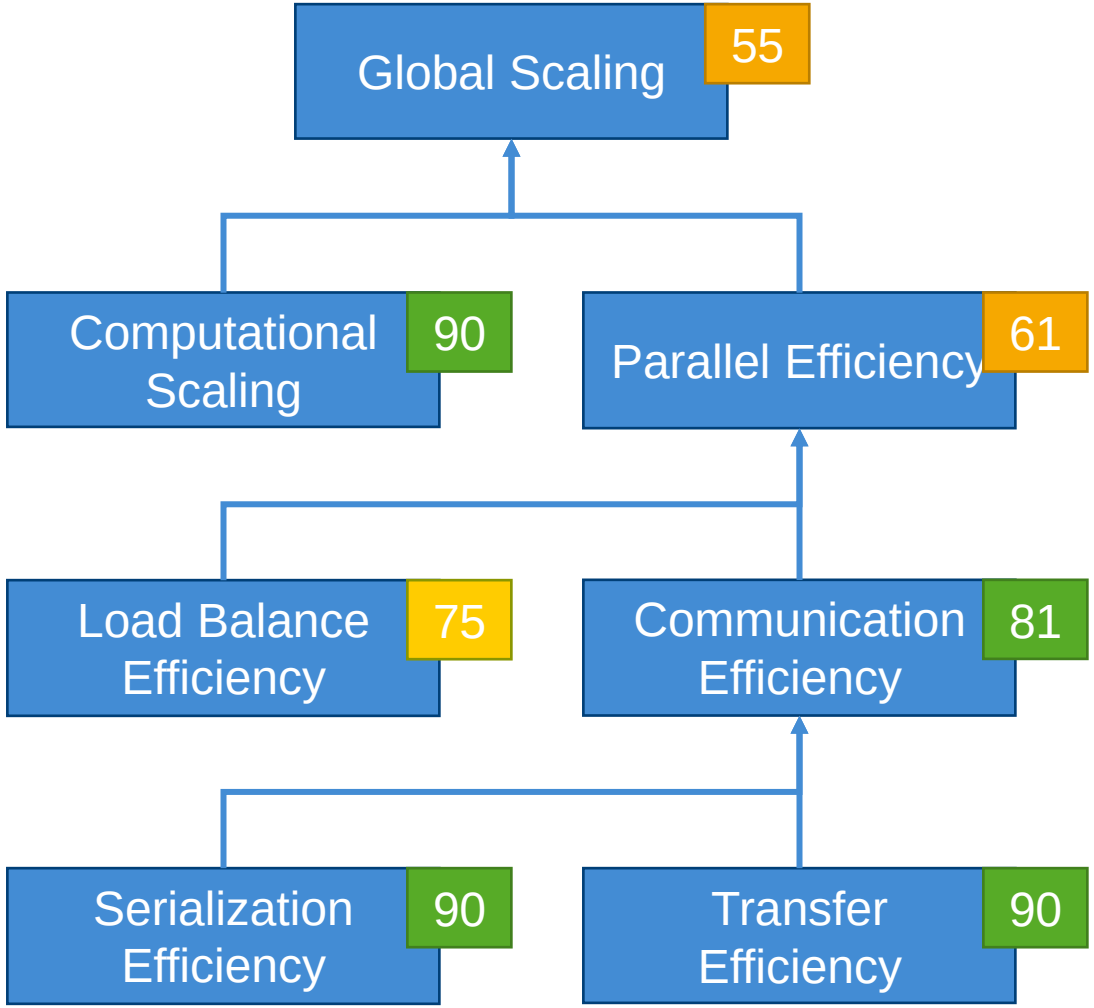
- Problem:
  - Why is my code getting inefficient at scale?
  - Multiple fundamental issues of (parallel) programming possible
- Solution: POP metrics
  - Standardized performance assessment independent of application / system
  - Easily verify performance improvements

Performance Optimisation and Productivity
A Centre of Excellence in HPC

NHR4CES NHR for Computational Engineering Science
IT Center
RWTH AACHEN UNIVERSITY

Standard POP metrics

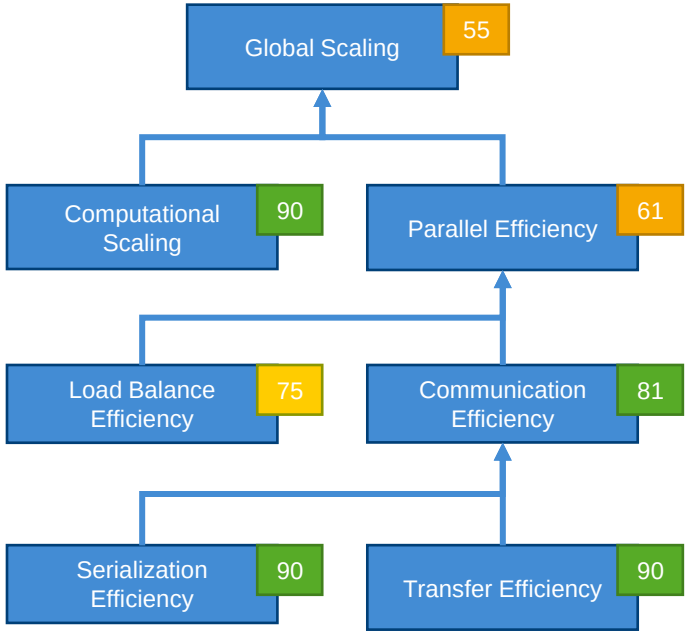
- Hierarchy of metrics
  - Aka *fundamental model factors*
  - Efficiency metrics ranging from 0 to 1
  - Child metrics multiply to yield parent metric by design
- Highlight issues in the parallel structure of an application
- Parallel Efficiency breaks down into
  - Load balance
  - Serialization
  - Transfer
- Computational Scaling captures impact of scaling to node-level performance

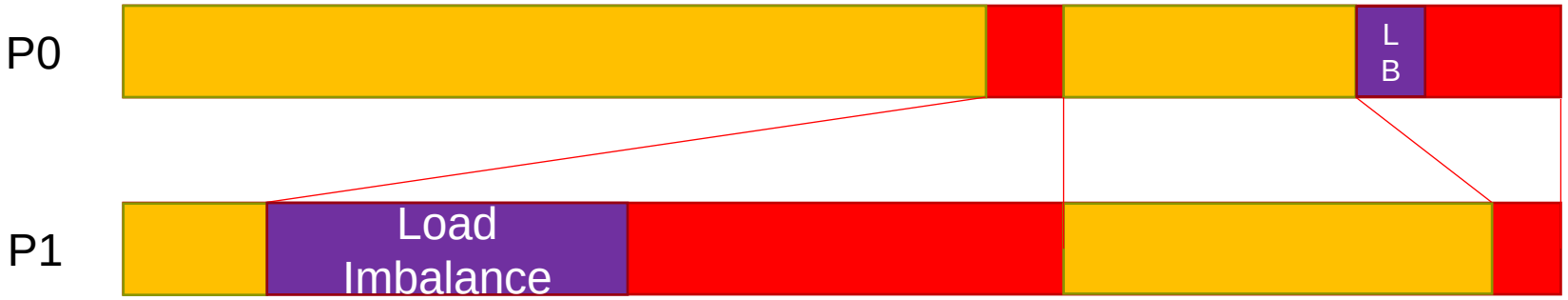


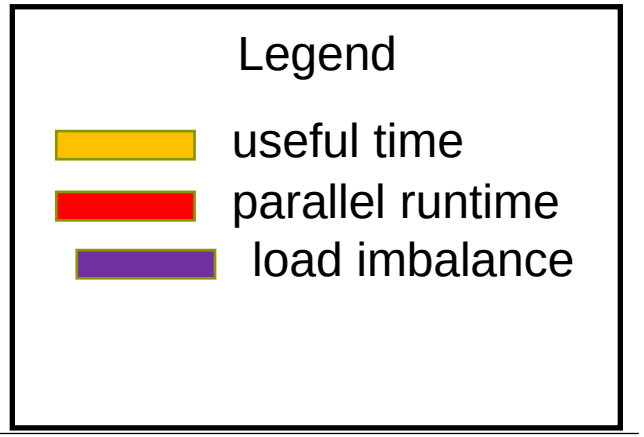
Standard POP metrics

Load Balance

- Reflects global imbalance of work between execution units
- $LB = \frac{avg(useful\ time)}{\max(useful\ time)}$
- *Useful time*: execution time outside parallel runtimes



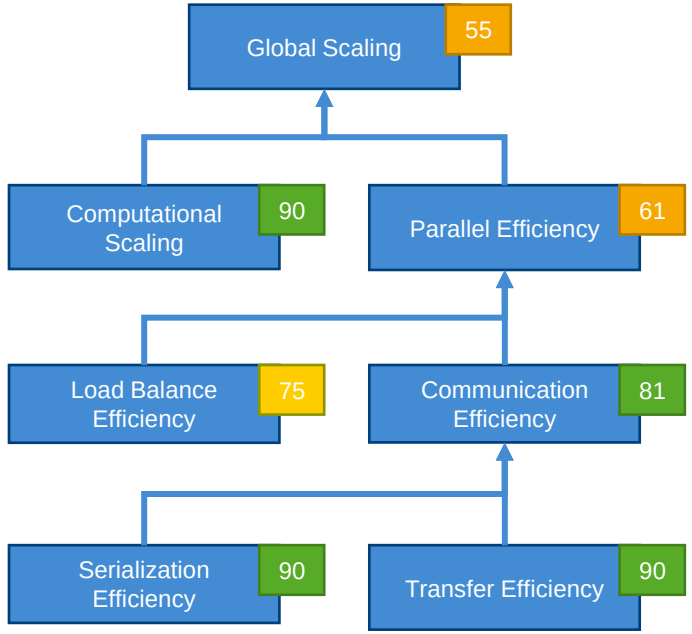


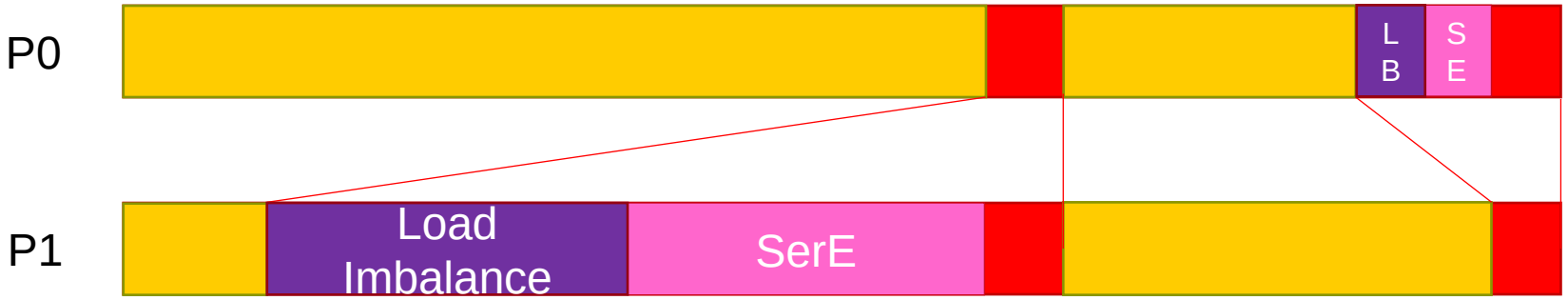


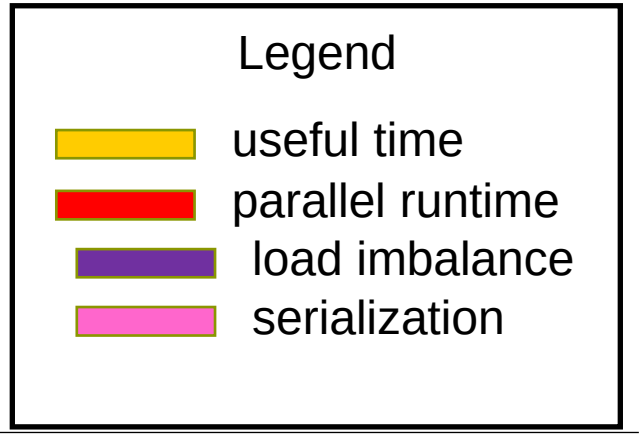
Standard POP metrics

Serialization Efficiency

- Reflects moving imbalance of work between execution units, resp., alternating dependencies
- $SerE = \frac{max(useful\ time)}{ideal\ runtime}$
- *Ideal runtime*: execution time on an ideal machine with 0 communication cost (inf. BW / 0 lat)



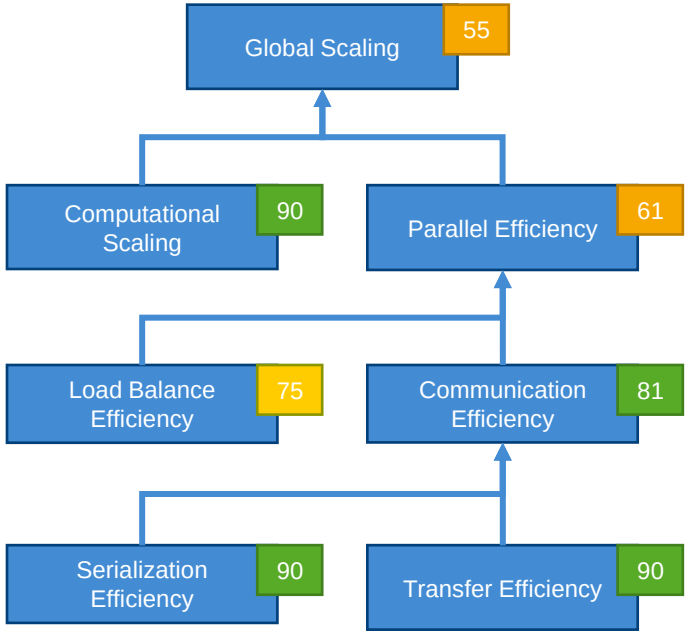


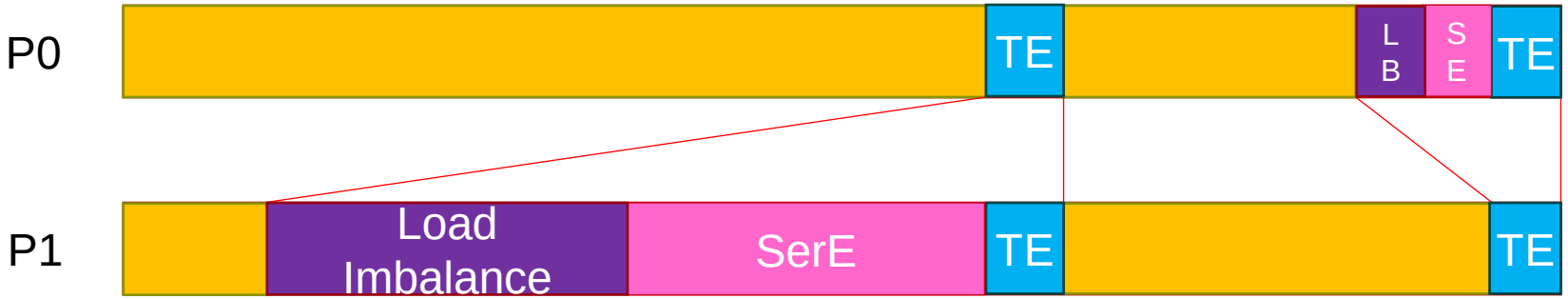


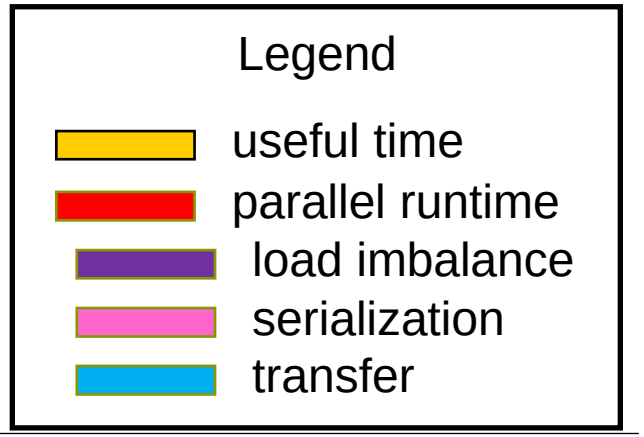
Standard POP metrics

Transfer Efficiency

- Cost of transfer/communication/synchronization
- $TE = \frac{ideal\ runtime}{real\ runtime}$
- *Real runtime*: observed execution time







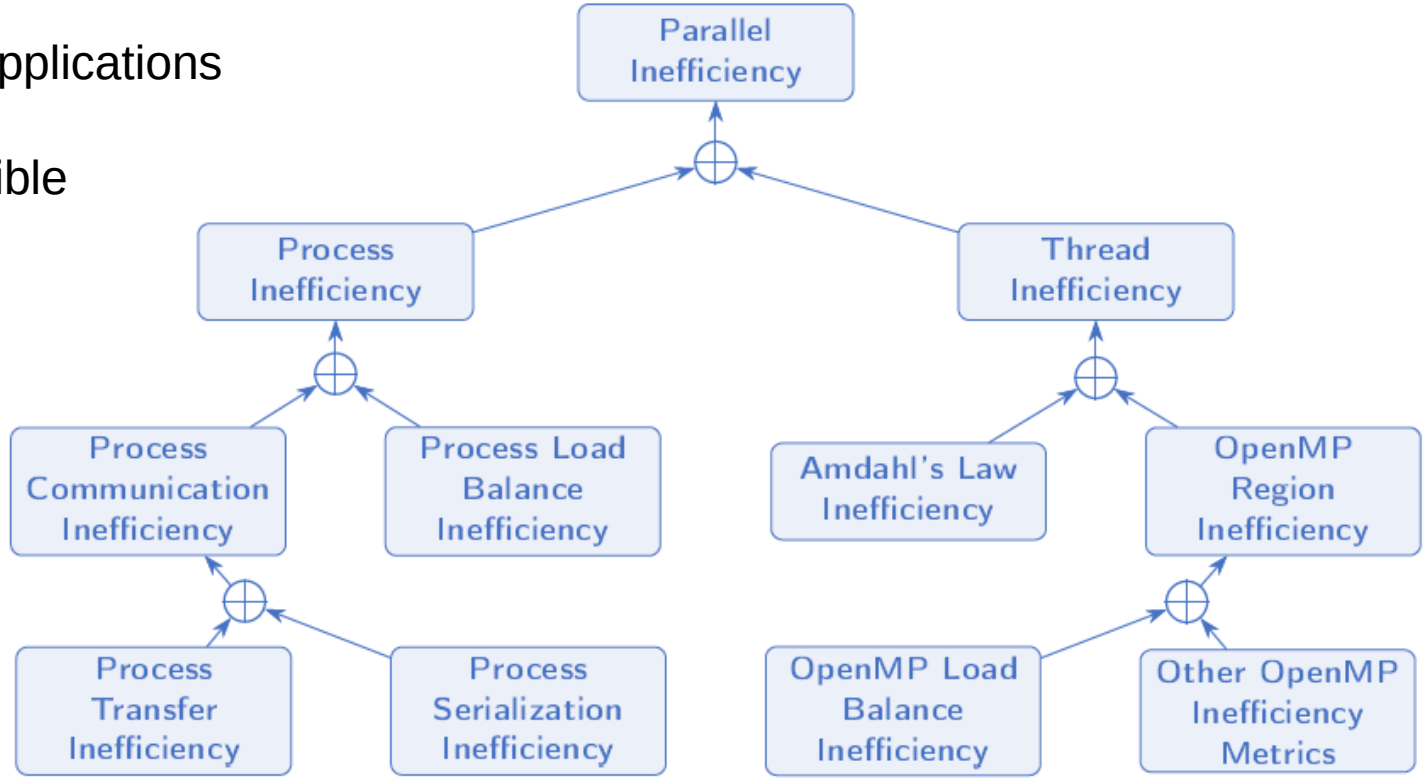
Computational Scaling

- In scaling experiments, node-level performance impacts the time metrics: useful time, ideal time, real time
- Computational scaling represents this impact
- Computational scaling and its factors are normalized to a baseline
- Impact factors:
  - IPC scaling
  - Instruction scaling
  - Frequency scaling
  - (Vectorization scaling)

Hybrid POP metrics

Additive model

- Currently only support for MPI+OpenMP applications
- Extension to MPI+CUDA(+OpenMP) possible
- Advantage: detailed OpenMP analysis
  - Easily compute metrics per OpenMP region
  - Identify which region(s) are most inefficient



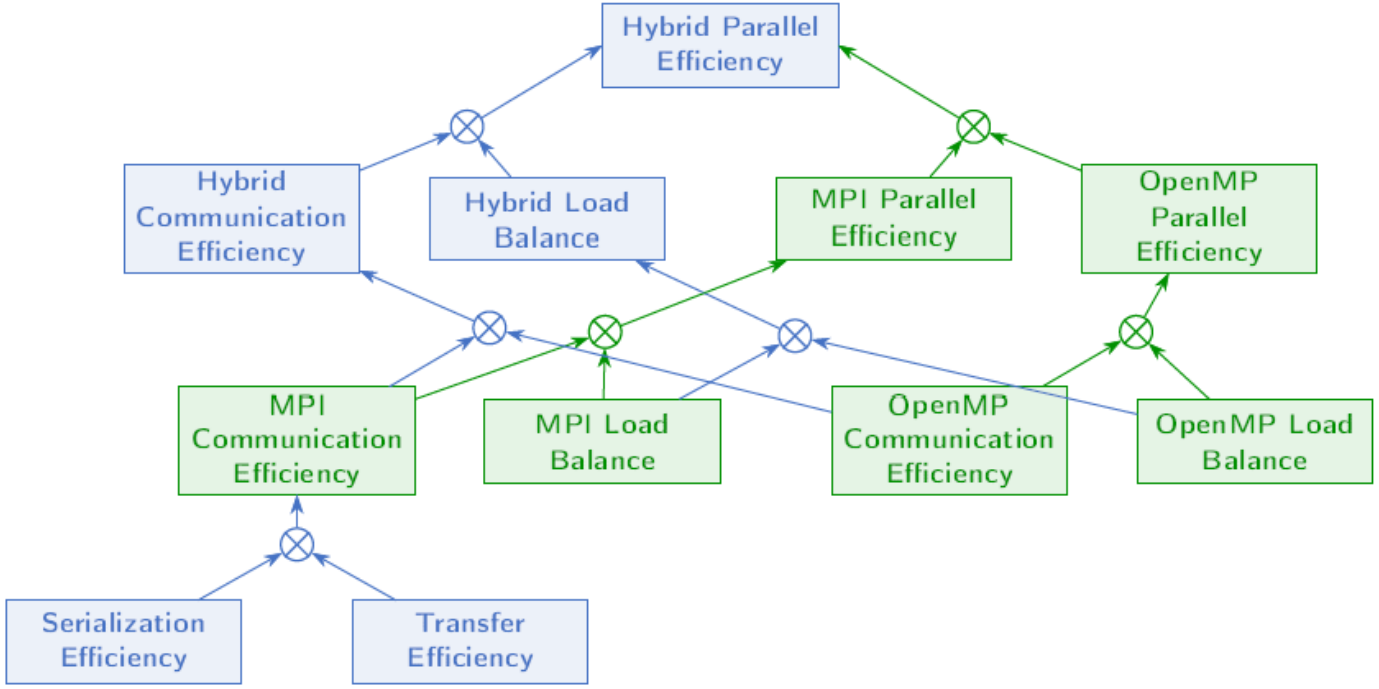
Details on how to compute these metrics can be found here: https://pop-coe.eu/further-information/learning-material

Hybrid POP metrics

Multiplicative model

- Support for hybrid MPI+X applications
  - X ∈ { OpenMP, CUDA, … }

- Idea:
  - Determine MPI efficiencies
  - Attribute remaining inefficienices to paradigm X

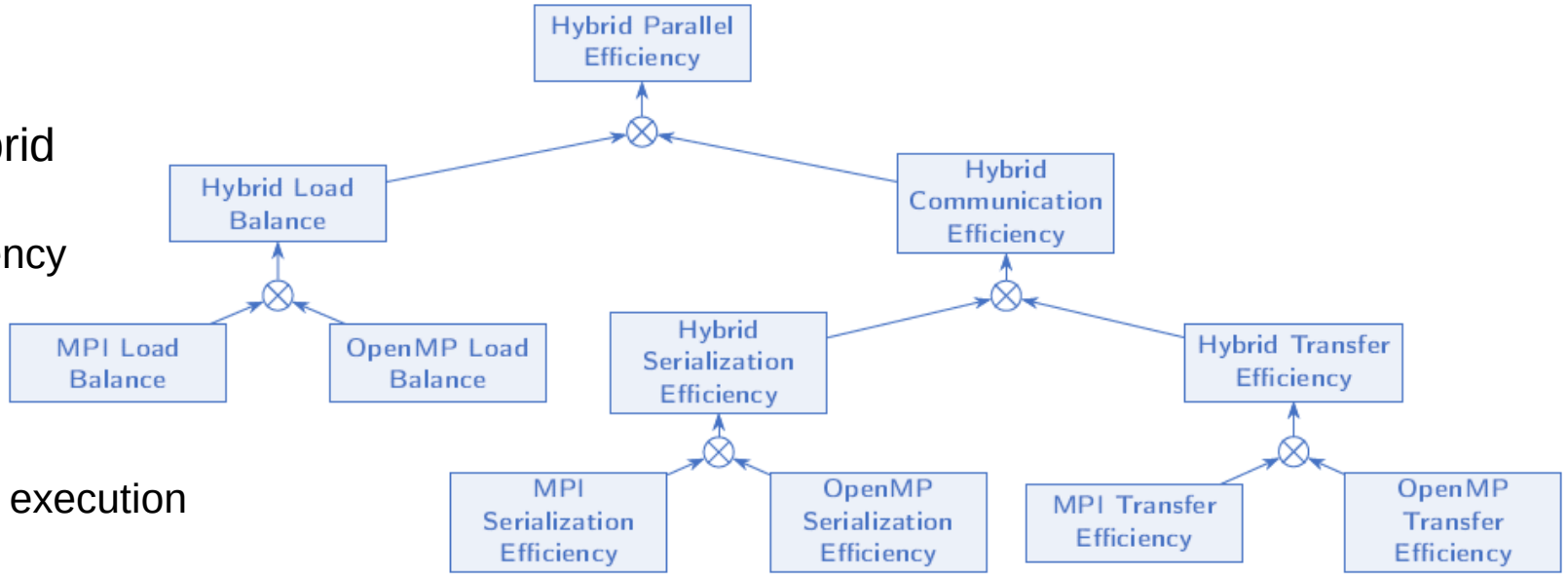


Details on how to compute these metrics can be found here: https://pop-coe.eu/further-information/learning-material

Hybrid POP metrics

Critical path-based model

- Generalization of multiplicative hybrid metrics
  - Hybrid split of Communication Efficiency into programming models



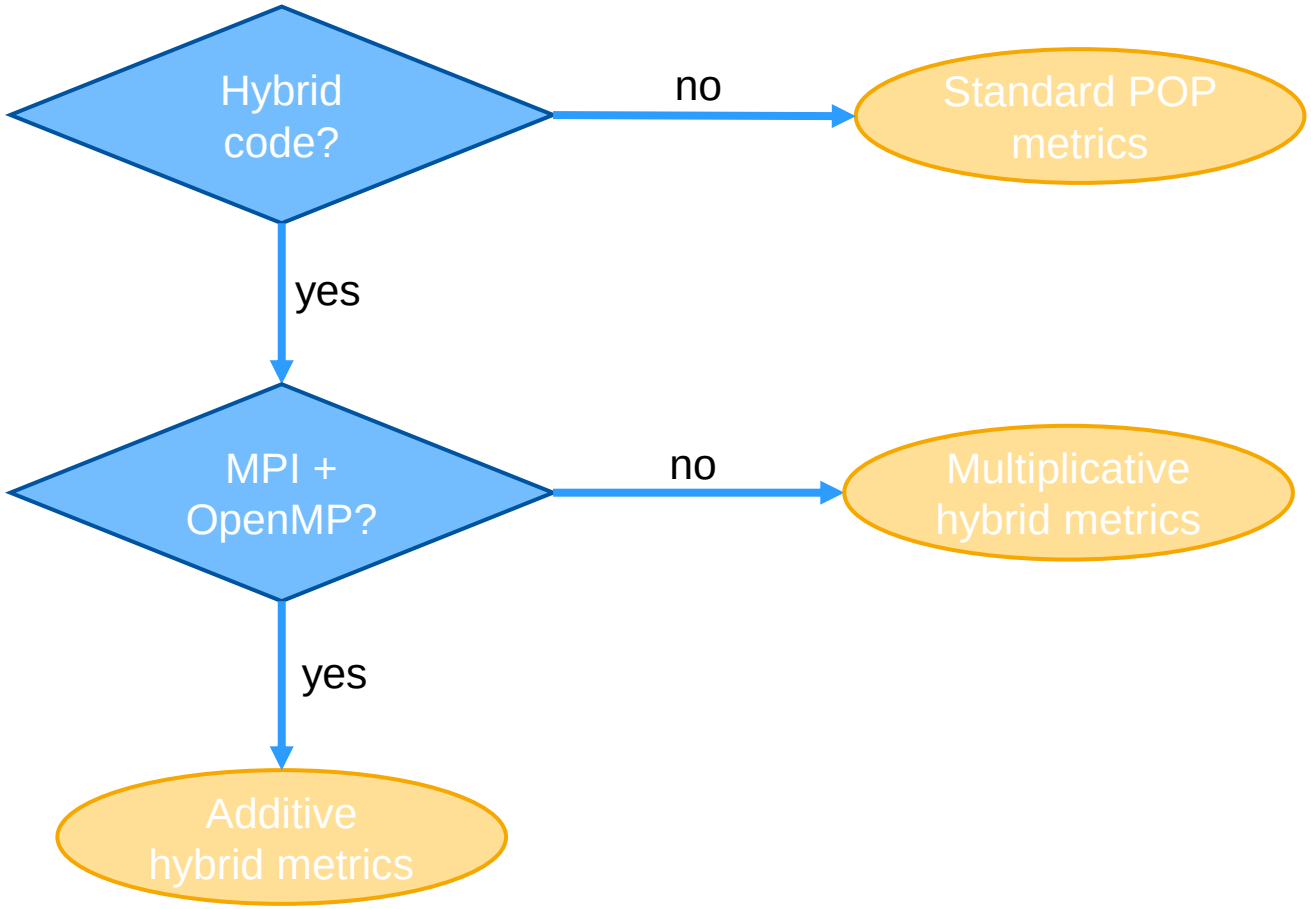
- Idea:
  - Critical path = event path in program execution with longest duration
  - $runtime_{ideal} \approx critical\ path\ of\ useful\ compution$

- Prototype tool for „on-the-fly“ calculation of hybrid metrics
  - Enables metric calculation for applications with non hierarchical communication (e.g. MPI-Detach with detached tasks)

Reference: J. Protze, F. Orland, K. Haldar, T. Koritzius, C. Terboven, „On-the-Fly Calculation of Model Factors for Multiparadigm Applications“, Euro-Par 2022

Hybrid POP metrics

Recommendations



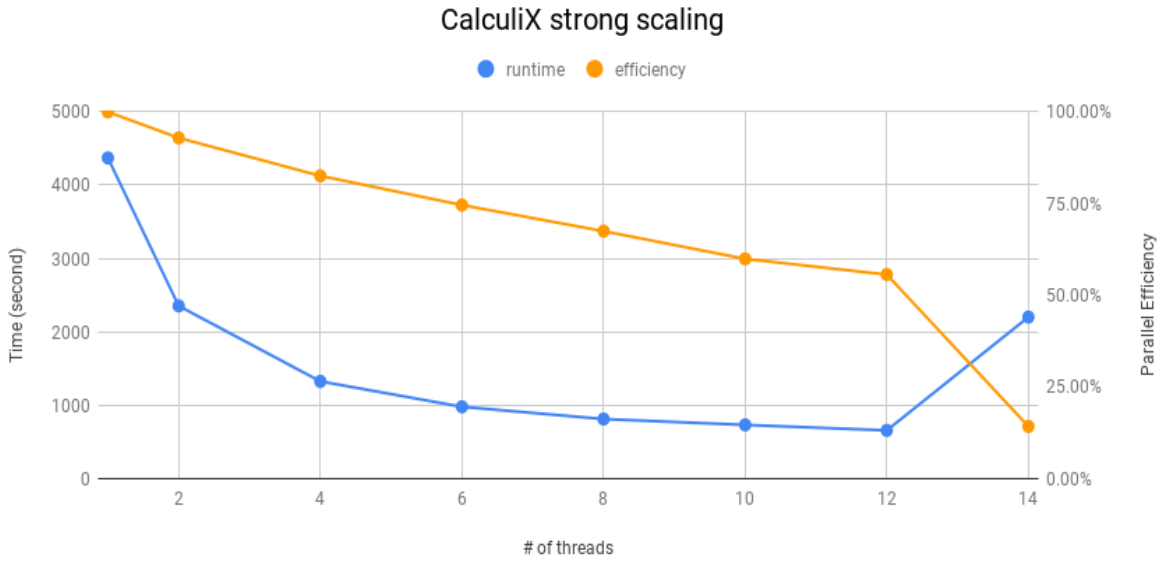
Case Studies

CalculiX - A Free Software Three-Dimensional Structural finite Element Program

- C+Fortran code parallelized with pthreads
- http://www.calculix.de/

Case Study: CalculiX

Performance Assessment



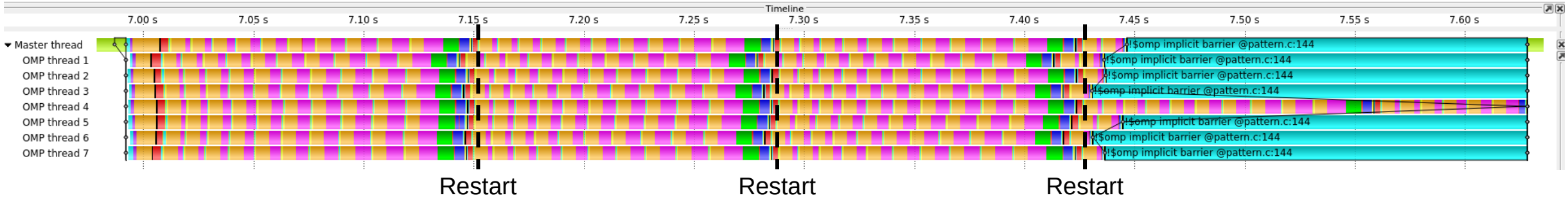
POP standard metrics for region: dgmres1mt
CalculiX

	2	4	6	8
Global Efficiency	0.8	0.85	0.88	0.89
Parallel Efficiency	0.8	0.76	0.75	0.75
Load Balance	0.81	0.77	0.76	0.76
Communication Efficiency	0.99	0.99	0.99	0.99
Serialization Efficiency	0.99	0.99	0.99	0.99
Transfer Efficiency	1.0	1.0	1.0	1.0
Computational Scaling	1.0	1.12	1.17	1.18
Instruction Scaling	1.0	1.06	1.08	1.08
IPC Scaling	1.0	1.06	1.08	1.09

- Issue with load balance identified by POP metrics
- Computations scale super-linear
  - Probably cache effects

Parallel GMRES Implementation

Trace for CalculiX-solver kernel with 8 OpenMP threads



Structure of GMRES solver:

- Matrix vector multiplication (orange)
- Orthogonalization (pink)
- Restart after every 10 inner iterations
- Each thread solves a small independent linear subsystem

Observation:

- Thread 4 has worse convergence of its subsystem solution compared to other threads
- If only 1/8 threads is actually computing ➔ load balance issue

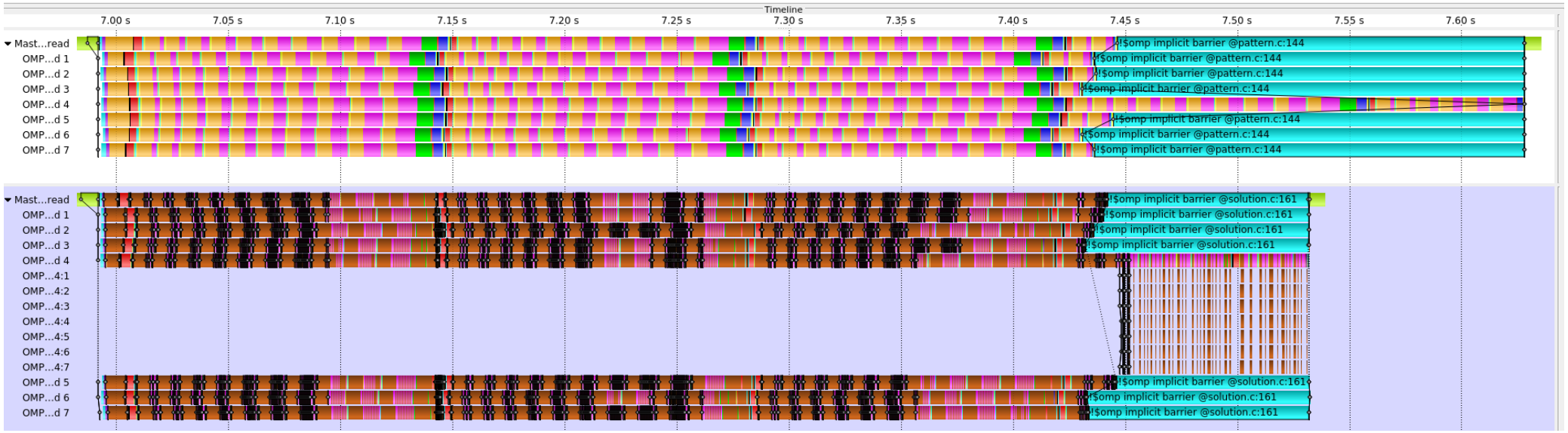
Case Study: CalculiX

Proof of Concept

- We extracted a single GMRES call from the application as a test kernel
- Idea: use nested parallelism to schedule additional threads on idling cores

original implementation

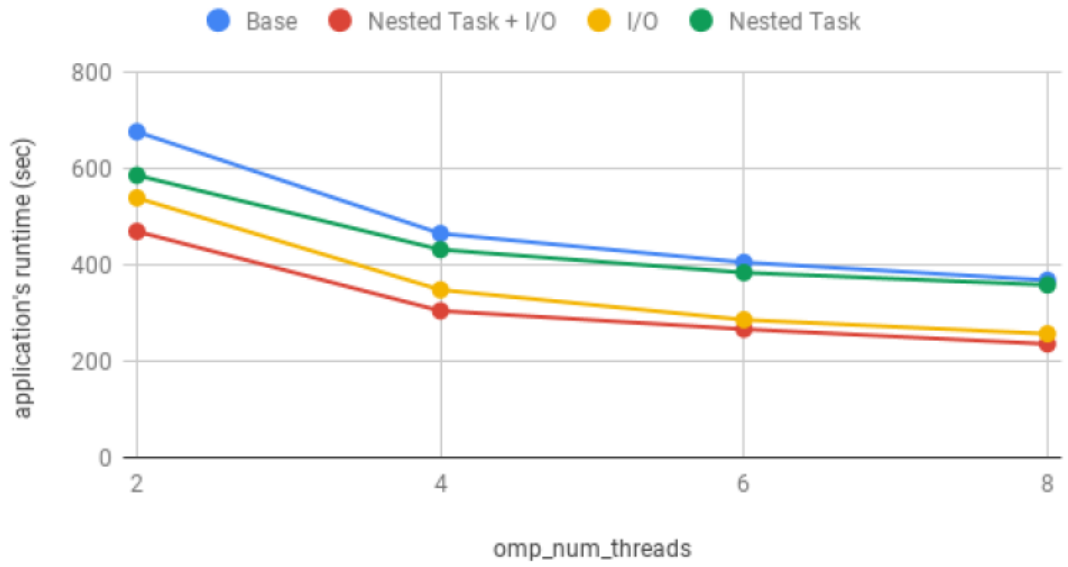
PoC implementation



- <u>Result:</u>
  - Load balance improvement: 74% -> 89% [orthogonalization routine (pink) not parallelized]
  - Speedup for the kernel: 1.18x, should be 1.09x for the real application (approx. 50% of runtime is GMRES solver)

Case Study: CalculiX

Final result



POP standard metrics for region: dgmres1mt
CalculiX

	2	4	6	8
Global Efficiency	0.8	0.85	0.88	0.89
Parallel Efficiency	0.8	0.76	0.75	0.75
Load Balance	0.81	0.77	0.76	0.76
Communication Efficiency	0.99	0.99	0.99	0.99
Serialization Efficiency	0.99	0.99	0.99	0.99
Transfer Efficiency	1.0	1.0	1.0	1.0
Computational Scaling	1.0	1.12	1.17	1.18
Instruction Scaling	1.0	1.06	1.08	1.08
IPC Scaling	1.0	1.06	1.08	1.09

POP standard metrics for region: dgmres1mt
CalculiX (PoC)

	2	4	6	8
Global Efficiency	0.92	1.01	1.05	1.06
Parallel Efficiency	0.92	0.87	0.87	0.86
Load Balance	0.98	0.93	0.92	0.9
Communication Efficiency	0.94	0.94	0.95	0.95
Serialization Efficiency	0.94	0.94	0.95	0.95
Transfer Efficiency	1.0	1.0	1.0	1.0
Computational Scaling	1.0	1.16	1.2	1.24
Instruction Scaling	1.0	1.03	1.04	1.05
IPC Scaling	1.0	1.13	1.16	1.19

- Runtime improvement by implementing nested tasks
- We also implemented buffered I/O (not covered in this talk)
- POP metrics verify performance improvements!

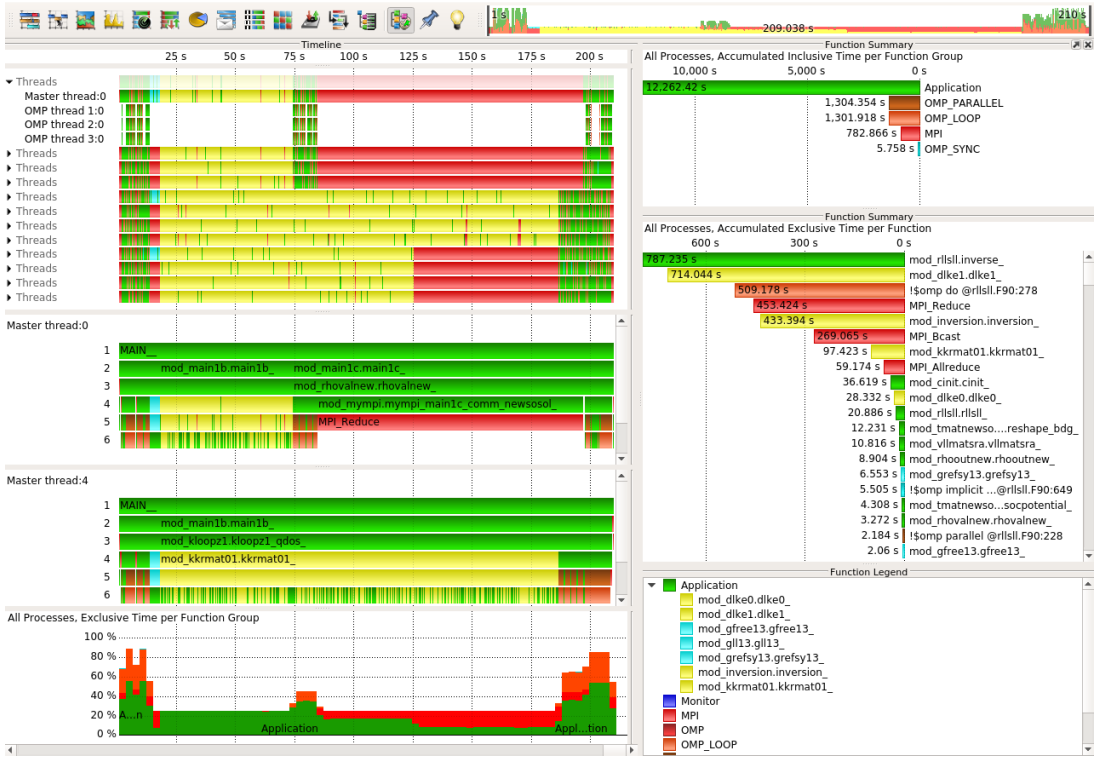
Case Studies

JuKKR – The Juelich Korringa-Kohn-Rostoker DFT code family

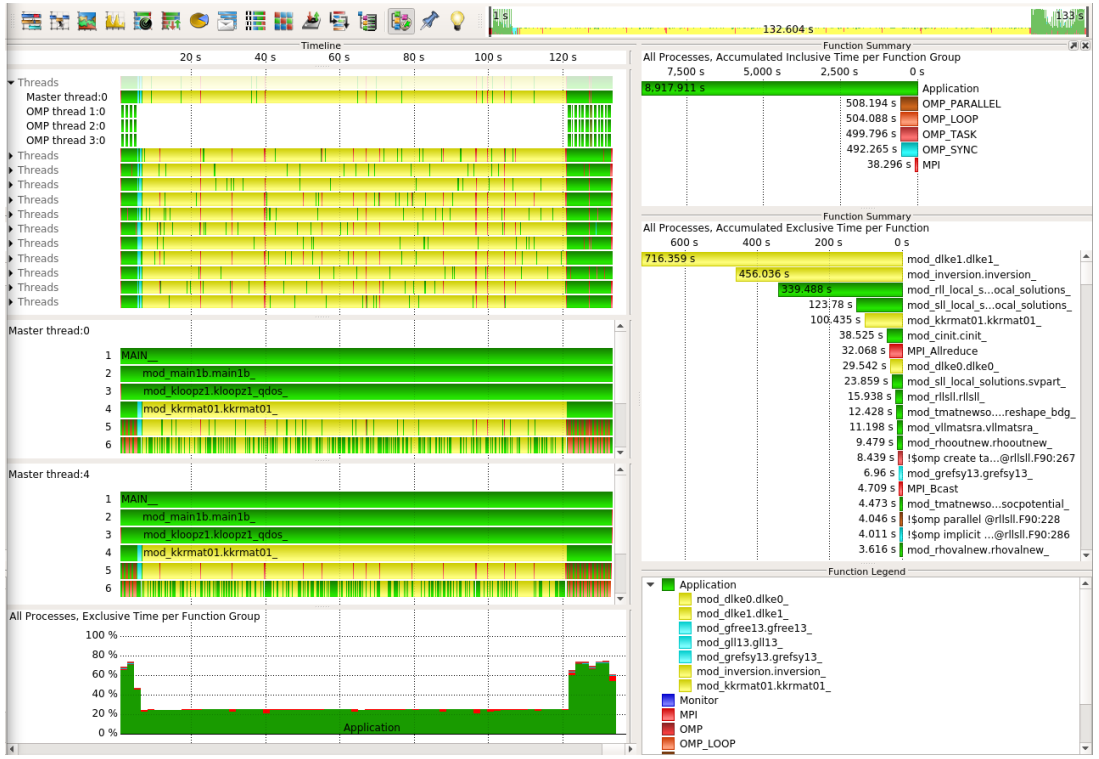
- Fortran code parallelized with hybrid MPI+OpenMP
- https://jukkr.fz-juelich.de/

Case Study: JuKKR

aiXcelerate 2019



Before aixcelerate 2019

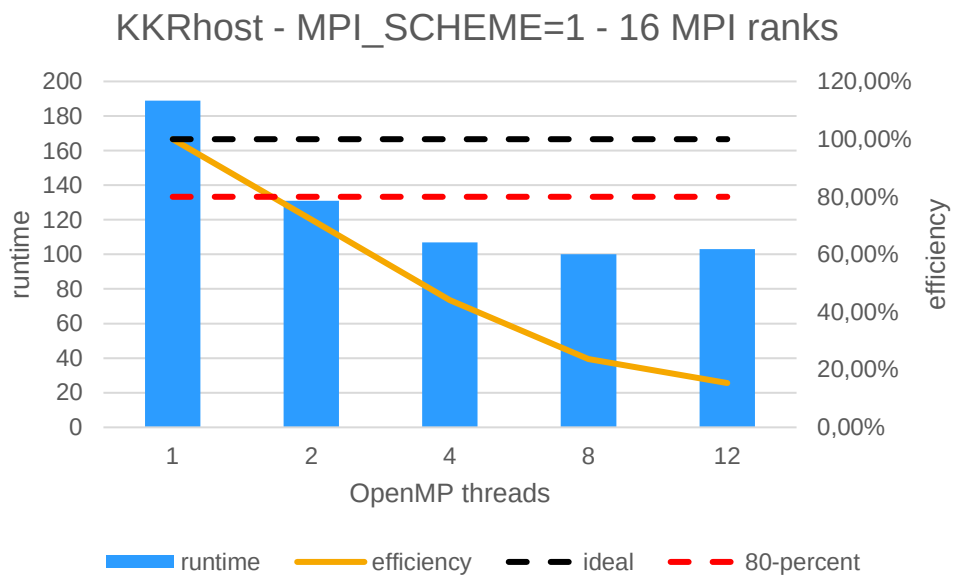


After aixcelerate 2019

- Load balance issue on MPI level identified

Case Study: JuKKR

Performance Assessment



Threads per Process	1	2	4	8	12
Global Efficiency	0.94	0.64	0.40	0.19	0.13
↪ Parallel Efficiency	0.94	0.76	0.59	0.44	0.39
↪ Process Level Efficiency	0.94	0.93	0.91	0.88	0.94
↪ Load balance	0.97	0.97	0.95	0.94	0.98
↪ MPI Communication Efficiency	0.97	0.96	0.96	0.94	0.96
↪ MPI Transfer Efficiency	1.00	1.00	1.00	1.00	1.00
↪ MPI Serialisation Efficiency	0.97	0.96	0.96	0.94	0.96
↪ Thread Level Efficiency	1.00	0.83	0.68	0.56	0.45
↪ OpenMP Region Efficiency	1.00	0.98	0.98	0.97	0.92
↪ Serial Region Efficiency	1.00	0.85	0.70	0.59	0.52
↪ Computational Scaling	1.00	0.84	0.67	0.44	0.34
↪ Instruction Scaling	1.00	0.97	0.94	0.90	0.86
↪ IPC Scaling	1.00	0.96	0.88	0.74	0.67

(add.) hybrid POP2 metrics for KKRhost, 16 MPI ranks á 12 OpenMP threads

Observations:

- (+) MPI communication is almost optimal
- (+) OpenMP regions are very balanced
- (-) lot of serial code on thread level
- (-) decreasing IPC in serial calculations

Case Study: JuKKR

K-point integration (main1b)

Algorithm 1 k-point integration

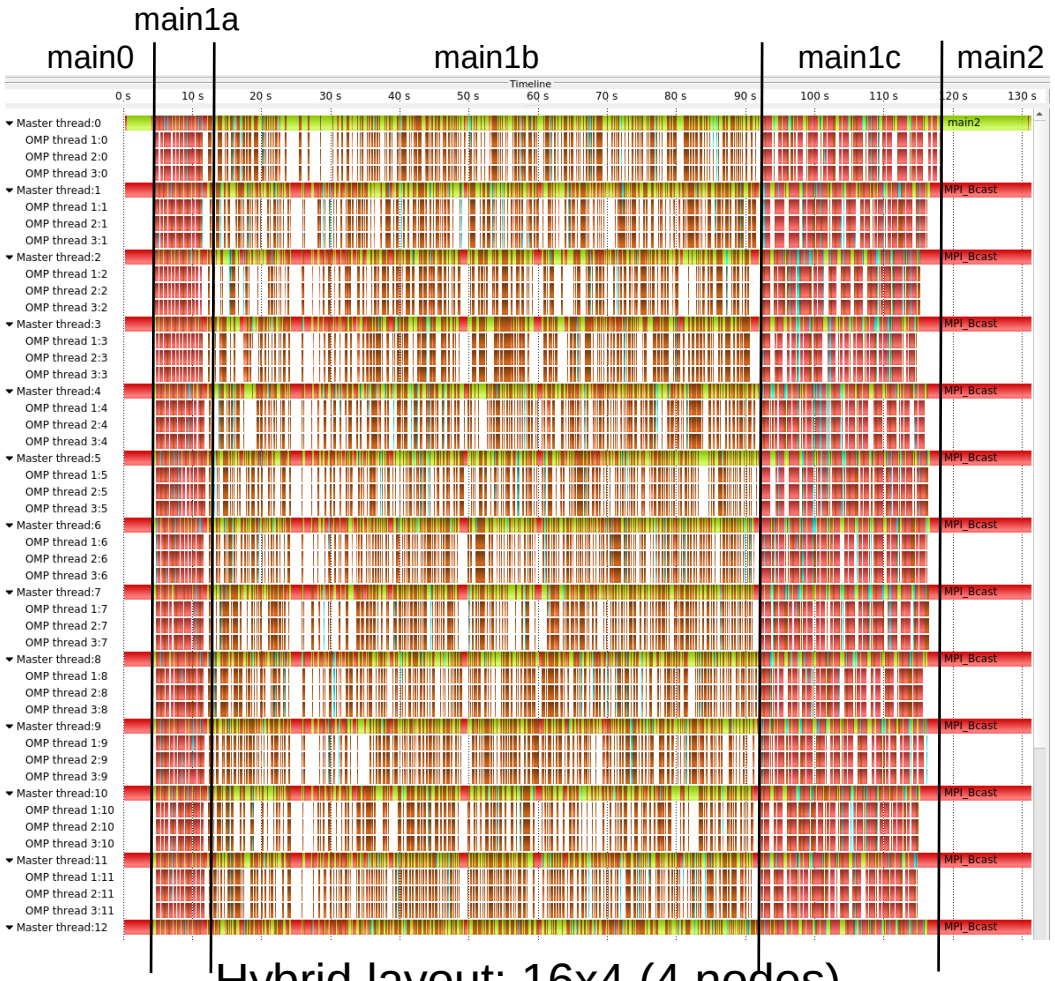
$s \leftarrow 0$
for $k = 0, \ldots, kpts$ **do**
  $c_k \leftarrow \frac{1}{V_{\mathrm{BZ}}} w^k e^{-i\mathbf{k}\cdot(\mathbf{R_i}-\mathbf{R_{i'}})}$
  $\mathbf{M}_k \leftarrow \left[(\mathbf{\Delta t}^{\mu}(E))^{-1}\ \delta_{\mu\mu'} - \mathbf{G}^{\mathrm{ref},\mu\mu'}(\mathbf{k};E)\right]$
  $\tau_k = \mathbf{M}_k^{-1}$
  $\mathbf{S} \leftarrow \mathbf{S} + c_k \cdot \tau_k$
end for

← k-points distributed over MPI ranks

← `zgetrf` (threaded Intel MKL)

Performance issues:

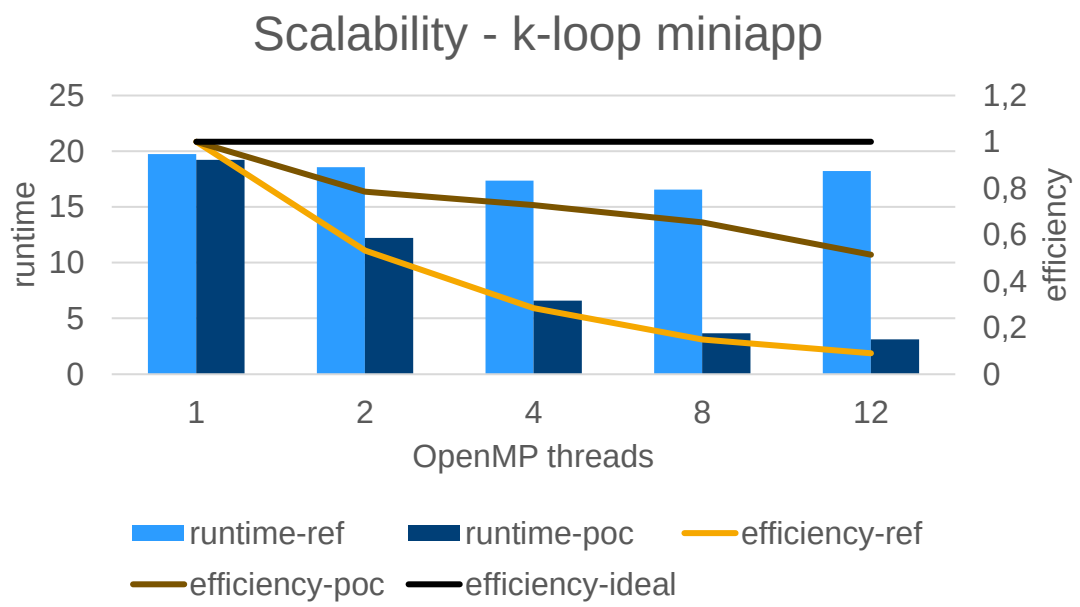
- 1.5 Mio. matrices $\boldsymbol{M}_k \in \mathbb{R}^{32\times32}$
- Lots of serial code parts
  - → Serial Region Efficiency
- Very short OpenMP parallel regions (avg 55 μs)
- Matrices fit into L1 cache
  - False Sharing on L1? → IPC scaling



Hybrid layout: 16x4 (4 nodes)

NHR4CES NHR for Computational Engineering Science
IT Center
RWTH AACHEN UNIVERSITY

PoC Results



- k-loop miniapp now scales to one NUMA domain
  - Speedup of 6x compared to original code
- Additive hybrid POP metrics verify efficiency improvements
  - Serial code parts are much smaller
  - IPC scaling is now perfect

POP additive hybrid metrics for k-loop miniapp (reference)

	1	2	4	8	12
Global Efficiency	0.99	0.53	0.28	0.14	0.09
Parallel Efficiency	0.99	0.62	0.39	0.36	0.21
Process Level Efficiency	0.99	0.98	0.98	0.98	0.98
MPI Load Balance	1.0	1.0	1.0	1.0	1.0
MPI Communication Efficiency	0.99	0.98	0.98	0.98	0.98
Thread Level Efficiency	1.0	0.64	0.4	0.27	0.23
OpenMP Region Efficiency	1.0	0.95	0.95	0.95	0.94
Serial Region Efficiency	1.0	0.69	0.45	0.32	0.29
Computational Scaling	1.0	0.85	0.72	0.57	0.42
Instruction Scaling	1.0	0.98	0.97	0.93	0.9
IPC Scaling	1.0	0.93	0.86	0.76	0.65
Frequency Scaling	1.0	0.93	0.86	0.81	0.71

POP additive hybrid metrics for k-loop miniapp (PoC)

	1	2	4	8	12
Global Efficiency	0.99	0.78	0.73	0.65	0.51
Parallel Efficiency	0.99	0.89	0.89	0.89	0.83
Process Level Efficiency	0.99	0.99	0.99	0.99	0.99
MPI Load Balance	1.0	1.0	1.0	1.0	1.0
MPI Communication Efficiency	0.99	0.99	0.99	0.99	0.99
Thread Level Efficiency	1.0	0.89	0.89	0.89	0.83
OpenMP Region Efficiency	1.0	0.99	0.99	0.99	0.99
Serial Region Efficiency	1.0	0.89	0.89	0.9	0.83
Computational Scaling	1.0	0.89	0.82	0.73	0.62
Instruction Scaling	1.0	1.0	1.0	1.0	0.99
IPC Scaling	1.0	1.0	1.02	1.02	1.01
Frequency Scaling	1.0	0.88	0.8	0.71	0.61

Summary

- POP metrics provide a standardized of assessing application performance
- Hierarchy of metrics allows to identify root causes of inefficiencies
- Different sets of metrics for different application requirements
  - Standard POP metrics for single paradigm applications (e.g. MPI **xor** OpenMP)
  - Hybrid POP metrics for multi paradigm applications (e.g. MPI+OpenMP, MPI+CUDA, …)

- POP metrics in practice (case studies)
  - CalculiX:
    - Load balance issue identified
    - ≈1.10x speedup after optimization
  - JuKKR:
    - Inefficient OpenMP parallelization of k-point integration identified
    - 6x speedup after optimization

Thank you for your attention!