OpenFOAM Optimization Strategies for Next-Gen ARM CPUs

- Smart Platforms Lab, Huawei
  - Anthony Berkow
  - Igal Tsarfis
  - Mark Wasserman
  - Alon Zameret
  - Yann Delorme
  - Zhaohui Ding
  - Weicheng Pei

- The 20th OpenFOAM Workshop, July 2025

HUAWEI

OpenFOAM - Background

❑OpenFOAM in Huawei

- Huawei provides a full HPC stack
- OpenFOAM is extensively used by our customers

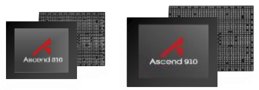
❑OpenFOAM's growth relies on effectively leveraging massively parallel HPC architecture

❑OpenFOAM's performance scaling is currently limited

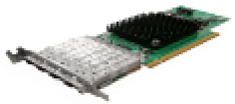
- <u>Memory bandwidth:</u> Irregular access patterns and low computational intensity → intra-node scalability
- <u>Communication overhead:</u> MPI communication dominates at large scale → inter-node scalability
- <u>Computation efficiency:</u> Complex data access patterns and lack of support for mixed-precision computation

CloudEngine (Switch)

OceanStor (Storage)



Ascend (AI)



IN200
(25/50/100GbE NIC)

Kunpeng 920 (CPU)

Investigated Optimization Goals

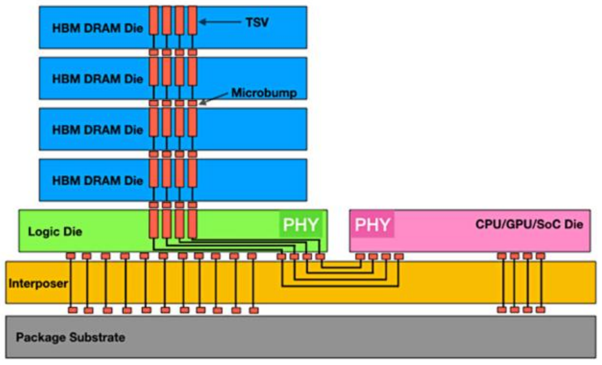
- **Use modern architecture capabilities to reduce bottlenecks**
  - Introduction of DDR5 and on-package memory → memory bound
  - Many-core architecture → communication bound
  - Vector/matrix SIMD units → computation bound

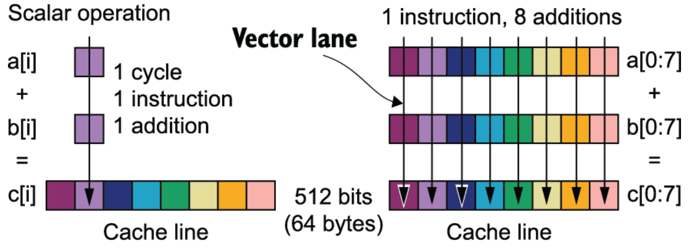
- **Investigated directions**
  - <u>Mixed-Precision (MxP) Solvers:</u> Reduce memory bandwidth pressure and increase computational throughput
  - <u>Smart Memory Allocator (SMA):</u> Efficient usage of all memory tiers (DDR/on-package memory) to accelerate memory access
  - <u>Vectorization:</u> Usage of SIMD to accelerate computation
  - <u>Asynchronous Solvers:</u> Reduce inter-process communication frequency

- **This presentation will focus on the first two directions**

HPL-MxP Benchmark Top 5 List, May 2024

Rank (HPL-MxP)	Computer	Cores	Speedup of HPL-MxP over HPL
1	Aurora, intel/HPE Cray EX	8,159,232	10
2	Frontier, HPE Cray EX235a	8,699,904	8.3
3	LUMI, HPE Cray EX235a	2,752,704	6.2
4	Fugaku, Fujitsu A64FX	7,630,848	4.5
5	Leonardo, Bull Sequana XH2000	1,824,768	7.5



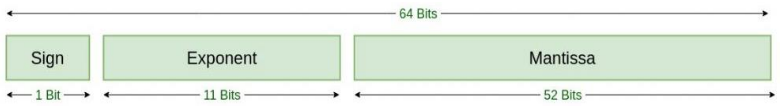


HUAWEI

MxP: Overview

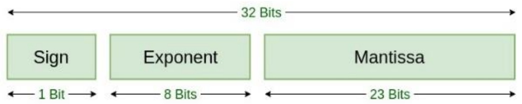
❑Precision level is the number of significant digits a value accurately represents

- Low precision enables lower memory usage, faster computation and better hardware efficiency
- High precision ensures greater accuracy but incurs higher memory and computation overhead



1.23456789012345

Double Precision
IEEE 754 Floating-Point Standard

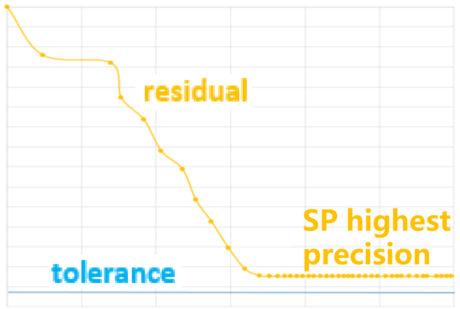


1.234567

Single Precision
IEEE 754 Floating-Point Standard

❑MxP combines reduced precision and high precision

- Balance performance, memory usage and accuracy



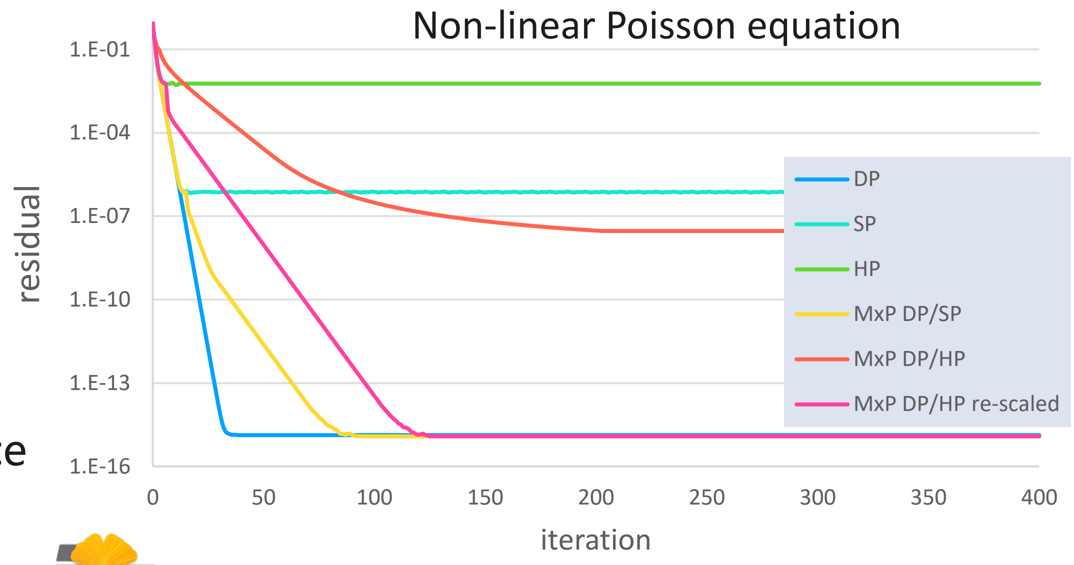
❑OpenFOAM supports the following precision levels

- Double-Precision (DP): Defines scalars as *double*
- Single-Precision (SP): Defines scalars as *float*
- Mixed-Precision (SPDP): Defines scalars as *float* but uses *double* for solvers
  → Compute in DP and truncate result to SP

❑Our approach is opposite: Store in DP and solve in SP

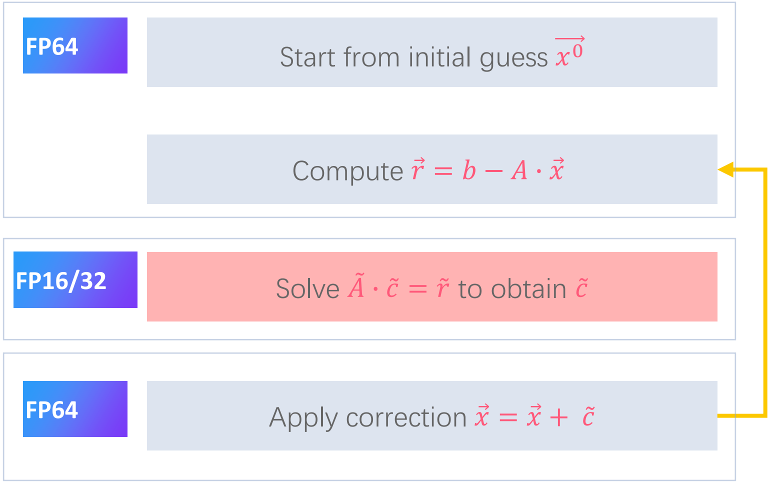
MxP: Corrective Solver Algorithm

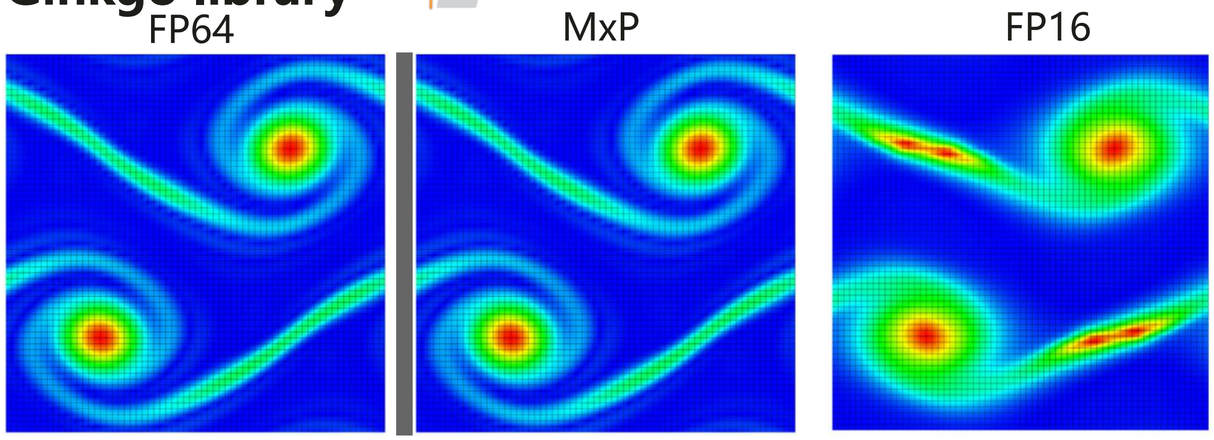
- **Linear solvers target $\boldsymbol{A \cdot x = b}$ system of equations where**
  - $r = b - A \cdot \hat{x}$ is the residual for a solution $\hat{x}$
  - Considered converged when $Norm(r) < tolerance$



- **Developed MXPC solver that adds correction term $\boldsymbol{c}$ such that $\boldsymbol{A \cdot (x + c) = b}$**
  - We receive $A \cdot c = b - A \cdot \hat{x} = r$
  - Obtains double-precision accuracy upon convergence

- **Preliminary PoC demonstrated using Ginkgo library**





2D Double Shear Layer

HUAWEI

MxP: Implementation in OpenFOAM

❑Introduced mixed-precision (*MXPC*) solver

- Creates low-precision inner solver instance to enable MxP execution
- Templatized numerus OpenFOAM classes to include support for solvers to have DP and SP instances
- Supports all segregated solvers, preconditioners and smoothers

```
template<class Type, class SType>
class TlduMatrix
```

```
typedef TGAMGSolver<floatScalar, floatScalar> GAMGSolverSP;
```

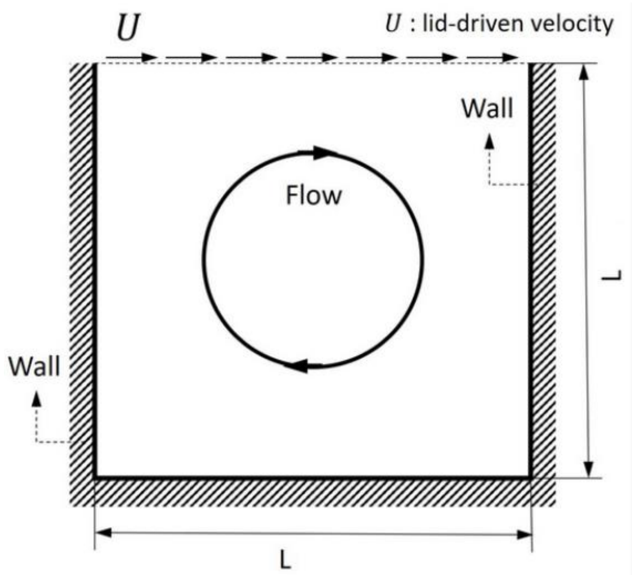
❑Implemented as a standalone module

❑Configured and used directly through the *fvSolution* dictionary

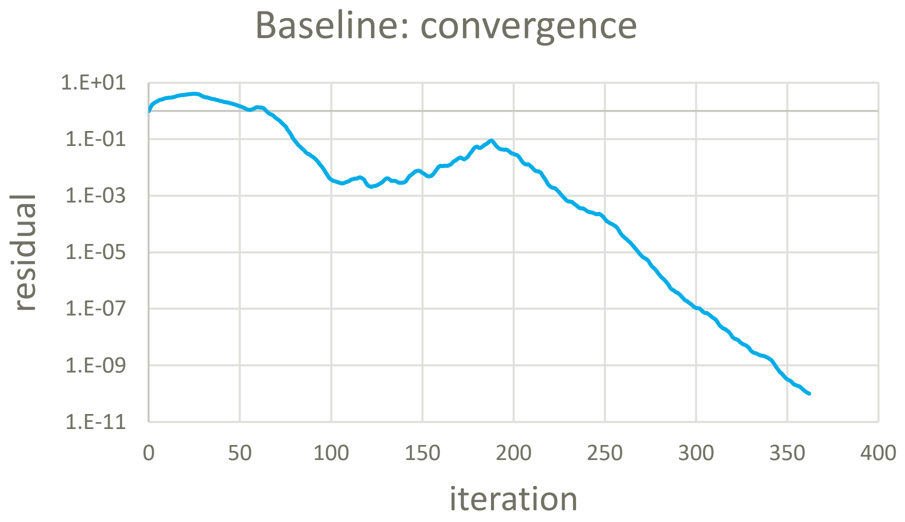
fields	scalarField, some of FieldFunctions
finiteVolume	coupledFvPatchField, processorFvPatchField, faceAreaPairGAMGAgglomeration
OpenFOAM/matrices	LUscalarMatrix, procLduInterface, procLduMatrix, LduInterfaceField, duInterfaceField, processorLduInterfaceField, lduMatrix, solverPerformance, scalarMatrices
OpenFOAM/matrices: **Preconditioners**	DIC, GAMG, none
OpenFOAM/matrices: **Smoothers**	GaussSeidel
OpenFOAM/matrices: **Solvers**	GAMG, PBiCGStab, PCG, diagonal, smoothsolver, MXPC
OpenFOAM/matrices: GAMG agglomerations	GAMGAgglomerateLduAddressing, GAMGAgglomeration, GAMGProcAgglomeration algebraicPair, dummy, pair, faceAreaPair, noneGAMGProcAgglomeration
OpenFOAM/matrices: GAMG interfaces	GAMGInterfaceField, processorGAMGInterfaceField, GAMGInterface
OpenFOAM/memory:	FieldFieldPrecisionAdaptor
OpenFOAM/primitives	Tensor

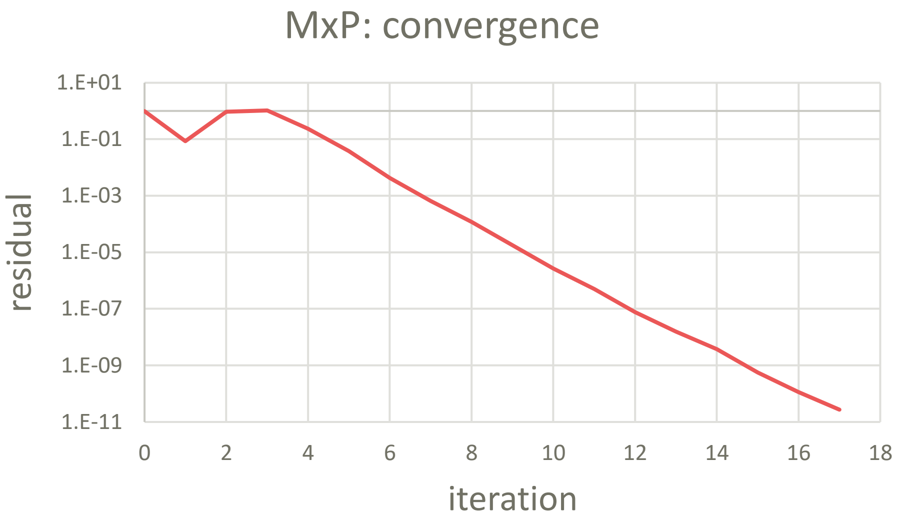
```
p
{
    solver      MXPC;
    tolerance   1e-7;
    relTol      0.01;

    SPsolver
    {
        solver      GAMG;
        smoother    GaussSeidel;
        tolerance   0;
        relTol      0.7;
        maxIter     50;
    }
}
```

MxP: 3D Cavity - MXPC validation (1 of 2)



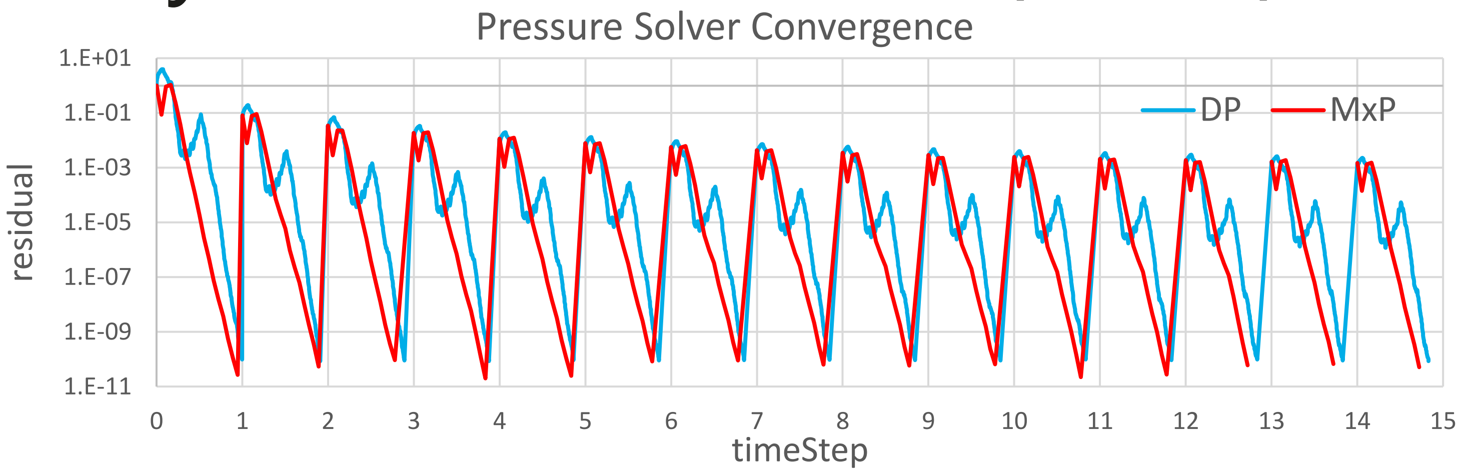
- **exaFOAM cavity3D benchmark (used for initial MXPC validation)**
  - Mesh size: 1M cells
- **Setup:**
  - Baseline: Pressure: DP PCG
  - MxP: Pressure: MXPC + SP PCG
- **Results:**
  - MXPC retains the DP precision, no error accumulates over time steps.
  - MXPC retains physical solution values (pressure and velocity).

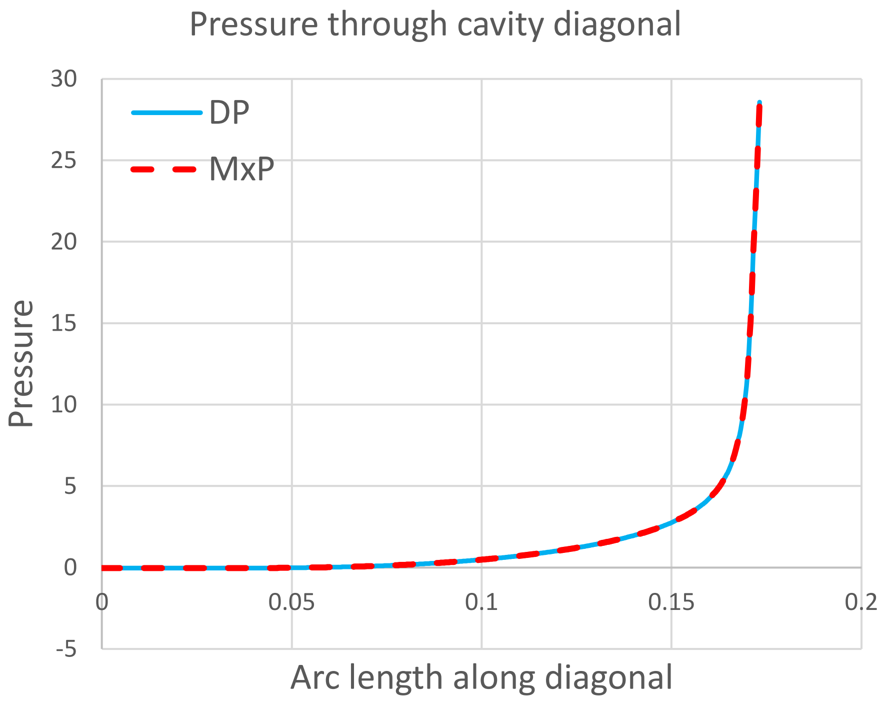


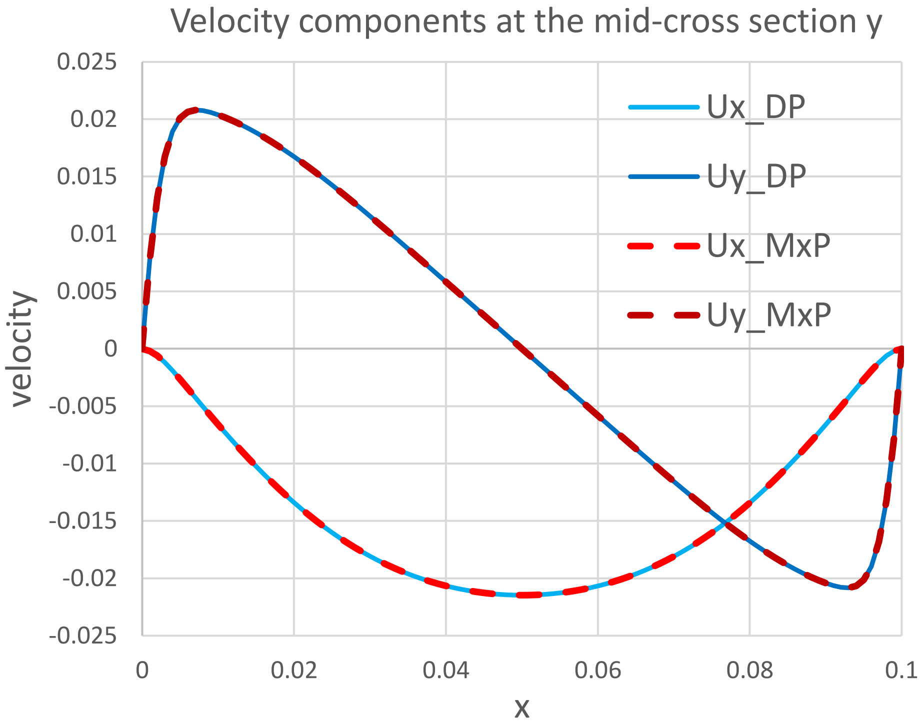


Pressure solver convergence: MxP vs DP (shown for first time step)

MxP: 3D Cavity - MXPC validation (2 of 2)

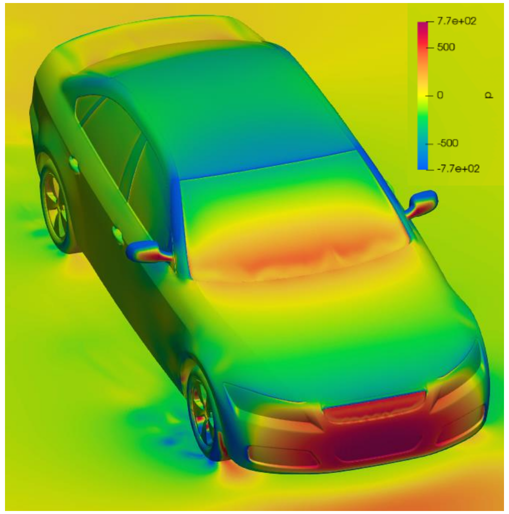






HUAWEI

MxP: DrivAer Benchmark with MXPC Solver



❑DrivAer B10 Benchmark

- DrivAer car model with stationary wheels and ground
- Solves steady-state RANS simulation with simpleFOAM
- Part of OpenFOAM HPC Challenge 1 software track

❑Setup

- Baseline: Pressure: DP GAMG Velocity: DP smoothsolver
- MxP: Pressure: MXPC + SP GAMG Velocity: MXPC + SP smoothsolver

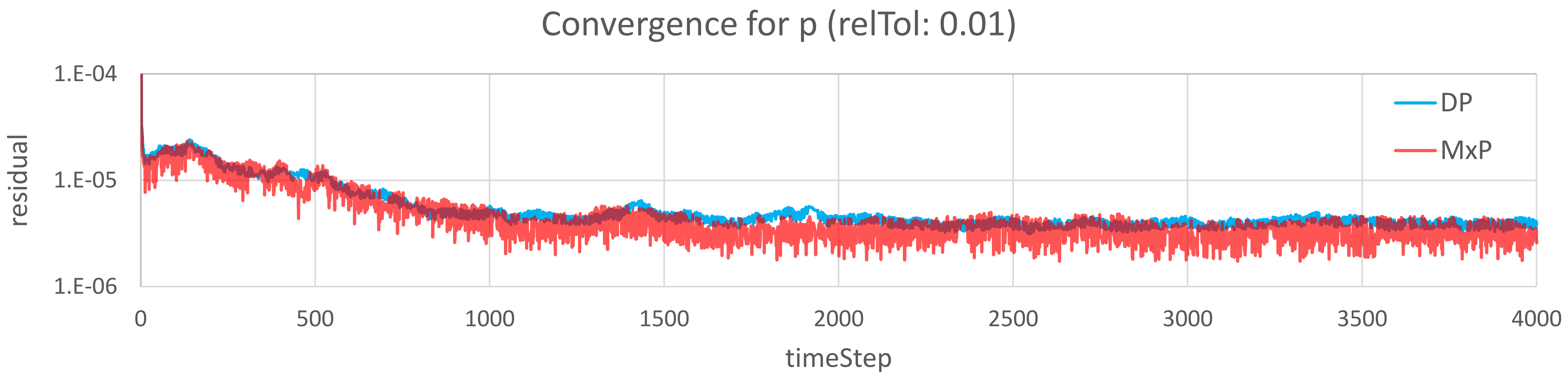
❑MXPC obtained consistent speedup while maintaining DP accuracy

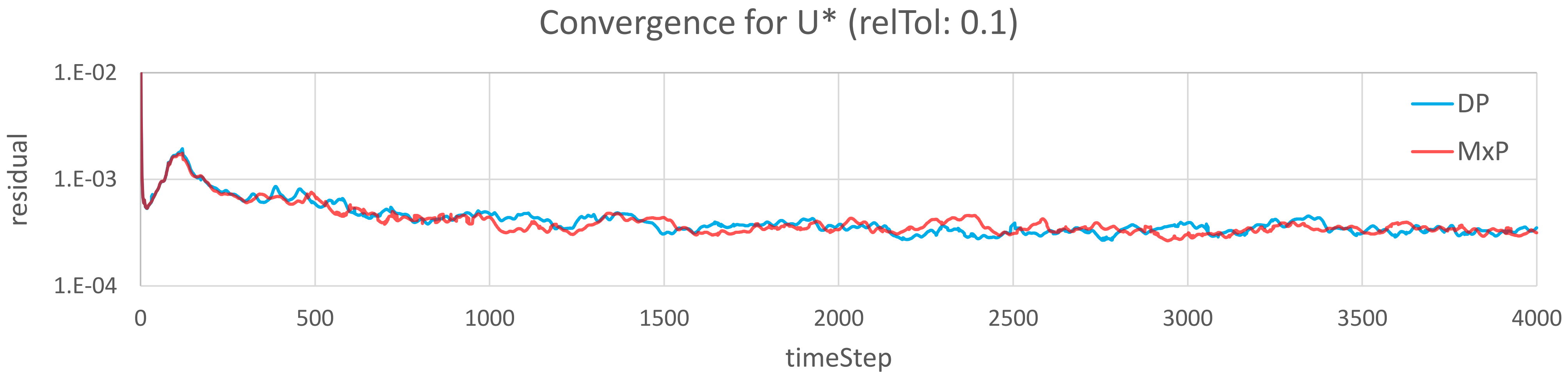
DrivAer mesh:	236M (fine)	**p relTol:**	0.01
Timesteps:	4000	**U relTol:**	0.1
Nodes:	8	**Cores per node:**	512
	DP	**MxP**	**MxP Speedup**
Total ClockTime (s):	22552	18098	1.25
Total solve time p:	7834	4013	**1.95**
Total solve time Ux+Uy+Uz:	3353	2643	1.27

Hardware	
Node	2 LX2 CPU
LX2 pilot high-performance CPU	System on-chip >256 cores 2 Computing dies
Compute die memory	Off-die DDR On-package memory
Frequency	>1.3GHz
Interconnect	RDMA

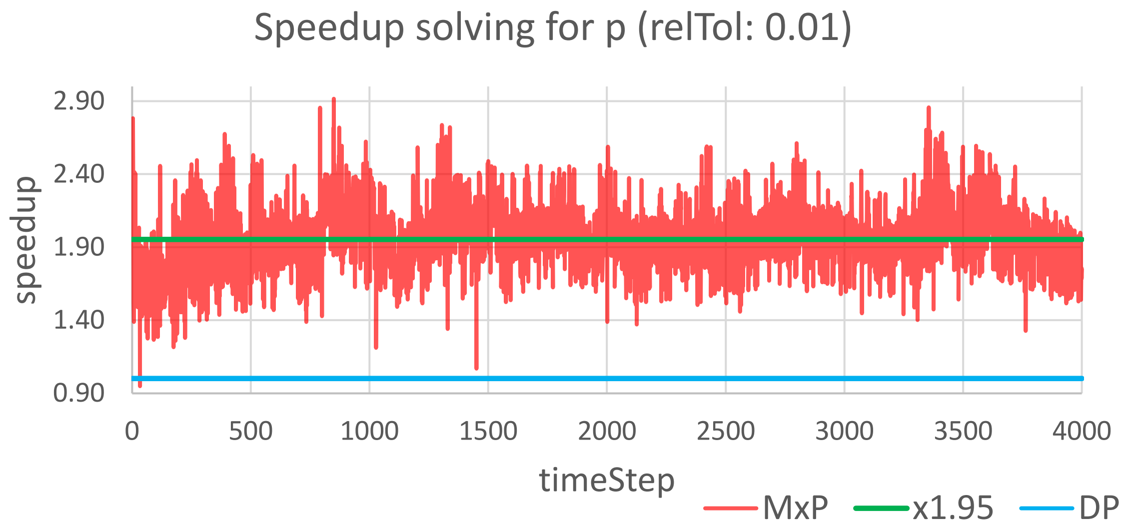
HUAWEI

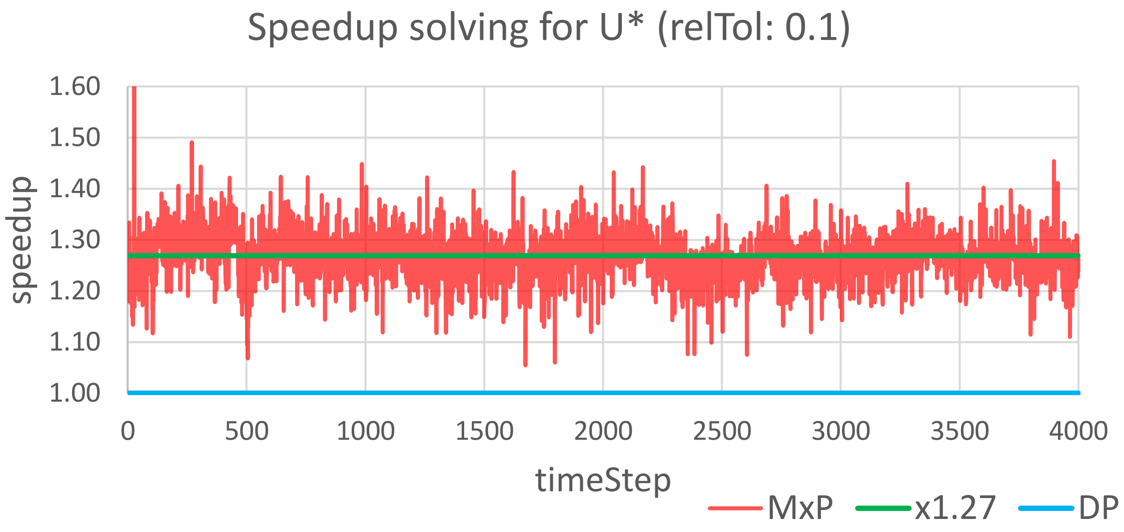
MxP: DrivAer Benchmark - convergence

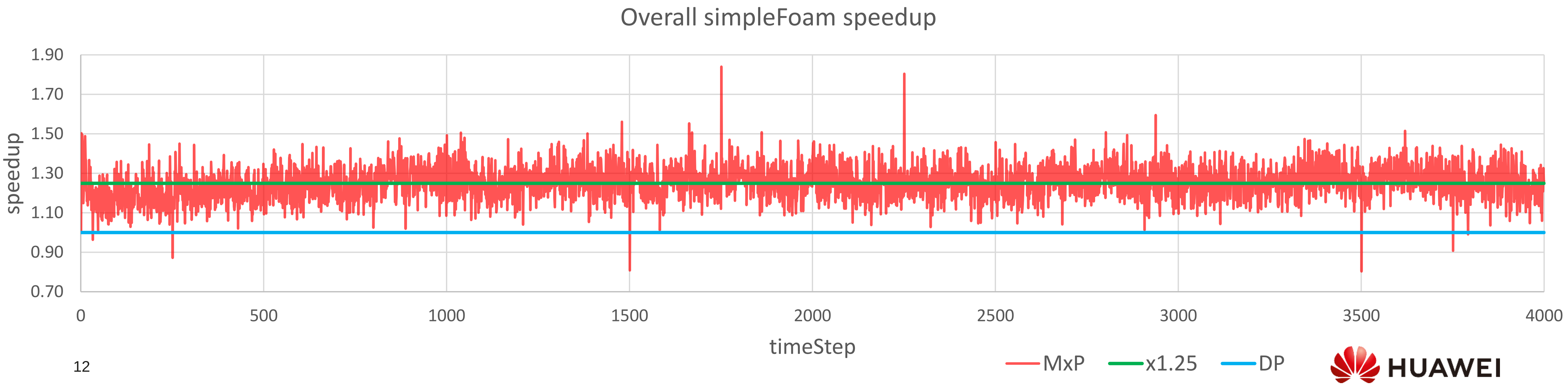




MxP: DrivAer Benchmark - MXPC speedup

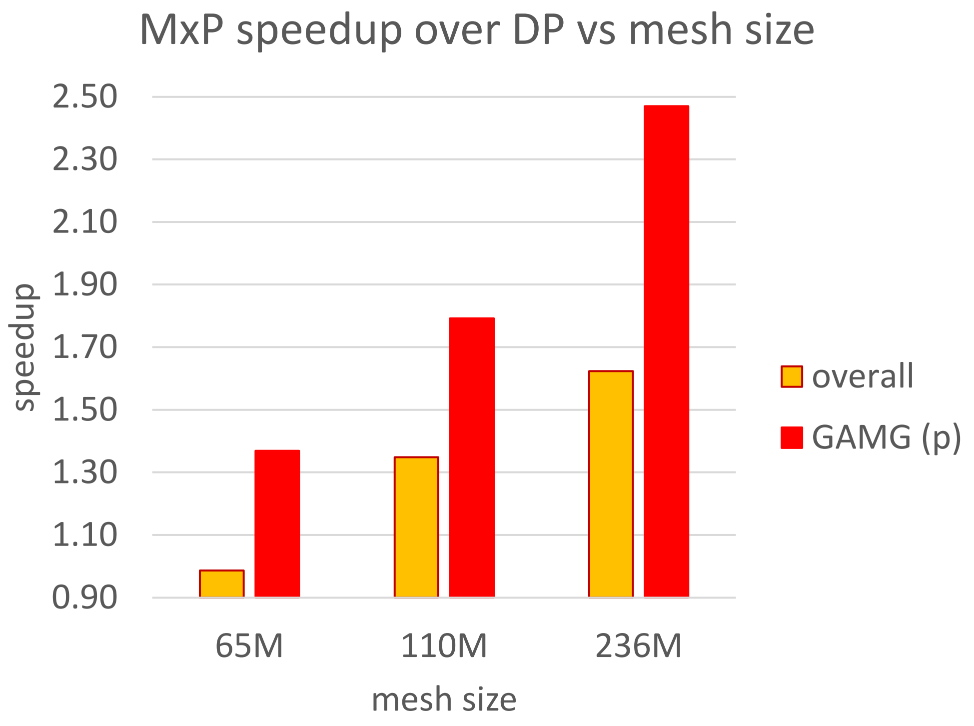


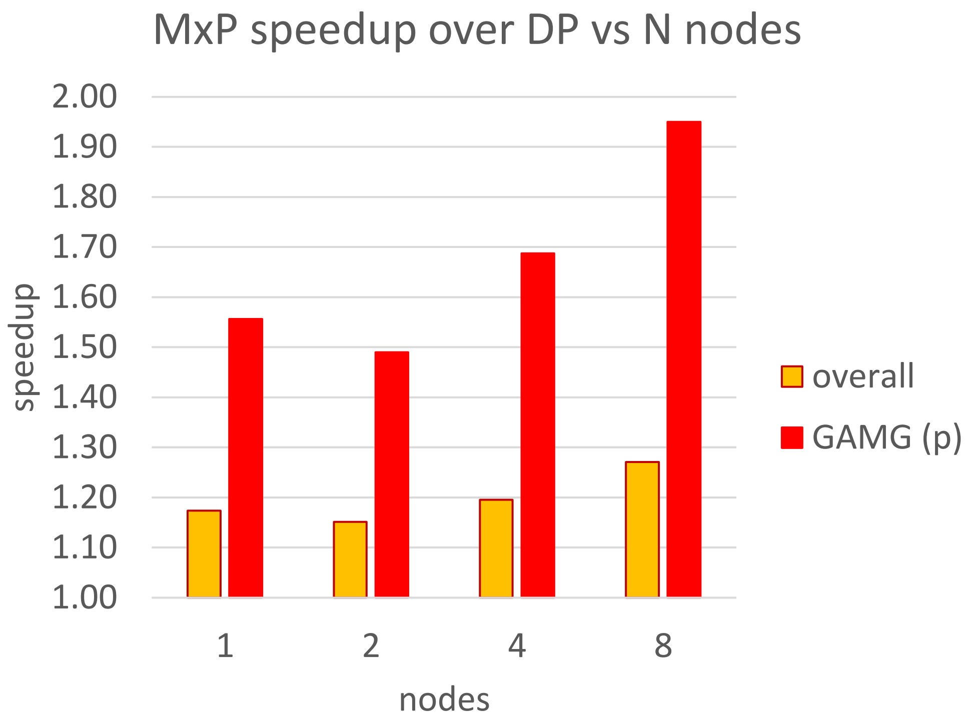




HUAWEI

MxP: DrivAer Benchmark - MXPC speedup vs mesh size & N nodes



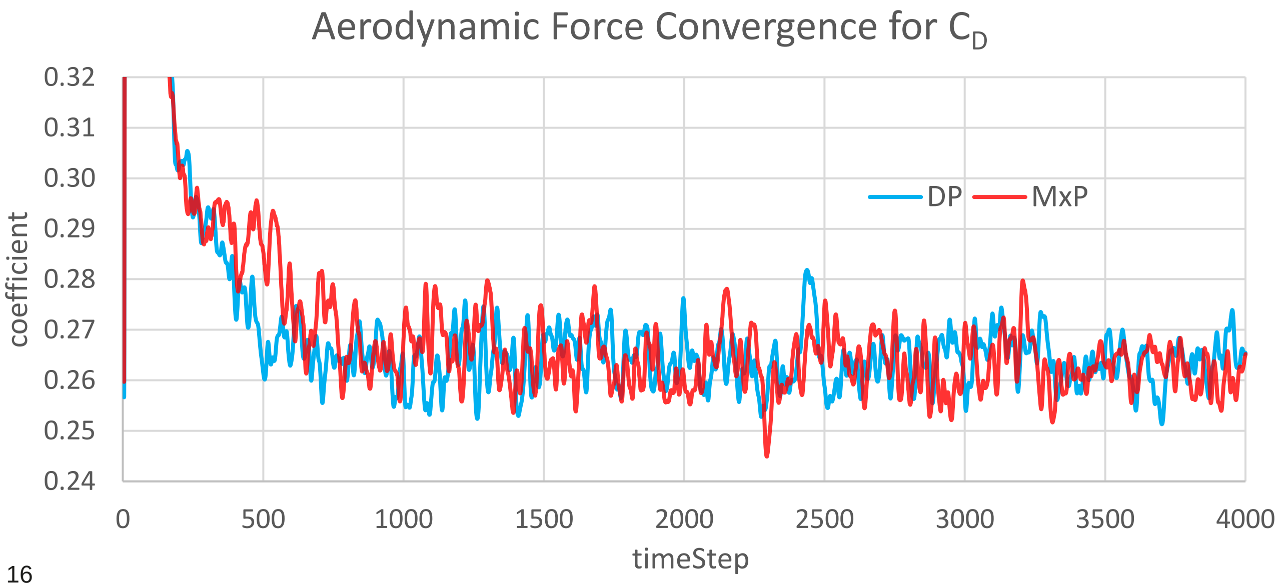


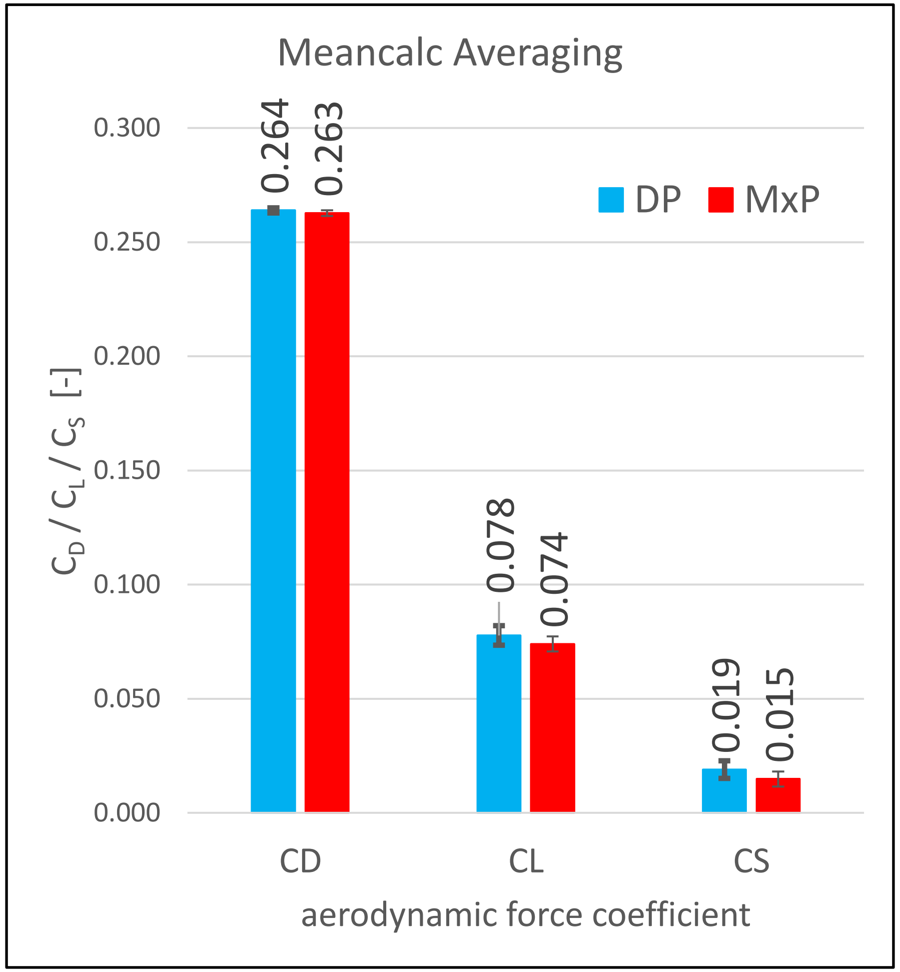
Note: Solving for p: relTol: 0.01 (inner relTol: 0.7)

HUAWEI

MxP: DrivAer Benchmark - MXPC accuracy

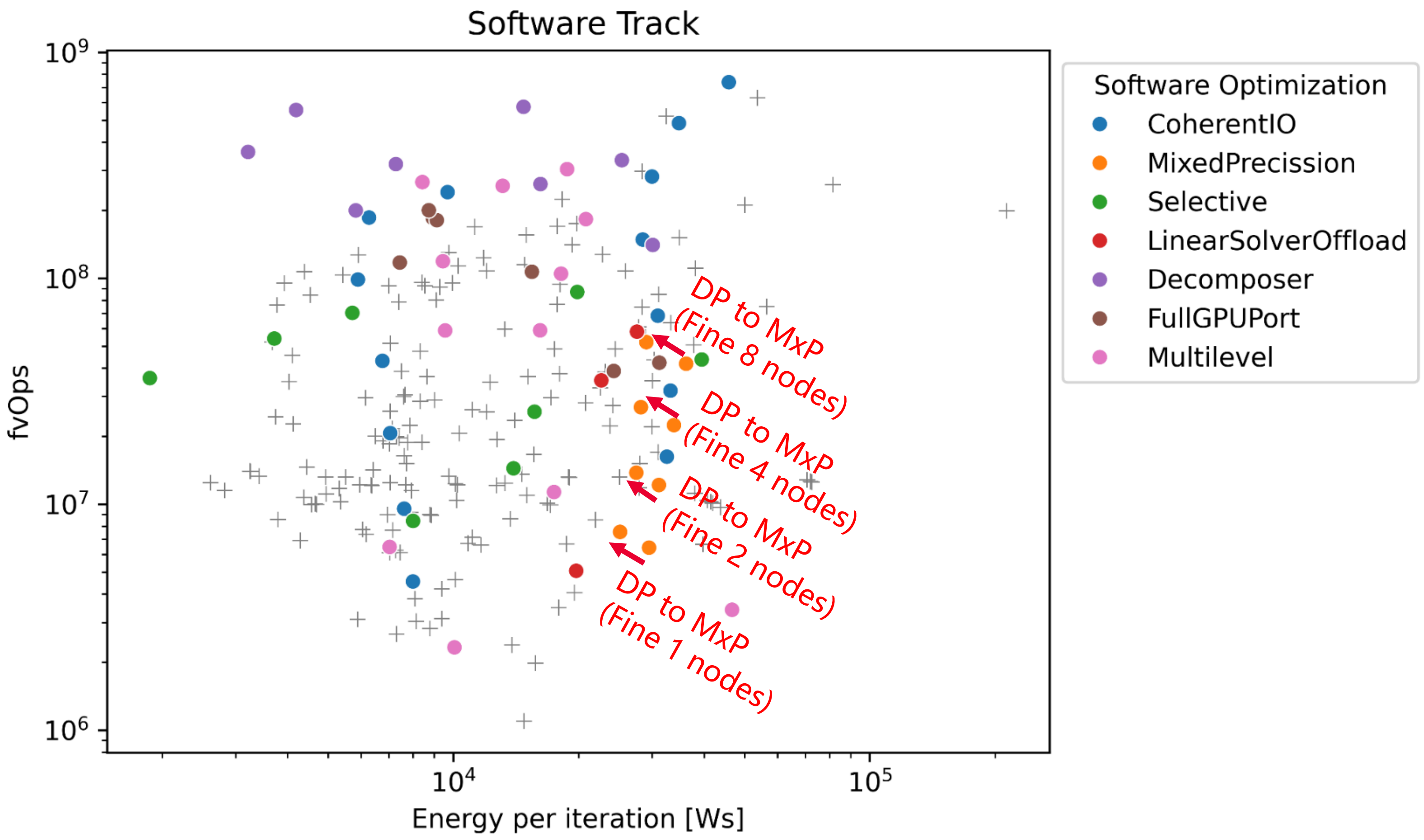
Standardized **Meancalc** based evaluation of aerodynamic forces						
	DP			MxP		
	C_D	C_L	C_S	C_D	C_L	C_S
Total timesteps [nread]	4000	4000	4000	4000	4000	4000
Averaging start timestep [nskip]	1719	1719	1719	1723	1723	1723
Averaging samples [nused]	2281	2281	2281	2277	2277	2277
Mean value μ	0.2639	0.0778	0.0190	0.2627	0.0740	0.0149
Error Mean Value $\sigma(\mu)$	0.00055	0.00213	0.00194	0.00065	0.00165	0.00166
95% conf. int. on mean $2*\sigma(\mu)$	0.0011	0.0043	0.0039	0.0013	0.0033	0.0033
Standard deviation s	0.0050	0.0202	0.0087	0.0052	0.0180	0.0088
Error Standard Deviation $\sigma(s)$	0.0004	0.0015	0.0014	0.0005	0.0012	0.0012





HUAWEI

MxP: DrivAer Benchmark - MXPC results



HUAWEI

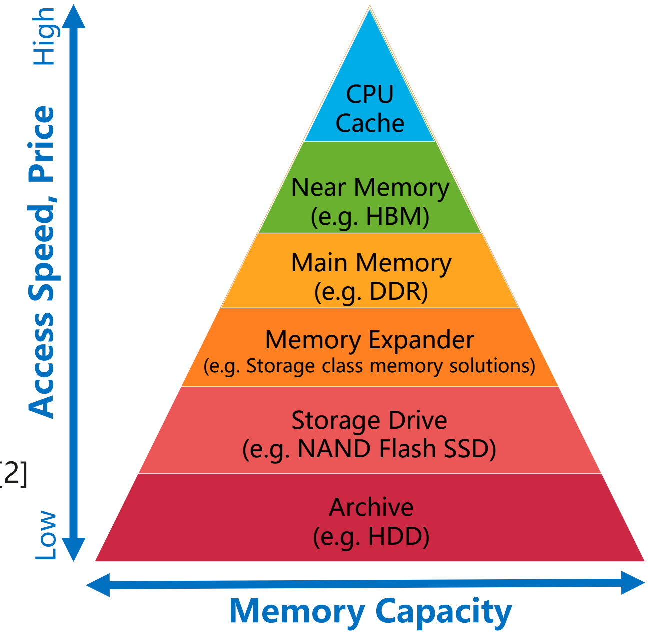
Smart Memory Allocator (SMA) - Overview

❑ OpenFOAM performance is often memory-bound due to

- Irregular access pattern to large field arrays
- Low computational intensity and lack of kernel fusion

❑ Leveraging faster memory tiers can alleviate this bottleneck

- Allow data-delivery rates to better align with computational throughput
- Demonstrated by AMD[1] on MI300A while using only 128 GB unified HBM[2]
- Larger mesh → more machines → higher cost and energy consumption



❑ Smart Memory Allocator (SMA)

- Control memory tier assignment for each allocated buffer
- Memory bandwidth-bound kernel buffers → higher memory tier
- Maximize performance gain with limited capacity of higher-tier memory
- Obtain comparable performances while minimizing required number of machines
- Fewer machines required → reduce communication overhead

[1] ISC 2024 https://ieeexplore.ieee.org/abstract/document/10528925
[2] https://www.amd.com/en/products/accelerators/instinct/mi300/mi300a.html

HUAWEI

Smart Memory Allocator (SMA) - Implementation

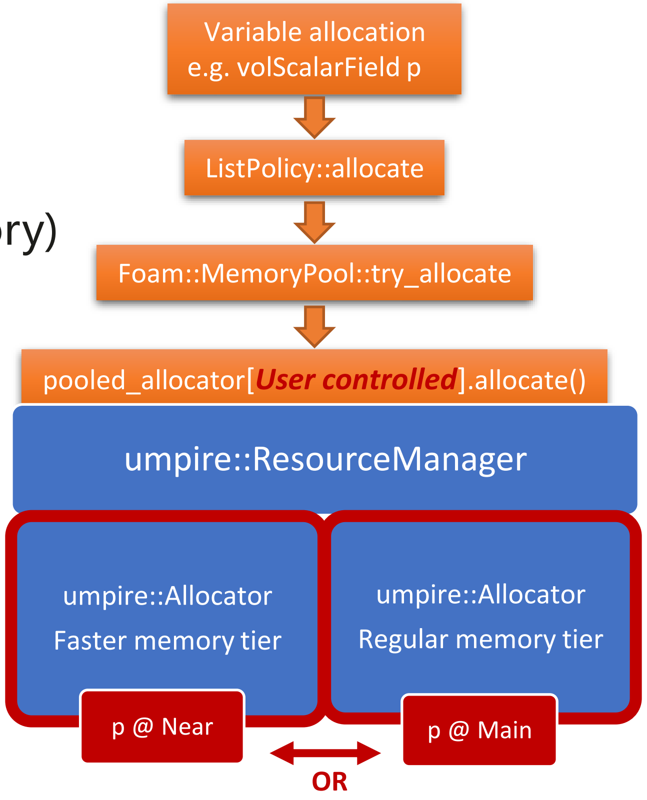
- **Based on *feature-memory_pool* branch (Mark Olesen)**
  - Memory resources (Lists/Matrix derived types) are management by Umpire library
  - Utilize pre-allocated memory buffer for constructed variables using single Umpire *allocator*
  - Acts as abstraction layer between application and underlying memory tier

Umpire

- **Extended framework to support two Umpire allocators**
  - Utilize Umpire built-in NUMA strategy
  - Each allocator manages a different memory-pool tier (Near / Main memory)

- **Variable allocation is directed to desired memory pool based on:**
  - Current compute kernel being executed
  - User-defined assignment of each compute kernel using environmental variables

- **Choose the best cost effective allocation for given mesh size and available resources**



HUAWEI

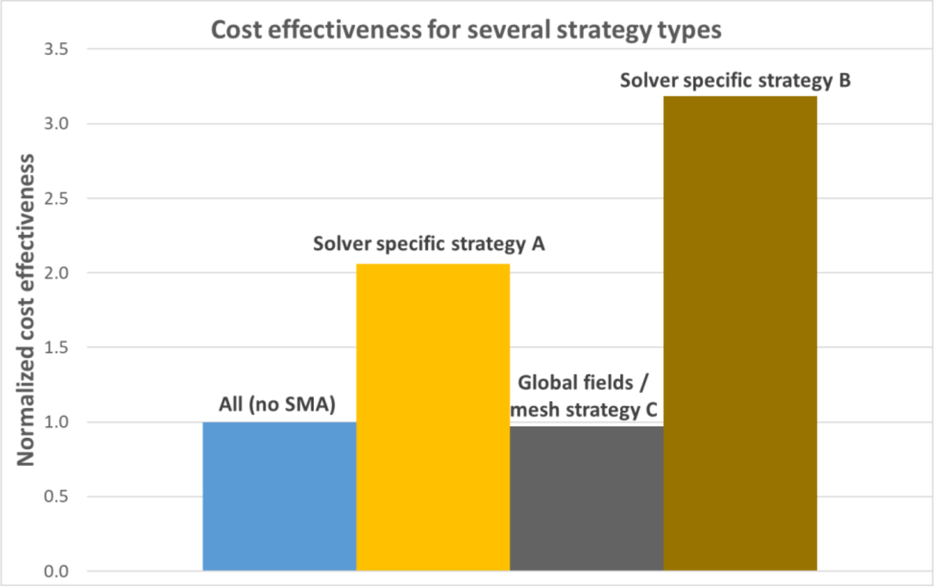
Smart Memory Allocator (SMA) - Results

❑Test Case

- DrivAer B10 benchmark used as part of the OHC-1 evaluation (coarse/medium/fine mesh)
- Executed on LX2 pilot high-performance CPUs
- Umpire allocators assigned to DDR and on-package memory tiers

❑Cost-effectiveness metric

- Decide which simpleFoam variables will be allocated on DDR / on-package memory tier
- Aim to effectively utilize on-package memory capacity to minimize run time
- Compared multiple strategies with varying memory demands (All > B > A > C)



$$\text{Cost effectiveness} = \frac{\text{Acceleration}}{\text{Ratio of used on-package memory}}$$

$$\text{Acceleration} = \frac{\text{Reference wall clock time (all variables on DDR)} - \text{Wall clock time for given strategy}}{\text{Reference wall clock time (all variables on DDR)}}$$

$$\text{Ratio of used on-package memory} = \frac{\text{Variables allocated on on-package memory for given strategy } [GB]}{\text{All variables allocated on on-package memory } [GB]}$$

$$\text{Normalized cost effectiveness} = \frac{\text{Cost effectiveness for given strategy}}{\text{Cost effectiveness of reference (all variables on on-package memory)}}$$

HUAWEI

Smart Memory Allocator (SMA) - Results

❑ Speedup

- Consistent speedup over DDR
- Retains performance compared to pure on-package memory execution

❑ Enabler

- Major enabler for scarce resources
- Coarse mesh on a single machine, Medium and fine meshes factors required #machines
- Utilization of on-package memory when capacity in not sufficient for pure run

❑ Energy efficient

- Fewer machines required + Retaining performance → High energy efficiency

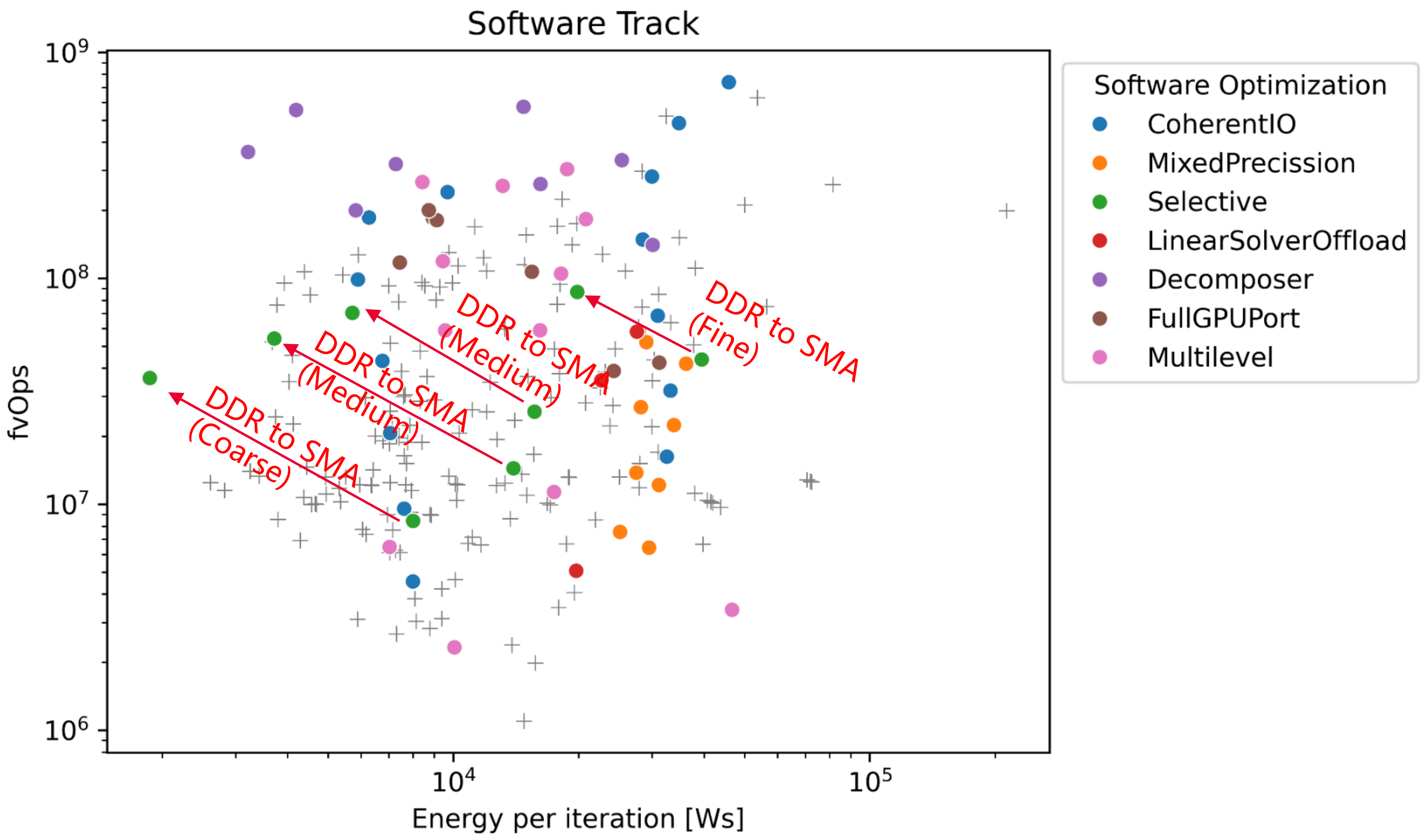
All (no SMA) | Solver specific strategy A | Solver specific strategy B

Mesh size	#Nodes	#Cores	WCT Speedup SMA vs. pure DDR	Total consumed energy [kW*s/step] Smart Memory Allocator	pure on-package memory
Coarse (~65M)	1	>512	4.3X	1.9 (1 node)	3.7 (4 nodes)
Medium (~110M)	1	>512	1.3X	11.5 (1 node)	6.6 (8 nodes)
	2	1024	3.7X	3.7 (2 nodes)	6.6 (8 nodes)
	4	2048	2.7X	5.7 (4 nodes)	6.6 (8 nodes)
Fine (~236M)	8	4096	2X	19.8 (8 nodes)*	17.7 (14 nodes)

HUAWEI

*Fine mesh energy demonstrates **further scaling** can be made to allow **higher efficiency** (8->10 nodes), exploiting the still existing memory bandwidth bottleneck

Smart Memory Allocator (SMA) - Results



Future Directions

- **Integrate selective memory allocation within MxP solver**
  - Allocate only single-precision buffers on faster memory tier
  - Accelerate SP solver and support larger simulations on limited on-package memory capacity
- **Integrate asynchronous solvers within OpenFOAM**
  - Reduce inter-process communication frequency → improve scalability
  - Obtain convergence in less iterations → reduce computation
- **Evaluate options for contributing the presented optimizations to the OpenFOAM community**