PERFORMANCE ANALYSIS AND OPTIMIZATION OF CFD APPLICATIONS

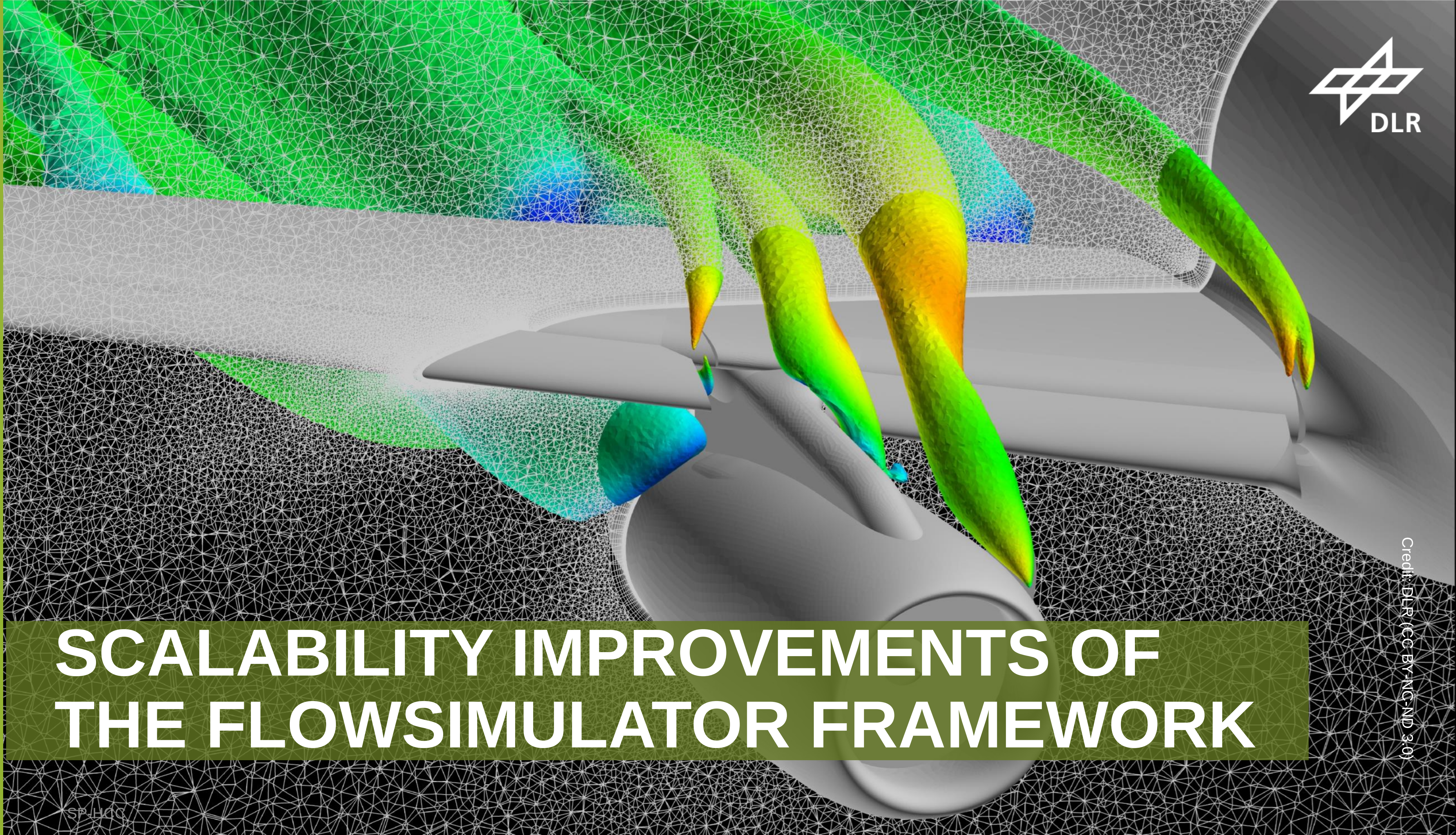
Martin Clemens, Jonathan Fenske, Daniel Molka, Hirav Patel, Ronny Tschüter, Michael Wagner

German Aerospace Center
Institute of Software Methods for Product Virtualization
(HPC Competence Center – SP-HCC)

DLR

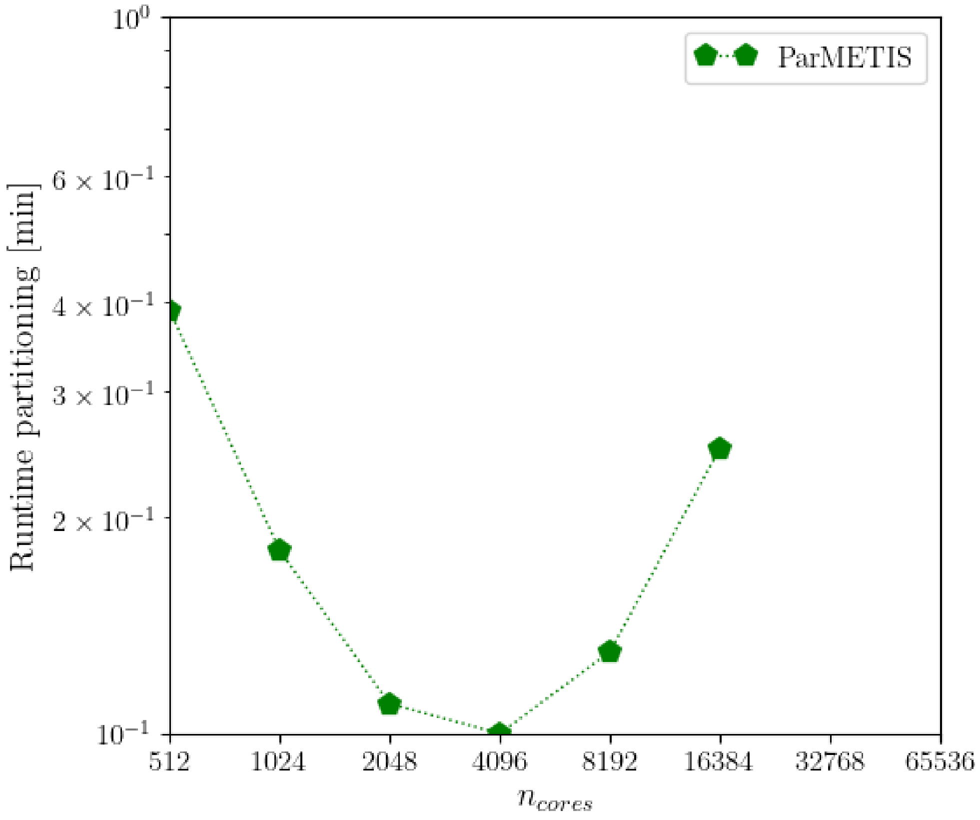
Agenda

- Scalability improvements of the FlowSimulator framework
- Investigation of HPC architectures using Amazon Web Services (AWS)
- Performance modelling of the CFD solver TRACE
- HPC operation



SCALABILITY IMPROVEMENTS OF THE FLOWSIMULATOR FRAMEWORK

Hierarchical Partitioning



- Partitioning runtime (pure MPI)
- Mesh with 723M cells, 629M nodes
- Problem: In FlowSimulator partitioning does not work anymore from a certain number of processes on.

Hierarchical Partitioning

- Partitioning data on multiple hardware hierarchy levels
- Can be done in two directions:
  - First partition data among compute nodes, then on lower hierarchy level within compute nodes, …
    - Benefit: reduced communication time, domain decomposition for less processes at once
    - Drawback: higher imbalance factor
  - First partition data among all processes, then redistribute partitions so that communication is minimized in higher hierarchy level, …
    - Benefit: reduced communication time, better imbalance factor
    - Drawback: potentially computation of large number of partitions at the same time

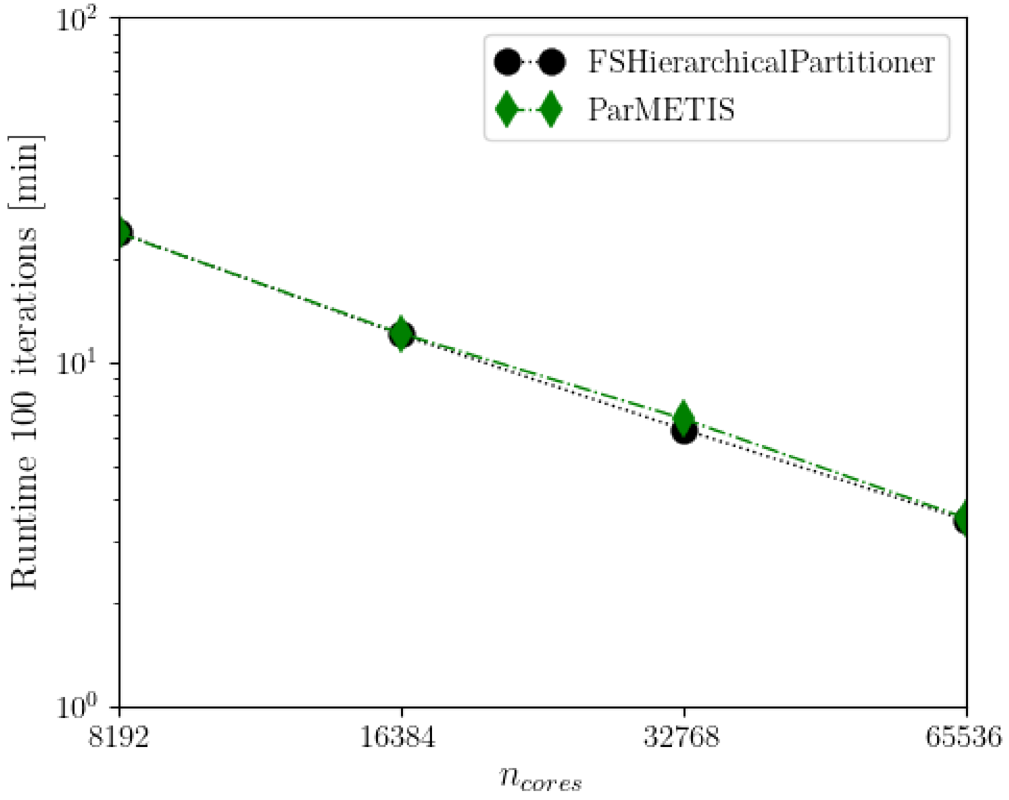
Hierarchical Partitioning Plugin for FlowSimulator

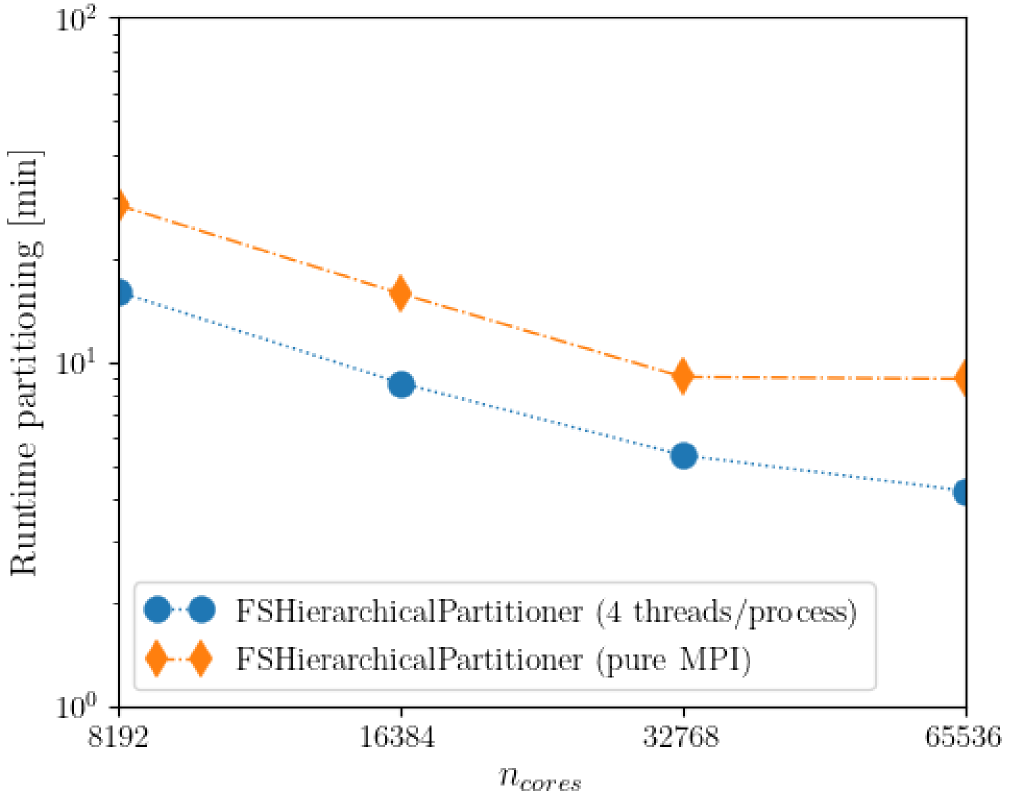
- Top down approach implemented as 2-level hierarchical partitioners
- To compute partitioning on each hierarchy level, an external graph partitioner is called (ParMETIS or Zoltan)
- To review the influence of the partitioning, CODA strong scaling benchmarks with large mesh (1.23B cells, 973M nodes) were done by taking the timings
- The partitioning were computed with ParMETIS because Zoltan does not work well for large meshes

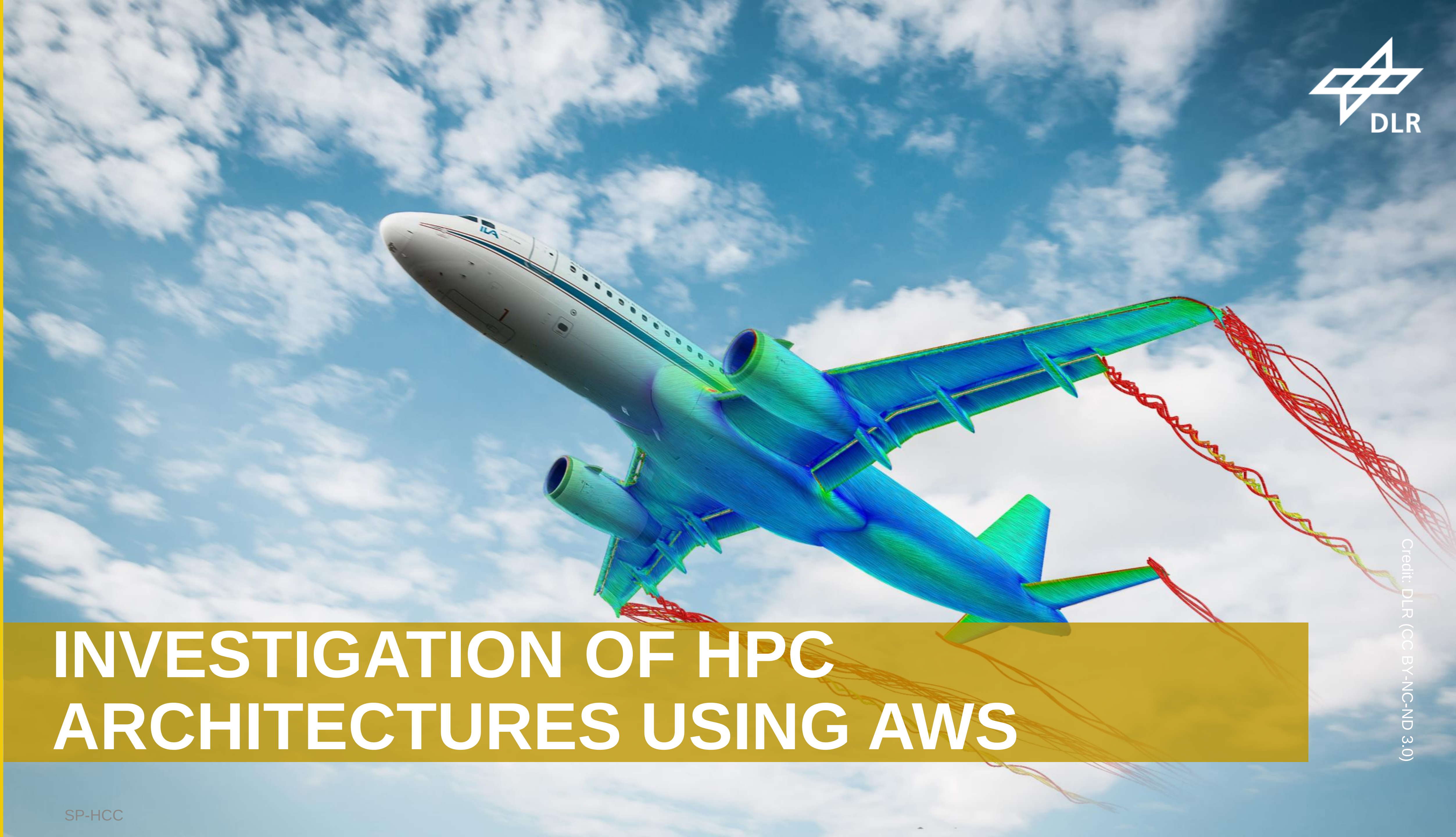
Results

Partitioning runtimes

- CODA runtimes (4 cores/process)







INVESTIGATION OF HPC ARCHITECTURES USING AWS

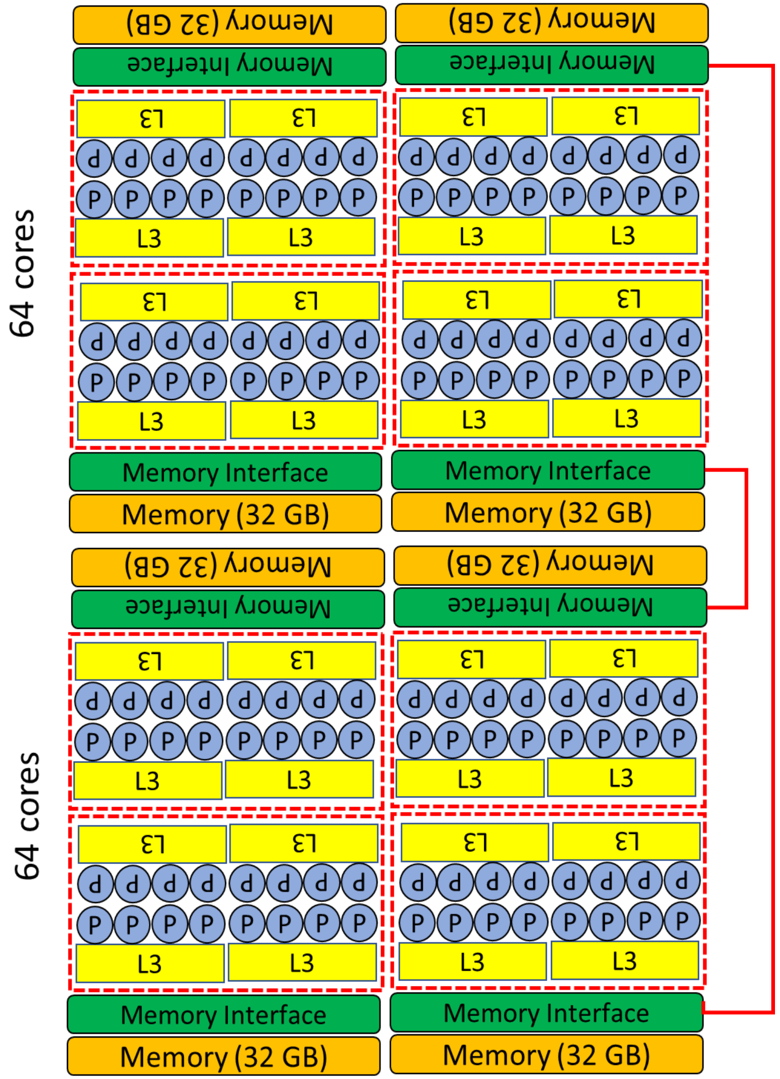
Architecture of Tested Nodes

DLR CARO	AMD EPYC Rome (Zen2) [100 Gbps Infiniband]
hpc7a	AMD EPYC Genoa (Zen4) [300 Gbps EFA (Elastic Fabric Adapter)]
hpc6a	AMD EPYC Milan (Zen3) [100 Gbps EFA]
c7gn, hpc7g	ARM Graviton3 instances, same specs and perf. [200 Gbps EFA]
c6gn	ARM Graviton2 [100 Gbps EFA]

Reference: https://aws.amazon.com/ec2/instance-types/

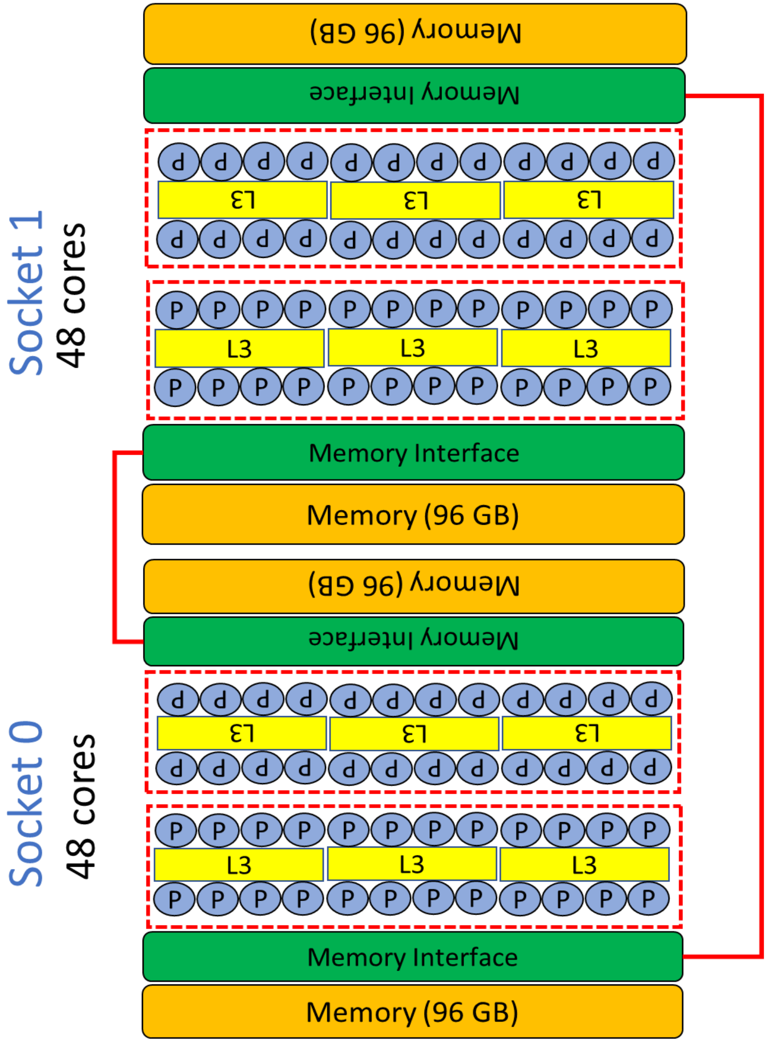
Zen Architectures

DLR CARO (AMD EPYC 7702)



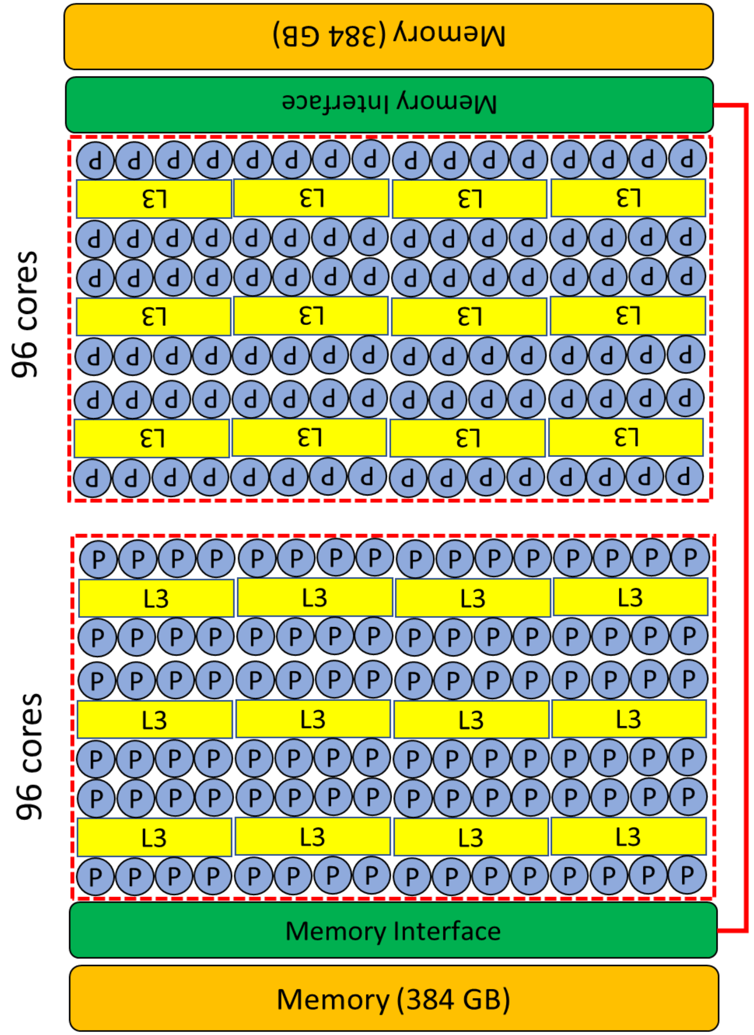
*L1 (32kB) and L2 (512 kB) *per core, L3 (16 MB)*

hpc6a.48xlarge (AMD EPYC 7R13)



*L1 (32kB) and L2 (512 kB) *per core, L3 (16 MB)*

hpc7a.96xlarge (AMD EPYC 9R14)



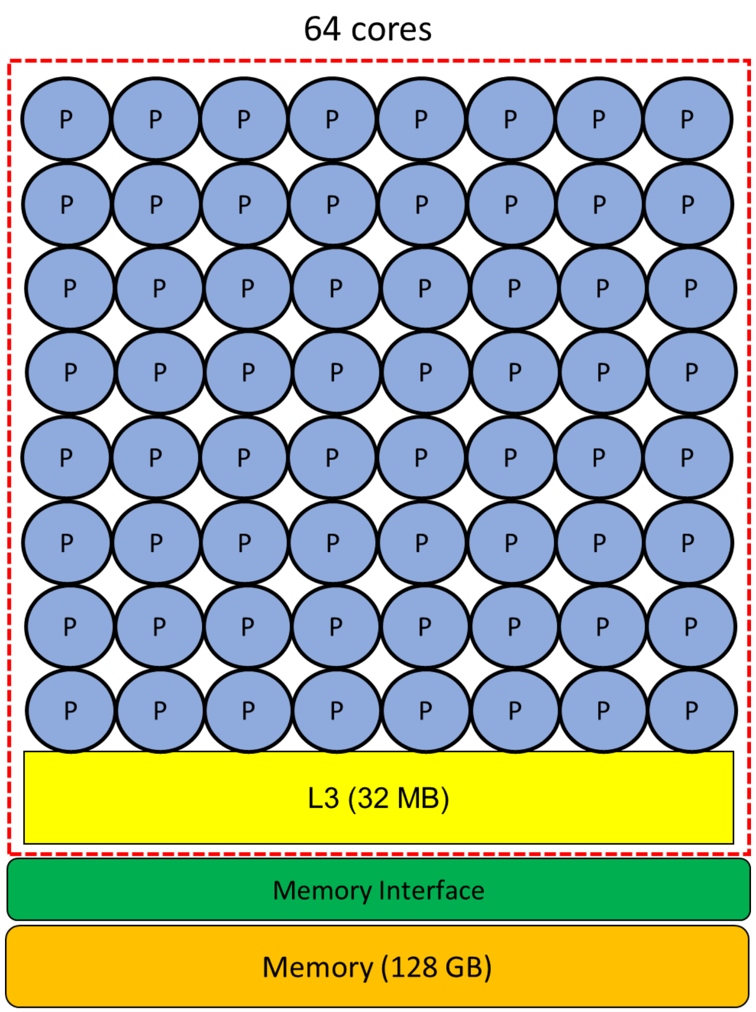
*L1 (32kB) and L2 (1 MB) *per core, L3 (32 MB)*

DLR

AWS Graviton Processor

- ARM 64bit architecture
- Graviton 3 is the successor of Graviton 2
- Graviton 3 (ARMv-8.4 ISA) vs Graviton 2 (ARMv-8.2 ISA)
- UMA – Uniform Memory Access system
- Single socket system with 64 cores
- 1 Node = 1 Socket
- Graviton3 – 8 memory channels
- L1 Cache - 64 kB (per core)
- L2 Cache - 1 MB (per core)
- L3 Cache - 32 MB (shared between 64 cores)
- Memory - 128 GB for Socket
- Graviton 3 (2.6 GHz) vs Graviton 2 (2.5Ghz)
- Graviton 3 has more core width than Graviton 2 - (higher IPC)
- Graviton 3 has DDR5 and faster memory channels than Graviton 2

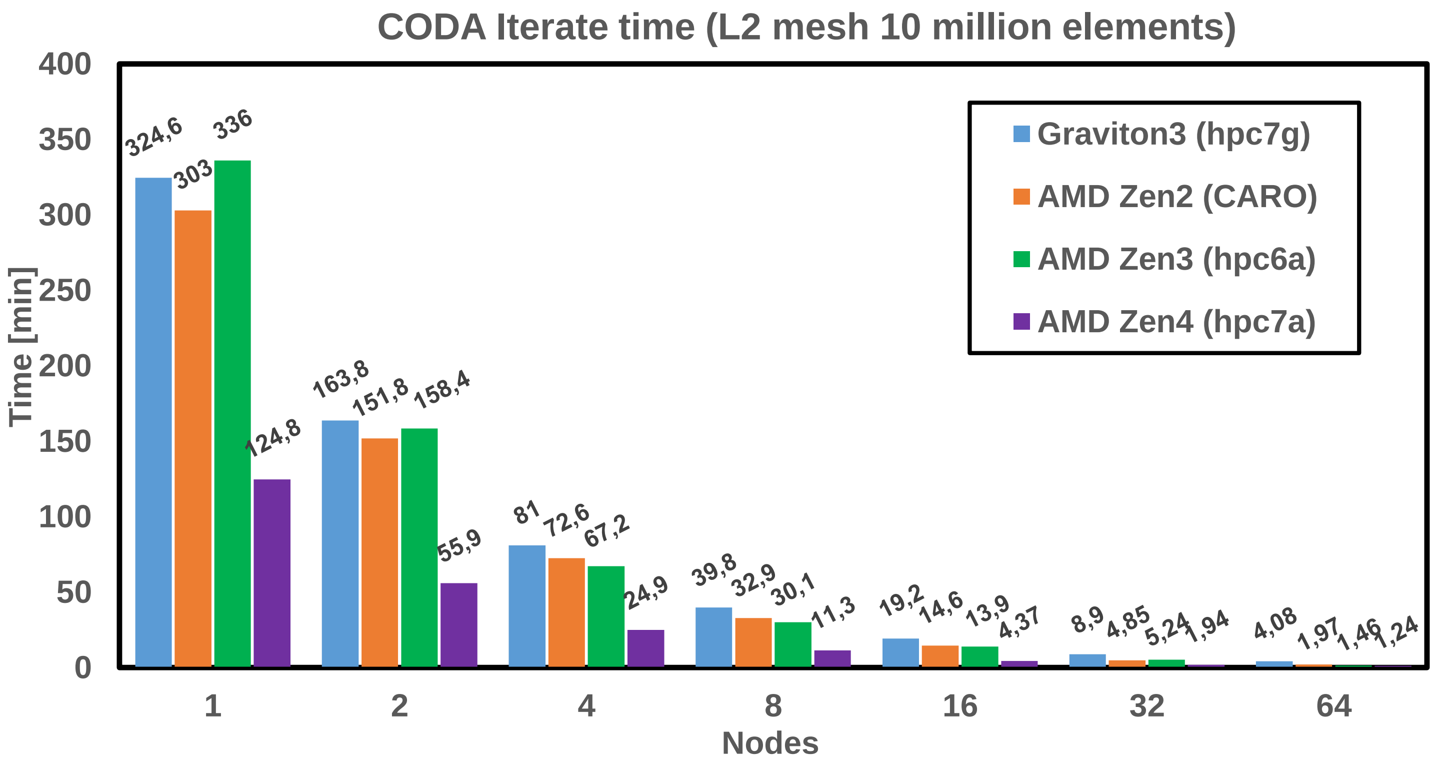
Graviton 3 and Graviton 2

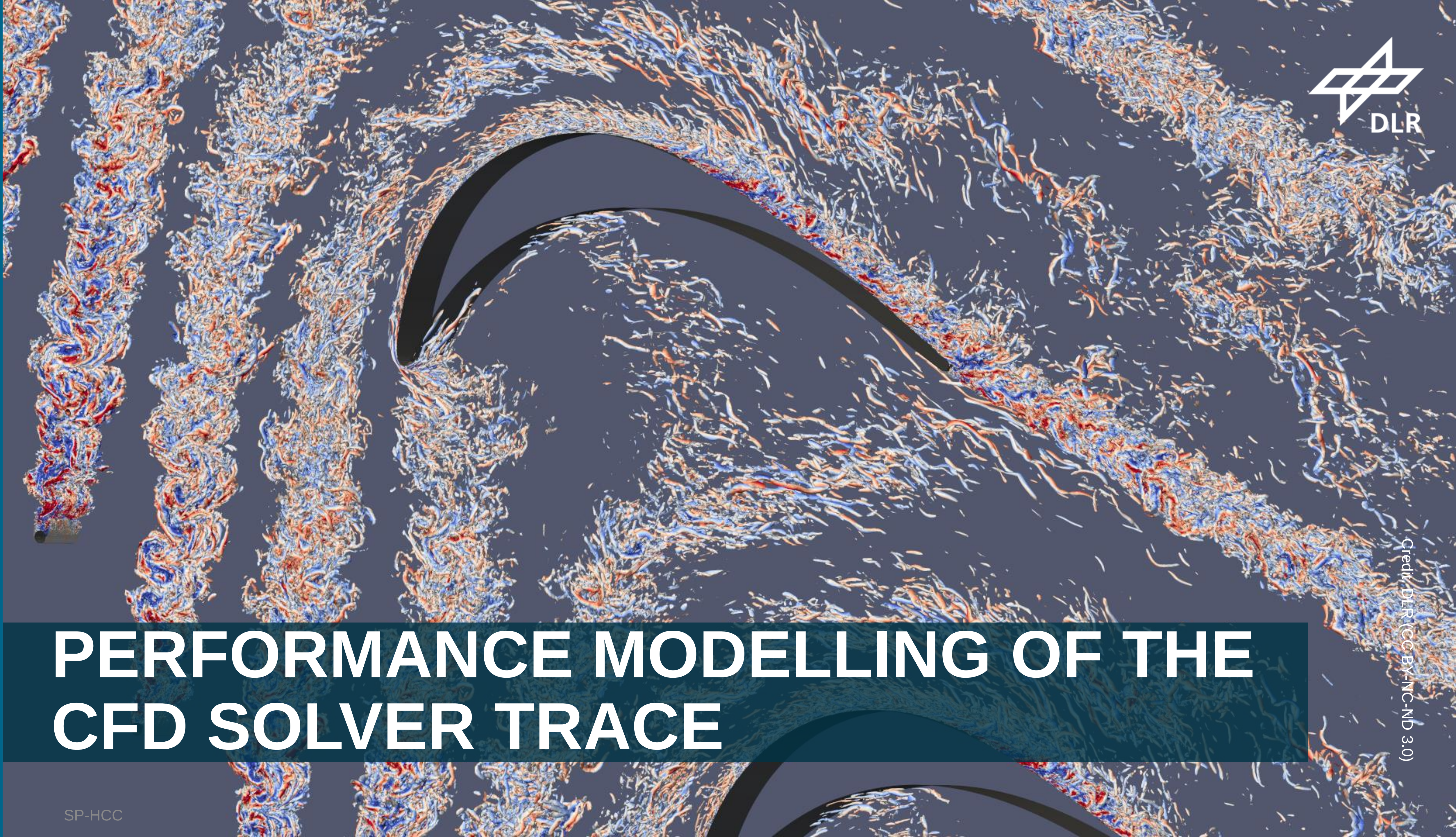


*L1 (64 kB) and L2 (1 MB) *per core*

CODA Iterate time comparison

• Graviton3 (hpc7g)	- 16 MPI tasks x 4 OMP threads per task (64 cores/node)
• CARO (Zen2)	- 32 MPI tasks x 4 OMP threads per task (128 cores/node)
• hpc6a (Zen3)	- 12 MPI tasks x 8 OMP threads per task (96 cores/node)
• hpc7a (Zen4)	- 24 MPI tasks x 8 OMP threads per task (192 cores/node)





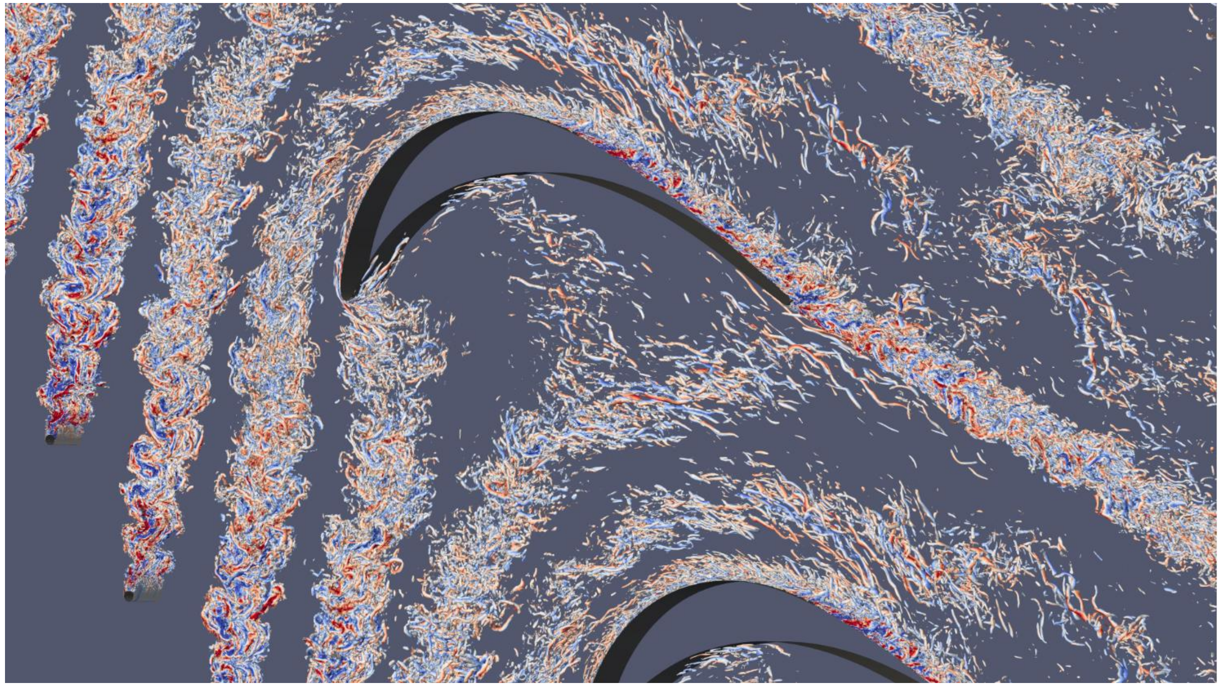
PERFORMANCE MODELLING OF THE CFD SOLVER TRACE

TRACE

Turbomachinery Research Aerodynamic Computational Environment

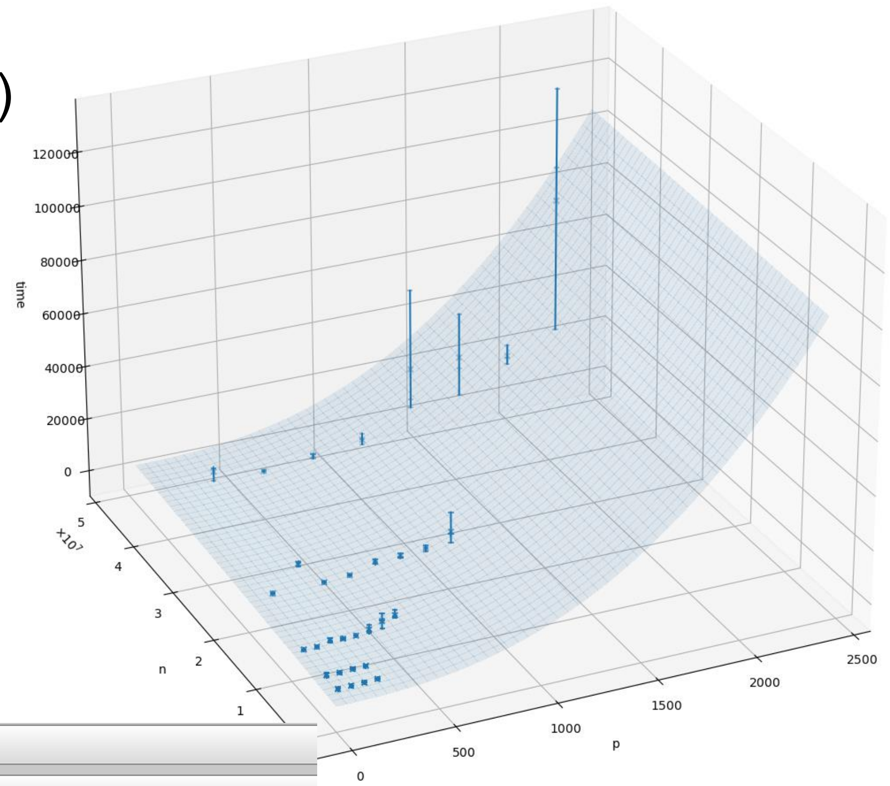
- DLR‘s standard CFD solver for turbomachinery flows
- Also used in industrial design processes by MTU Aero Engines AG and Siemens Energy AG

- Steady and unsteady RANS solver on structured and unstructured grids
- Hybrid parallelization with MPI and OpenMP



Extra-P

- Fix model parameters (#procs, #cells, polynomial degree, …)
- Run repeated measurements
  → experiment directory with Cube profiles
- Automatically generate performance model for every node in the call tree using Extra-P
  - Metrics include time, #calls, MPI bytes sent, …



Model: Default Model

Metric: time

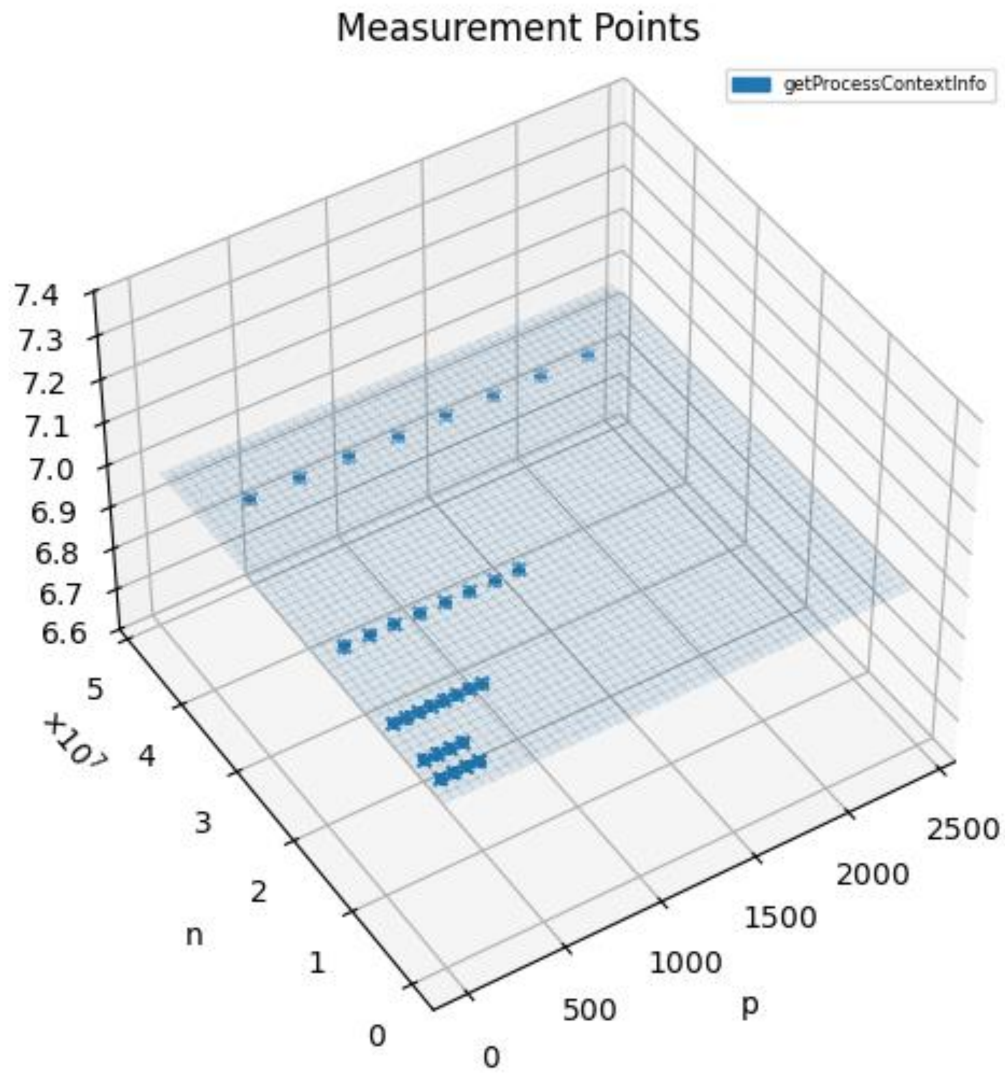
Se	Callpath	An	Value	RSS	Adj. R^2	SMAPE	RE
	TRACE		$0.351 + 5.395\times10^{-06} * p^{3/2} * \log_2(p)$	4.428	0.924	36.819	0.532
	main		$0.160 - 2.514\times10^{-04} * n^{2/3} + 3.427\times10^{-05} * \log_2(p) * n^{2/3}$	78.718	0.893	66.985	1.080
	traceInitEnvironment		$155.225 - 0.013 * n^{1/2} * \log_2(n) + 5.827\times10^{-06} * p^{3/4} * \log_2(p)^2 *$ …	1.050…	0.954	56.301	0.694
	allocateSolverStructures		$2752.430 - 0.208 * n^{2/3} + 2.986\times10^{-03} * p^{3/4} * n^{2/3}$	1.002…	0.941	74.168	1.147
	commProcessIsRootGroupMaster		$2.924\times10^{-03} + 3.502\times10^{-06} * p * \log_2(p) + 2.944\times10^{-10} * n^{1/2} *$ lo…	0.019	0.367	75.980	1.275
	getProcessContextInfo		$5.084\times10^{-05} + 9.093\times10^{-13} * n$	2.017…	0.353	14.844	0.147
	passLogMessage		0.088	0.612	1	109.9…	0.856
	MPI_Comm_rank		$-0.132 - 0.024 * p^{1/2} + 5.974\times10^{-04} * p^{1/2} * n^{1/4}$	0.644	0.808	116.3…	10.161
	MPI_Allreduce		$-434.672 + 3.593\times10^{-04} * p^{5/2} + 1.491\times10^{-05} * n^{3/4} * \log_2(n)^2$	9.499…	0.891	41.482	0.424

Testcase

- 6 grids with number of cells ranging from 2.5e6 to 8.1e7
  → strong scaling in p-direction

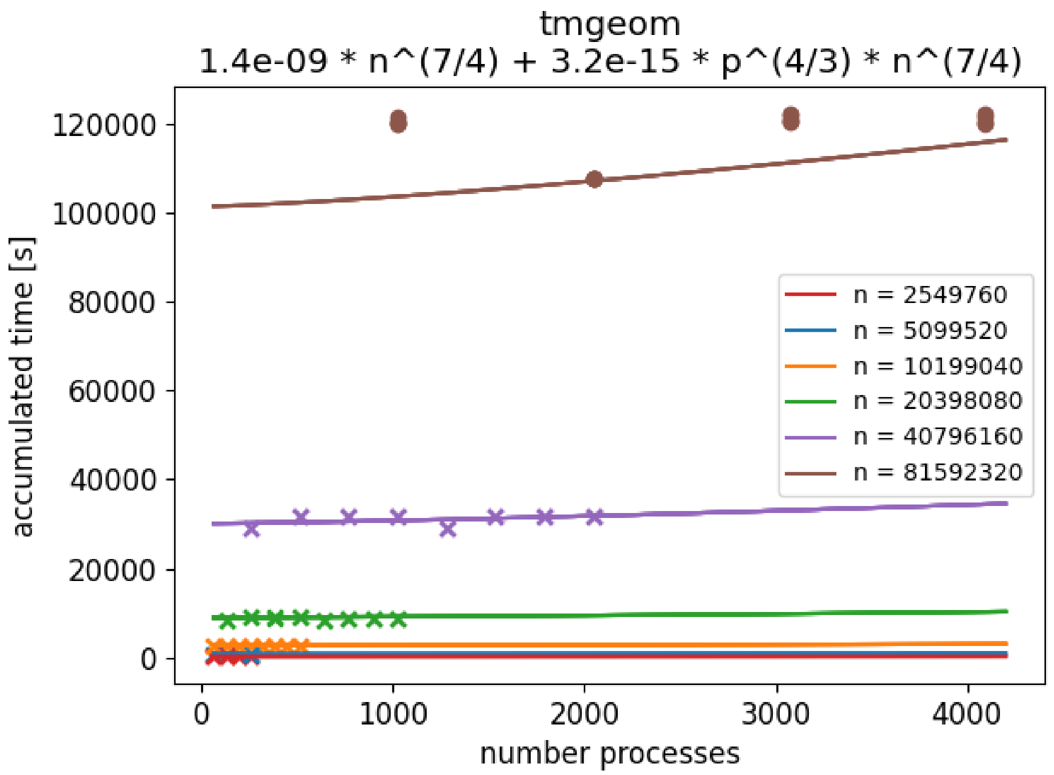
# cells (n)	# processes (p) Low	High	# partitions
2.5e6	64	256	4
5.1e6	64	256	4
10.2e6	64	512	8
20.3e6	128	1024	8
40.8e6	256	2048	8
81.6e6	1024	4096	4

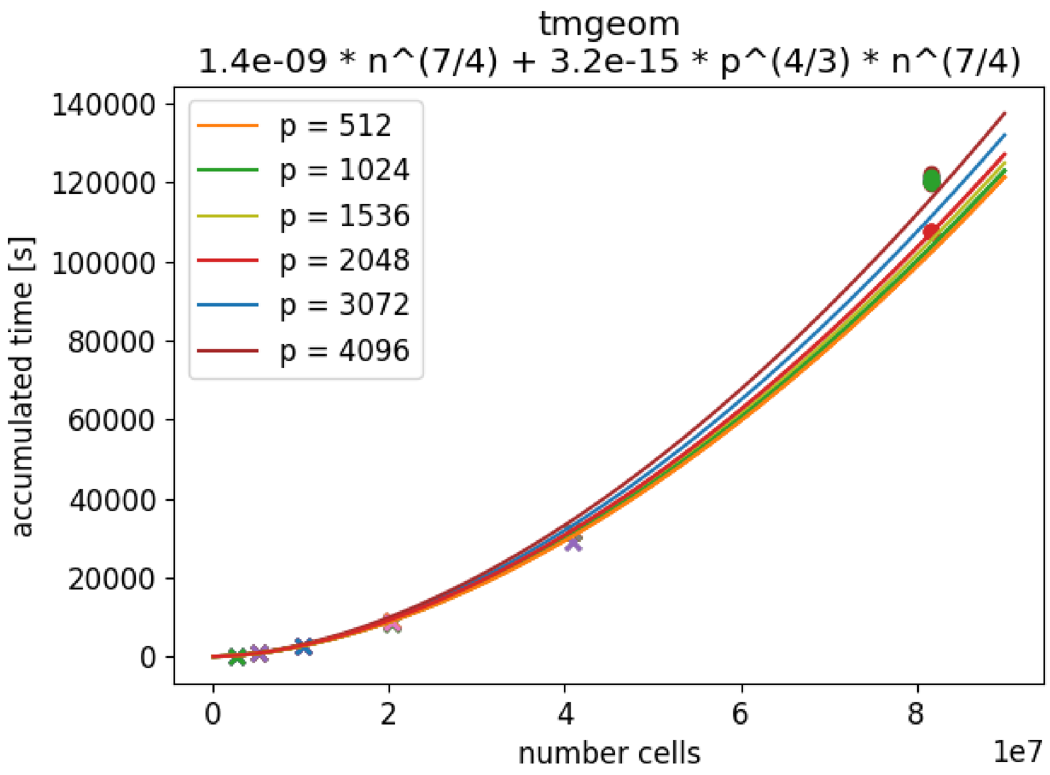
- Last line used as validation data
- Variables:
  - n: number of cells
  - p: number of processes
- Investigated routines part of not optimized setup



Results: Computation

- For every cell (n) lookup distance to wall in kd-tree (ideally log(n))
- Satisfying model

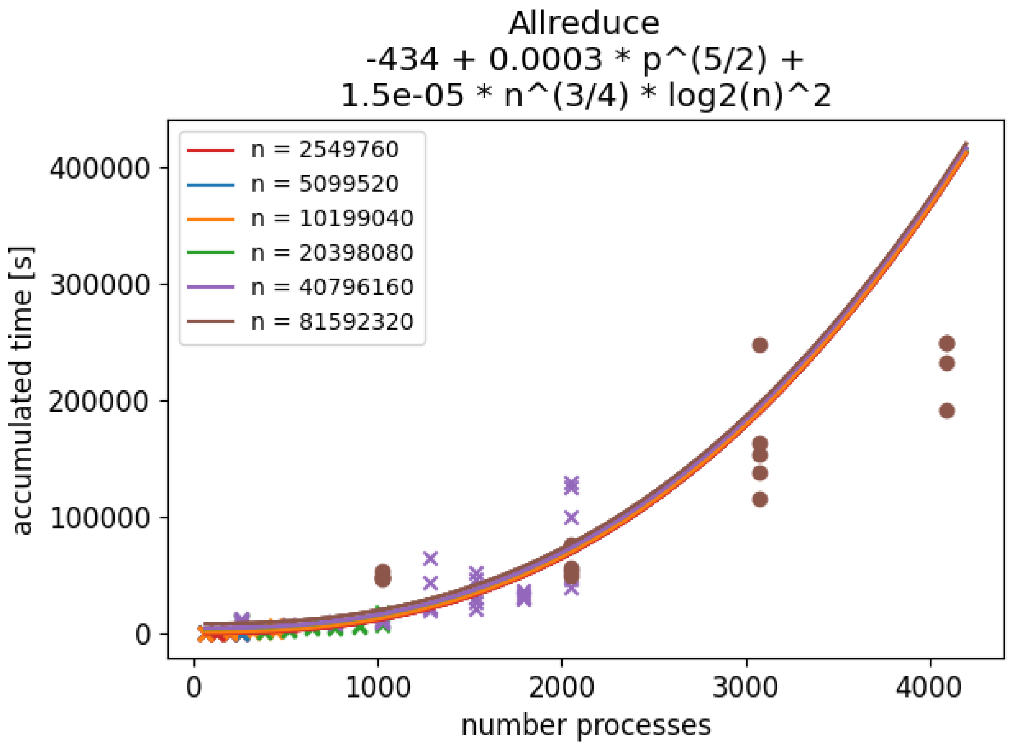


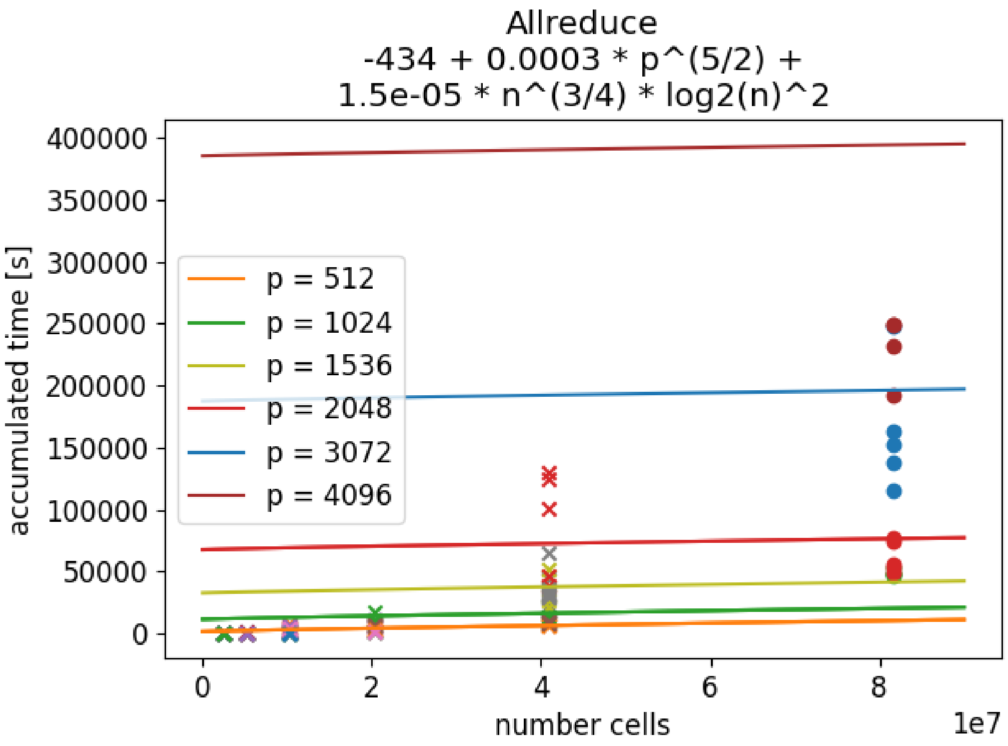


Results: Communication

- Deeper look in p^(5/2) runtime necessary
- Model satisfying only in trend, not in quantitative values

- Problems:
  - Spread of measurement points
  - Setup of testcase





HPC OPERATION

HPC Systems operated by DLR-SP

HPC cluster CARA, Dresden

Credit: DLR (CC BY-NC-ND 3.0)

- 2,168 CPU-nodes with 2 AMD EPYC 7601 (2x 32 cores)
- 664 CPU-nodes with 2 AMD EPYC 7702 (2x 64 cores)
- 10 GPU-nodes with 4 Nvidia A100 and 2 AMD EPYC 7702
- 17 PB Luste file system (0,5 PB SSD / 16,5 PB HDD)
- Operational since 2020/2023 → Replacement 2025

HPC cluster CARO, Göttingen

Credit: DLR (CC BY-NC-ND 3.0)

- 1,364 CPU-nodes with 2 AMD EPYC 7702 CPUs
- 8.4 PB Lustre file system (HDD with SSD cache)
- Operational since 2022 → Replacement 2027/28

HPC Operation

- Continuous application monitoring
  - Identify applications with inefficient resource usage
  - Identify candidates for detailed performance analysis
  - Verify performance optimizations
  - Track performance degradation

- Input for next HPC procurements
  - Information about (performance) characteristics of our application mix
  - Investigation of HPC architectures
  - Performance modelling to estimate performance on future systems

Acknowledgement

The authors gratefully acknowledge the scientific support and HPC resources provided by the German Aerospace Center (DLR). The HPC system CARA is partially funded by “Saxon State Ministry for Economic Affairs, Labour and Transport“ and „Federal Ministry for Economic Affairs and Climate Action”.