Hardware-oriented Numerics for Massively Parallel & Low Precision Accelerator Hardware and Application to „large scale" CFD Problems

***Faster* & *more reliable* predictions are needed…**

S. Turek & *FeatFlow* Team

Institute for Applied Mathematics (Chair LS III)

TU Dortmund University

https://wwwold.mathematik.tu-dortmund.de/lsiii

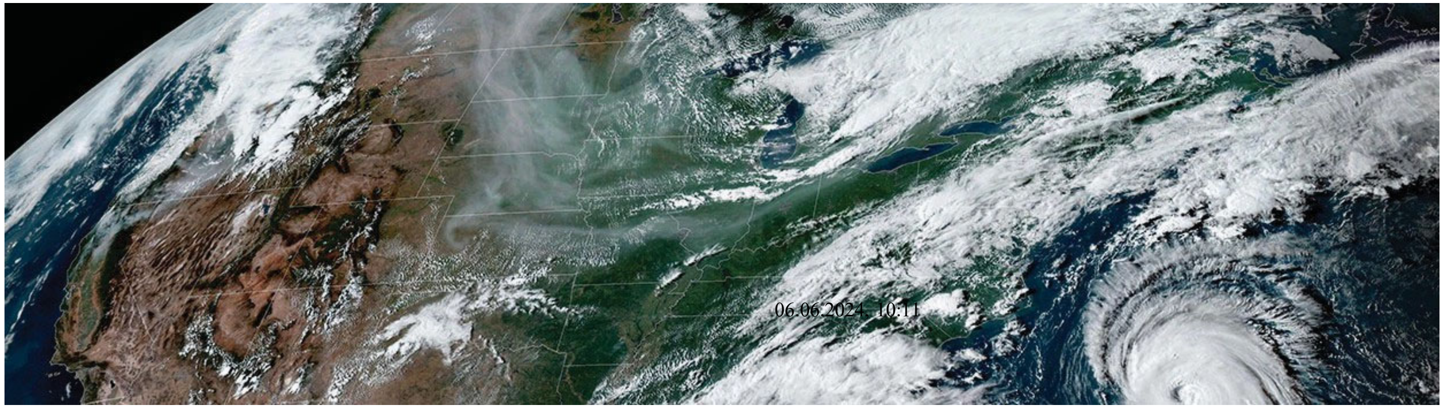
nature

NEWS | 04 June 2024

Superfast Microsoft AI is first to predict air pollution for the whole world

The model, called Aurora, also forecasts global weather for ten days — all in less than a minute.

By Carissa Wong



Weather forecasting is benefitting from the boom in artificial intelligence. Credit: NESDIS/STAR/NOAA/Alamy

An artificial intelligence (AI) model developed by Microsoft can accurately forecast weather and air pollution for the whole world —and it does it in less than a minute.

The model, called Aurora, is one of a slew of AI weather-forecasting tools being developed by tech giants, including GraphCast from Google DeepMind in London and FourCastNet from Nvidia, based in Santa Clara, California. But Aurora's ability to quickly predict air pollution globally is pioneering, say researchers.

"This, for me, is the first big step in a journey of atmospheric chemistry and machine learning," says machine-learning researcher Matthew Chantry at the European Centre for Medium-Range Weather Forecasts (ECMWF) in Reading, UK.

[DeepMind AI accurately forecasts weather — on a desktop computer]

Conventional weather forecasting uses mathematical models of physical processes in the atmosphere, land and sea. To predict air-pollution levels, researchers have previously used machine learning along with conventional mathematical models, says Chantry. Aurora seems to be the first entirely AI model to generate a global pollution forecast — which is a much more complex task than weather forecasting, says Chantry. ← !!!

“That was the thing where I went: wow, that’s a really cool result,” he says. The benefit of AI models is that they often require less computational power to make predictions than do conventional models, says Chantry.

AI researcher Paris Perdikaris at Microsoft Research AI for Science in Amsterdam and his colleagues found that Aurora could in less than a minute predict the levels of six major air pollutants worldwide: carbon monoxide, nitrogen oxide, nitrogen dioxide, sulfur dioxide, ozone and particulate matter. Its predictions span five days. It can do it “at orders of magnitude smaller computational cost” than a conventional model used by the Copernicus Atmosphere Monitoring Service at the ECMWF, which predicts global air-pollution levels, the team wrote in a preprint[1] published on arXiv on 20 May. ← !!!

RELATED

How AI is improving climate

Aurora’s predictions were of a similar quality to those of the conventional model. Policymakers use such predictions to track air pollution and protect against the related health harms. Air pollution has been linked to an increased risk of asthma, heart disease and dementia. ← !!!

The researchers trained Aurora on more than a million hours of data from six weather and climate models. After training the model, the team tweaked it to predict pollution

forecasts

and weather globally. The model generates a ten-day global weather forecast alongside the air-pollution prediction.

The team says that, on some tasks, Aurora could outperform other AI weather-forecasting models, such as GraphCast —which can outperform conventional models and make global weather predictions in minutes. But it is too early to make a definitive comparison, says Chantry. "You'd have to spend a lot of time, and probably have access to the models themselves, to be able to really go into detail and say with some certainty that model A is better than model B," he says. ← !!!

Further research will reveal whether 'foundational' AI models trained on diverse data sets, such as Aurora, perform better than those trained on a single data set, such as GraphCast. "There's lots of cool science to be done," he says.

doi: https://doi.org/10.1038/d41586-024-01677-2

References

1. Bodnar, C. *et al.* Preprint at arXiv https://doi.org/10.48550/arXiv.2405.13063 (2024).

Reprints and permissions

My personal view:

We, the **MFM** (*Mathematical Fluid Mechanics*) & **CFD** (*Computational Fluid Dynamics*) **community**, have to work „harder", or differently, if we don't want to be displaced by **AI** (*Artifical Intelligence*)….

….because this could lead to (political) problems in research, teaching and industrial applications in the future (especially with regard to the **associated budgets, human and computer ressources**) …….☹

…..and it is bad for AI without **accurate training data**

Because, I am (still) convinced that the **combination of modern and powerful MFM & CFD tools** can (and must) provide:

- **more accurate simulations results**, for instance via **user-specific & goal-oriented a posteriori error control**

→ Theory: **OK**   Practical realization in „real life" cases: **Not yet**
→ My personal experience: appr. **10 – 100 more effort** needed than for one (1) simulation (which is in most cases fully nonstationary & 3D!)

- **more efficient results** due to **numerical, computational & algorithmic improvement** and **exploitation** of much faster **supercomputing power**

Here: I will mainly concentrate onto efficiency aspects!

Because, I am (still) convinced that the **combination of modern and powerful MFM & CFD tools** can (and must) provide:

- **more accurate simulations results**, for instance via **user-specific & goal-oriented a posteriori error control**

→ Theory: **OK** Practical realization in „real life" cases: **Not yet**
→ My personal experience: appr. **10 – 100 more effort** needed than for one (1) simulation (which is in most cases fully nonstationary & 3D!)

- **more efficient results** due to **numerical, computational & algorithmic improvement** and **exploitation** of much faster **supercomputing power**

Here: I will mainly concentrate onto efficiency aspects!
However: Accuracy might be more important w.r.t. AI

No. 1 in year	System	**Cores**	Rmax **(PFlop/s)**	Rpeak (PFlop/s)	Power (kW)
Nov **2024**	**El Capitan** - HPE Cray EX255a, AMD 4th Generation EPYC 24C 1.8GHz, AMD Instinct **MI300A,** DOE/NNSA/LLNL USA	**11,039,616**	1,742.00	2,746.38	29,581

Compare 2024 vs. 1996:

Factor (more than) 1.000.000

No. 1 in year	System	**Cores**	Rmax (**GFlop/s**)	Rpeak (GFlop/s)	Power (kW)
June **1996**	SR2201/1024, Hitachi University of Tokyo Japan	**1,024**	220.40	307.20	???

No. 1 in year	System	**Cores**	Rmax **(PFlop/s)**	Rpeak (PFlop/s)	Power (kW)
Nov **2024**	**El Capitan** - HPE Cray EX255a, AMD 4th Generation EPYC 24C 1.8GHz, AMD Instinct **MI300A**, DOE/NNSA/LLNL USA	**11,039,616**	1,742.00	2,746.38	29,581

Compare 2024 vs. 1996:

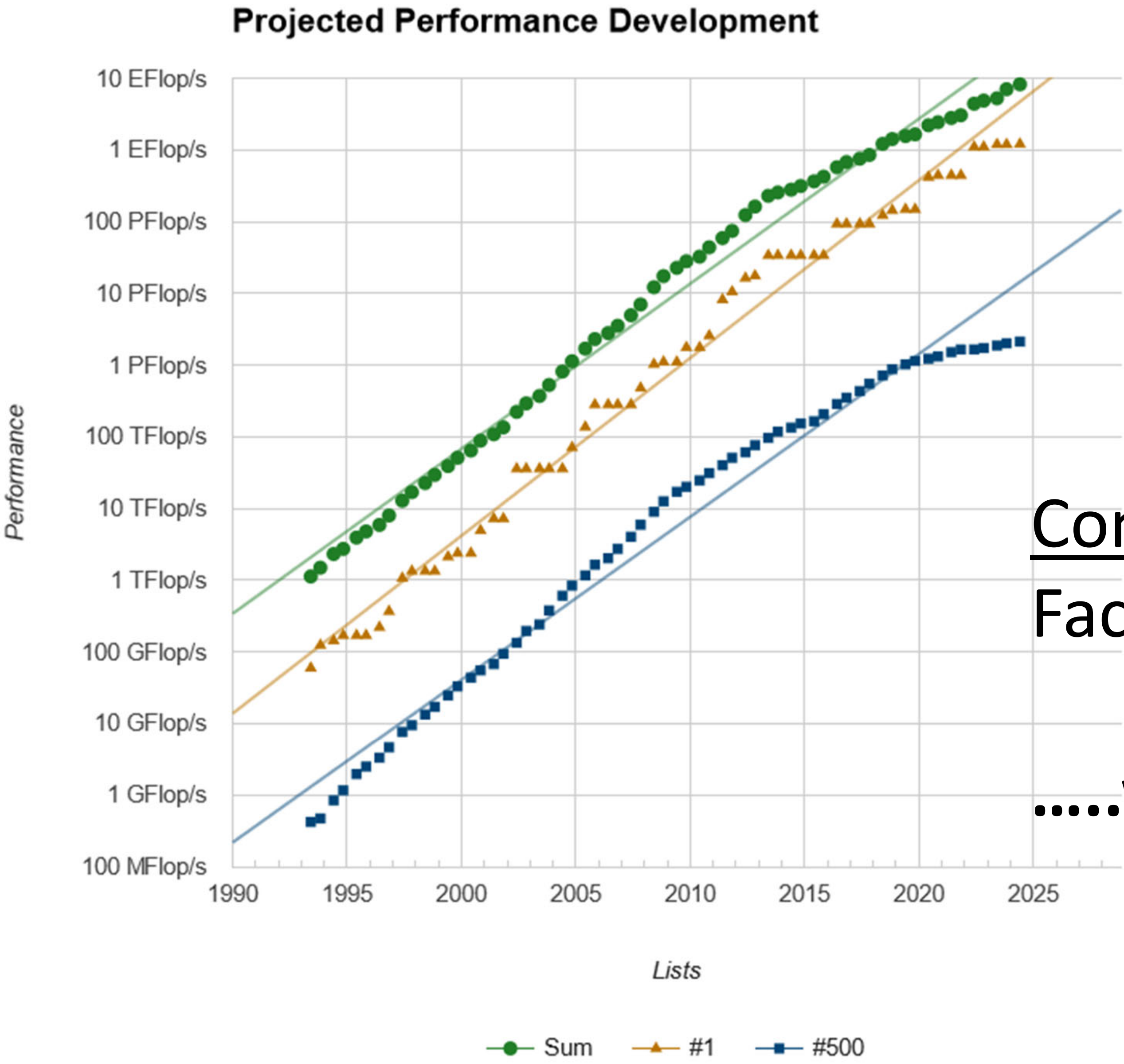
Factor (more than) 1.000.000

Colossus (Elon Musk, 3-4 Bill. Dollar):

100,000 x H100 (150 MW)

3.4 EFlop/s in FP64 (49.5 EFlop/s in TF32)

No. 1 in year	System	**Cores**	Rmax (**GFlop/s**)	Rpeak (GFlop/s)	Power (kW)
June **1996**	SR2201/1024, Hitachi University of Tokyo Japan	**1,024**	220.40	307.20	???

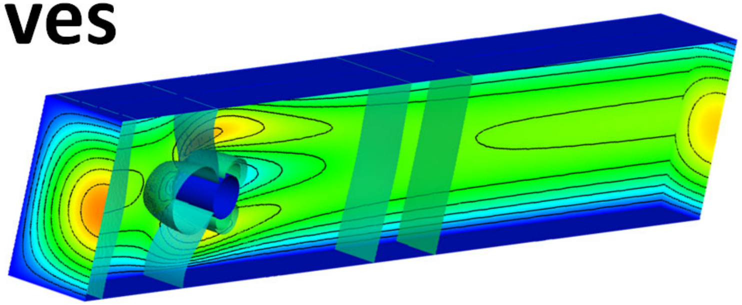


<u>Compare 2024 vs. 1996:</u>

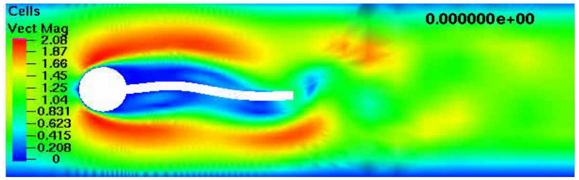
Factor (more than) 1.000.000

.....why 1996?

Start for 3 “successful” Benchmark Initiatives

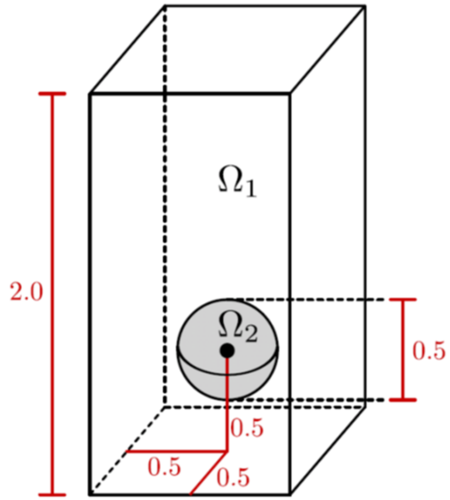


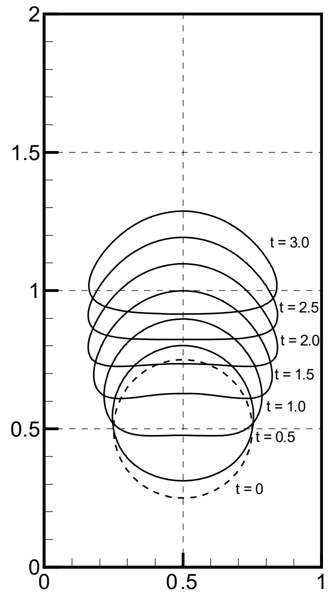
- **FAC: Flow Around Cylinder (2D + 3D) → 1996**
- **FSI**: Fluid-Structure-Interaction (2D.......and 3D) → 2006
- **RISING BUBBLE**: Multiphase Flow (2D.....and later also 3D) → 2009 and 2019



http://www.featflow.de/en/benchmarks/cfdbenchmarking.html

Important since <u>well-accepted</u> tools to evaluate the „realistic“ quality of AI, ML (PINN) or „unconventional“ tools (LBM, SPH)





FAC Benchmarks (1996): M. Schäfer, S. Turek, R. Rannacher et al

Stefan Turek
TU Dortmund
Mathematik

	Alle	Seit 2019
Zitate	14402	4812
h-index	54	30
i10-index	147	81

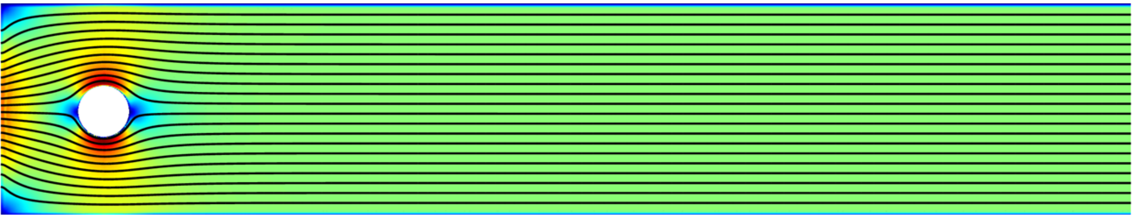
13 Artikel nicht verfügbar

95 Artikel verfügbar

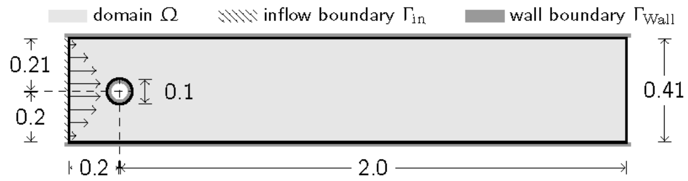
Basierend auf Fördermandaten

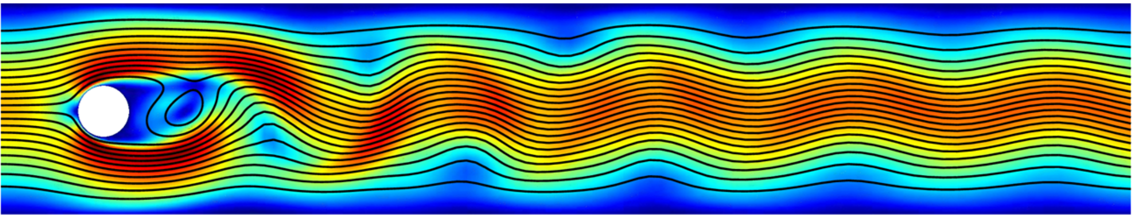
TITEL	ZITIERT VON	JAHR
Benchmark computations of laminar flow around a cylinder M Schäfer, S Turek, F Durst, E Krause, R Rannacher Flow simulation with high-performance computers II: DFG priority research …	1142	1996
Efficient solvers for incompressible flow problems: An algorithmic and computational approache S Turek Springer Science & Business Media	890	1999
Simple nonconforming quadrilateral Stokes element R Rannacher, S Turek Numerical Methods for Partial Differential Equations 8 (2), 97-111	889	1992
Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow S Turek, J Hron Fluid-Structure Interaction: modelling, simulation, optimisation, 371-385	829	2006
Artificial boundaries and flux and pressure conditions for the incompressible Navier–Stokes equations JG Heywood, R Rannacher, S Turek International Journal for numerical methods in fluids 22 (5), 325-352	793	1996
Quantitative benchmark computations of two-dimensional bubble dynamics S Hysing, S Turek, D Kuzmin, N Parolini, E Burman, S Ganesan, ... International Journal for Numerical Methods in Fluids 60 (11), 1259-1288	677	2009

The „Flow around a Cylinder“ Benchmarks (1996)

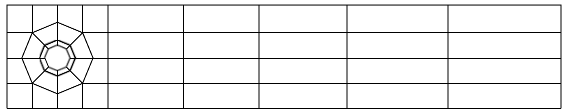


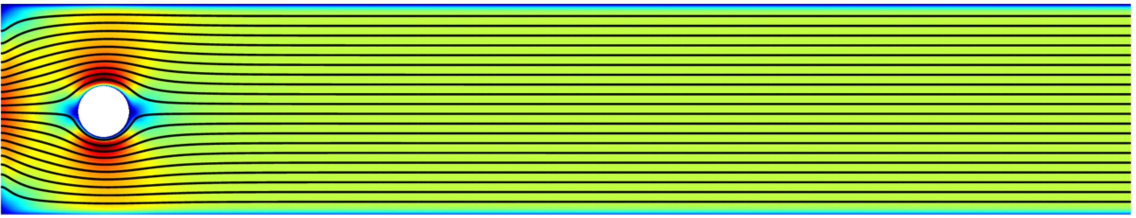
BENCH1: Re=20
(stationary)





BENCH2: Re=100
(periodical vortex
shedding)





BENCH3: Re(t)<=100
(for T <= [0,8s])

level	nodes	edges	elements	#dofs(**v**)	#dofs(p)	total
0	65	113	48	452K	65K	517K
1	226	418	192	1672K	226K	1898K
2	836	1604	768	6416K	836K	7252K
3	3208	6280	3072	25120K	3208K	28328K
4	12560	24848	12288	99392K	12560K	111952K

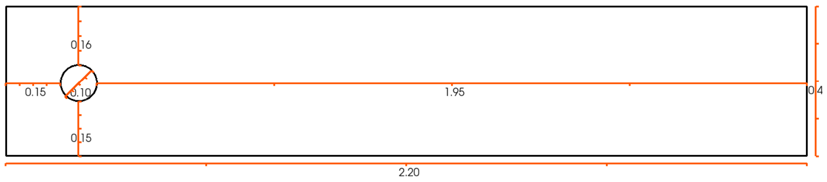
Simulations on IBM SP2 in 1996:
6 million (6e6) unknowns (in 3D)
(in hours, resp., 1 day)

Today: More than 1e13, that means 10 trillion unknowns should be possible???

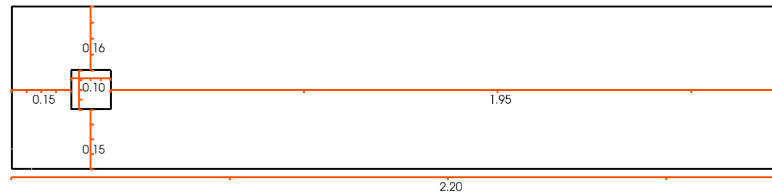
FAC Benchmarks (1996): M. Schäfer, S. Turek et al.

BENCH1: Re=20 → most recent calculations with more than **1 Billion unkowns** (in 2D and 3D)

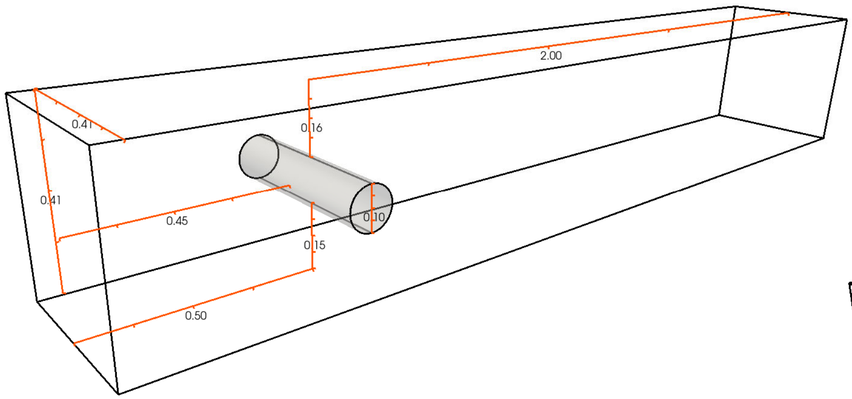
2D FAC:
Drag: 5.5795352338440[35:59]
Lift: 0.01061894814606[80:91]
P-Diff: 0.1175201[65:70]



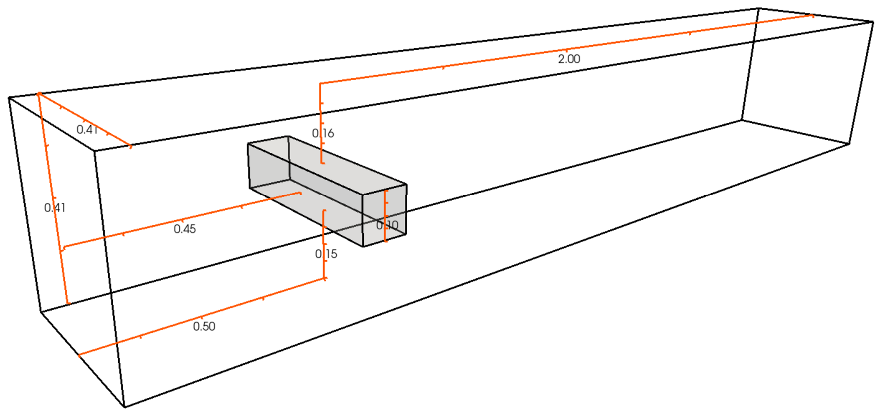
2D FAS:
Drag: 6.940[64:71]
Lift: 0.08619[00:32]
P-Diff: 0.12622[75:81]



3D FAC:
Drag: 6.18532[04:86]
Lift: 0.0094010[27:65]
P-Diff: 0.17[0999:1010]



3D FAS:
Drag: 7.7[69:72]
Lift: 0.069[05:16]
P-Diff: 0.1757[06:29]



FAC Benchmarks (1996): M. Schäfer, S. Turek et al.

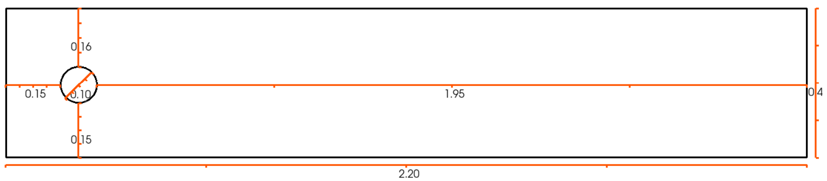
BENCH1: Re=20 → most recent calculations with more than **1 Billion unkowns** (in 2D and 3D)

2D FAC:
Drag: 5.5795352338440[35:59]
Lift: 0.010618948146069[80:91]
P-Diff: 0.1175201[65:70]

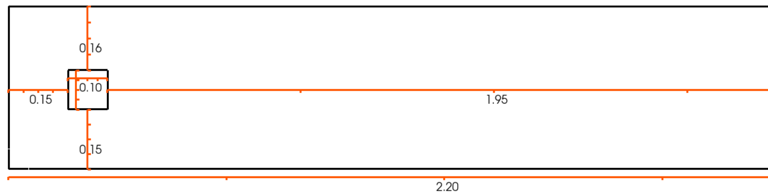
2D FAS:
Drag: 6.940[64:71]
Lift: 0.08619[00:32]
P-Diff: 0.12622[75:81]

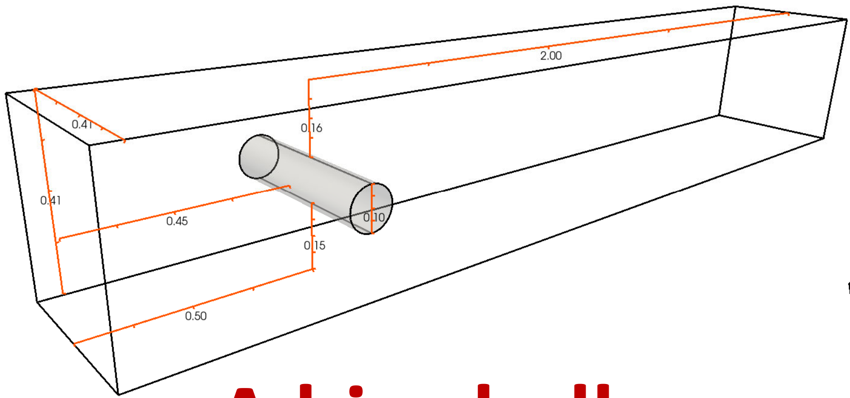
3D FAC:
Drag: 6.18532[04:86]
Lift: 0.0094010[27:65]
P-Diff: 0.17[0999:1010]

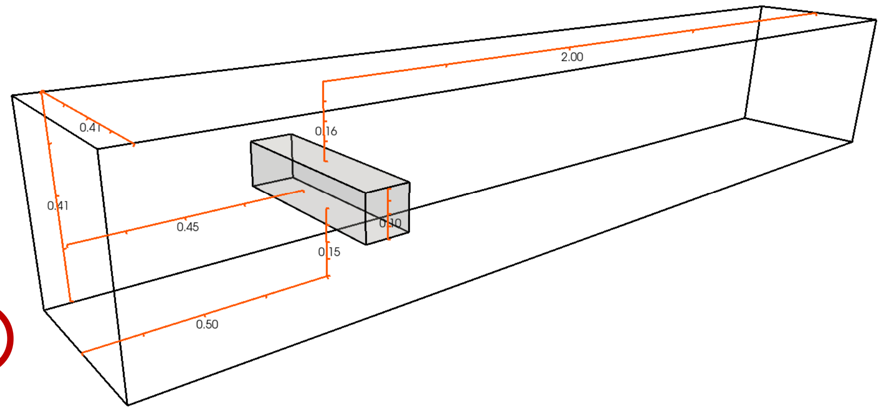
3D FAS:
Drag: 7.7[69:72]
Lift: 0.069[05:16]
P-Diff: 0.1757[06:29]



10 Billion unkowns are feasible







A big challenge even in 2025...☹

Why do our MFM & CFD techniques not scale appropriately with the obviously increasing compute power???

And what can we do from a numerical, computational & algorithmic perspective to realize much more efficient CFD simulation tools?

2 trends for HPC Hardware→ TOP500 November 24 (LINPACK)

Rank	System	#Cores	R_{max} (PFlop/s)	Accelerator
1	El Capitan	11,039,616	1,742	AMD MI300A
2	**Frontier**	9,066,176	1,353	AMD MI250X
3	Aurora	9,264,128	1,012	Intel Ponte Vecchio
4	**Eagle**	2,073,600	561	NVIDIA H100
5	HPC6	3,143,520	478	AMD MI250X
6	Fugaku	7,630,848	442	(A64FX)
7	Alps	2,121,600	435	NVIDIA GH200
8	LUMI	2,752,704	380	AMD MI250X
9	Leonardo	1,824,768	241	NVIDIA A100
10	Tuolumne	1,161,216	208	AMD MI300A

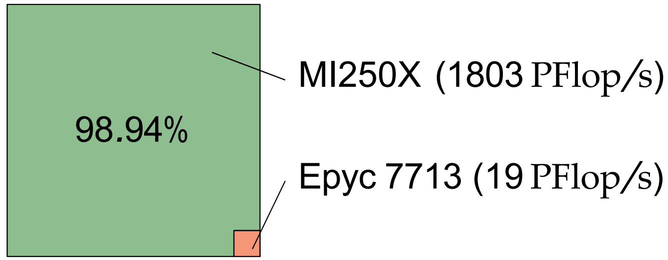
Exploiting massive parallelism: more than 1 million cores

Exploiting single node performance: special accelerators (NVIDIA, AMD)

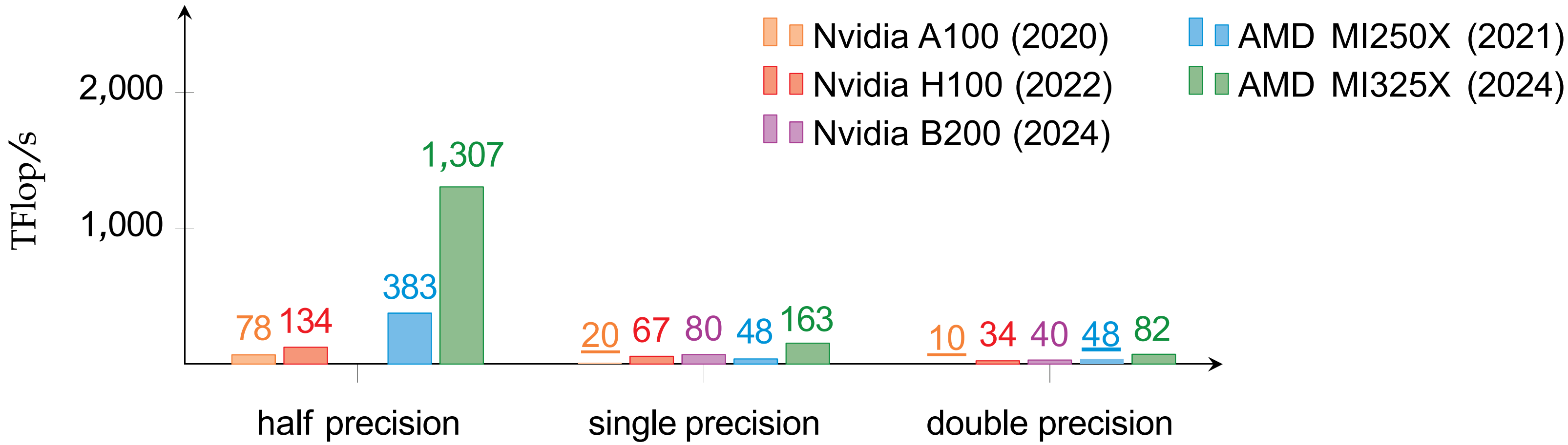
→ HPC compute power = #nodes x #TFLOP/s per node

Frontier supercomputer (8 881 152 cores)

- 9408 AMD Epyc 7713 (64 cores)
- 9408 · 4 AMD Instinct MI250X (220 cores)
- Leading TOP500 supercomputer (June 2024) ($R_{\max}$ = 1.2 EFlop/s)



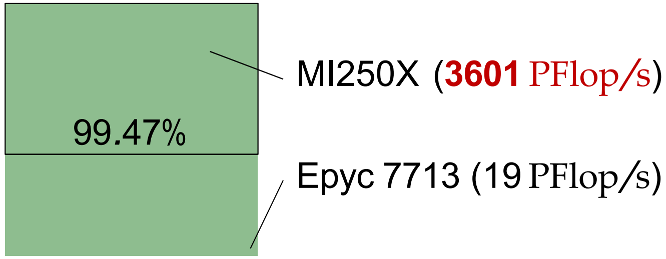
Theoretical performance of Frontier in double precision



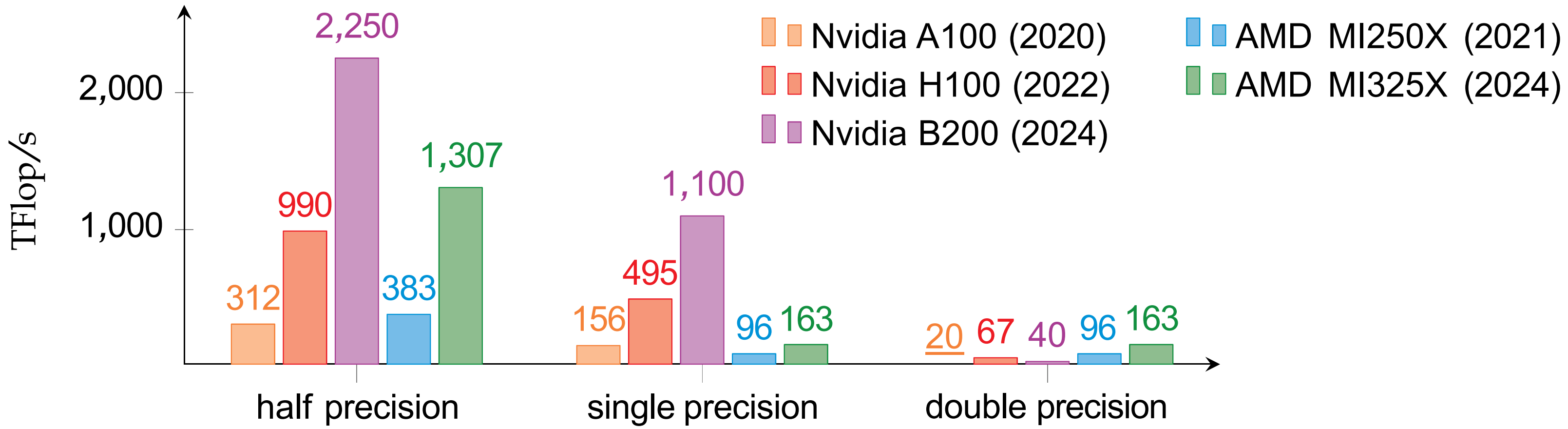
Without tensor cores (Nvidia) and matrix cores (AMD)

Frontier supercomputer (8 881 152 cores)

- 9408 AMD Epyc 7713 (64 cores)
- 9408 · 4 AMD Instinct MI250X (220 cores)
- Leading TOP500 supercomputer (June 2024) (R_{max} = 1.2 EFlop/s)



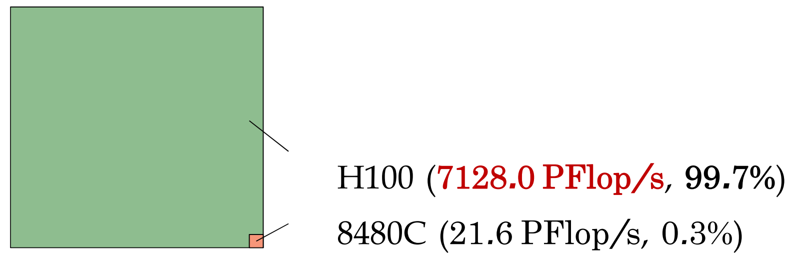
Theoretical performance of Frontier in double precision



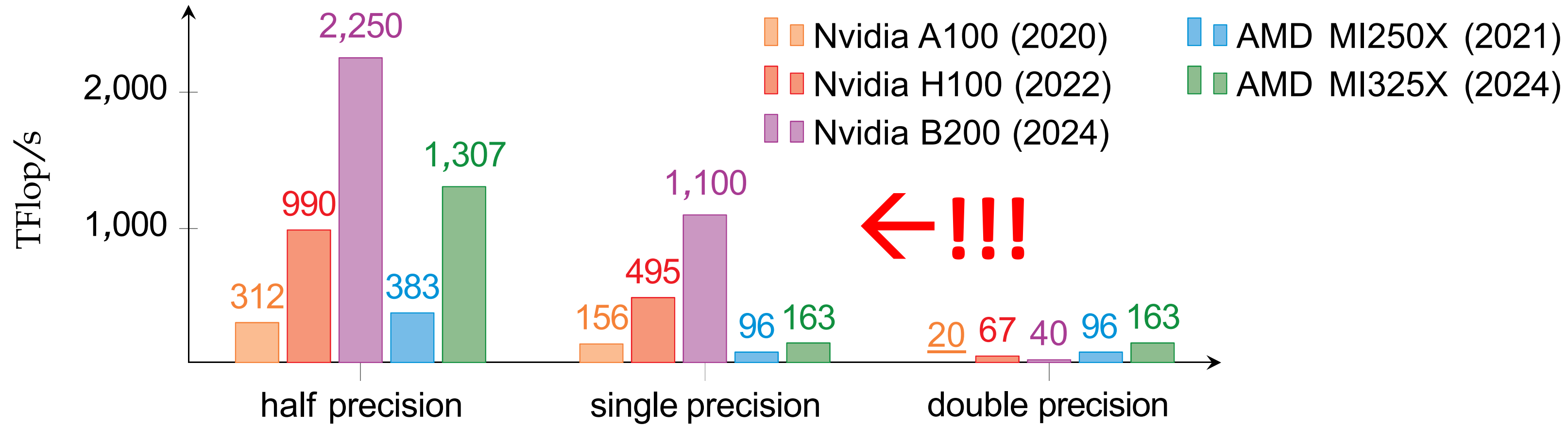
Using tensor cores (Nvidia) and matrix cores (**AMD**)

Eagle supercomputer (2 073 600 cores)

- 1800 x 2 INTEL XEON (48 cores)
- 1800 x 8 NVIDIA H100 (132 cores)
- Nr.3 TOP500 supercomputer (June 2024) ($R_{\max}$ = 0.56 EFlop/s)



Theoretical performance of Eagle in **single** precision



Using tensor cores (**Nvidia**) and matrix cores (AMD)

To be more precise:

How to exploit efficiently („that means with optimal computational and numerical efficiency“) not only **Massively Parallel** hardware, but also **Lower Precision Accelerator** hardware as major trends?

Important components:

→ **Prehandling** & **semi-iterative sparse-dense** solvers in **lower** precision
→ **Global-in-time Newton** & **Oseen** (= linearized NSE) solvers
→ **Parallel-in-time / Simultaneous-in-time Multigrid** approaches

To be more precise:

How to exploit efficiently („that means with optimal computational and numerical efficiency“) not only Massively Parallel hardware, but also Lower Precision Accelerator hardware as major trends?

How to realize for “large scale” CFD problems?

Important components:

→ **Prehandling & semi-iterative sparse-dense solvers in lower precision**
→ **Global-in-time Newton & Oseen (= linearized NSE) solvers**
→ **Parallel-in-time / Simultaneous-in-time Multigrid approaches**

Prototypical „large scale“ CFD simulations (I)

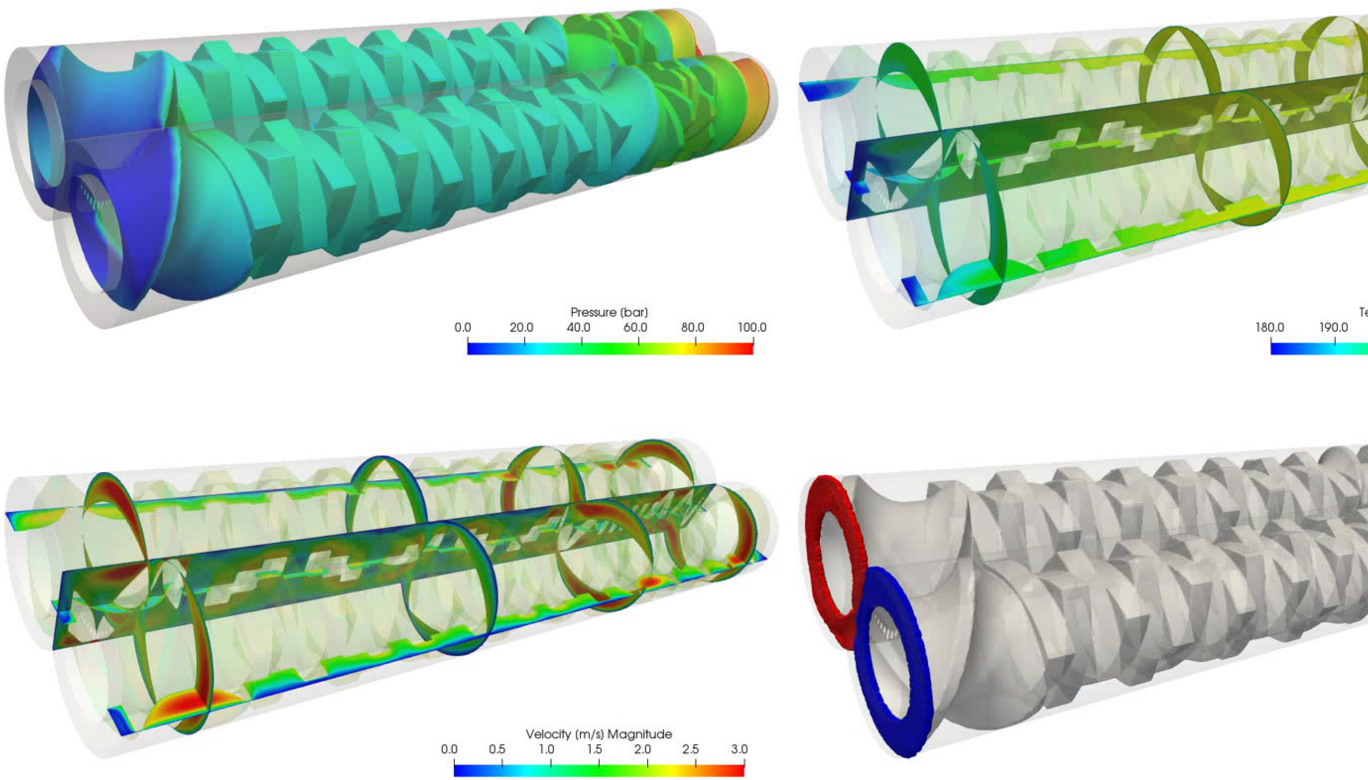
Navier-Stokes equations

$$\varrho(u_t + u \cdot \nabla u) - \nu \Delta u + \nabla p = g, \quad \nabla \cdot u = 0$$

→ (many) **Oseen-type Problems**
→ (very many) **Poisson Problems**
→ (very many) **Convection-Diffusion Problems**

Particularly on (almost) arbitrarily complex geometries in realistic applications

→ **Twin screws…**



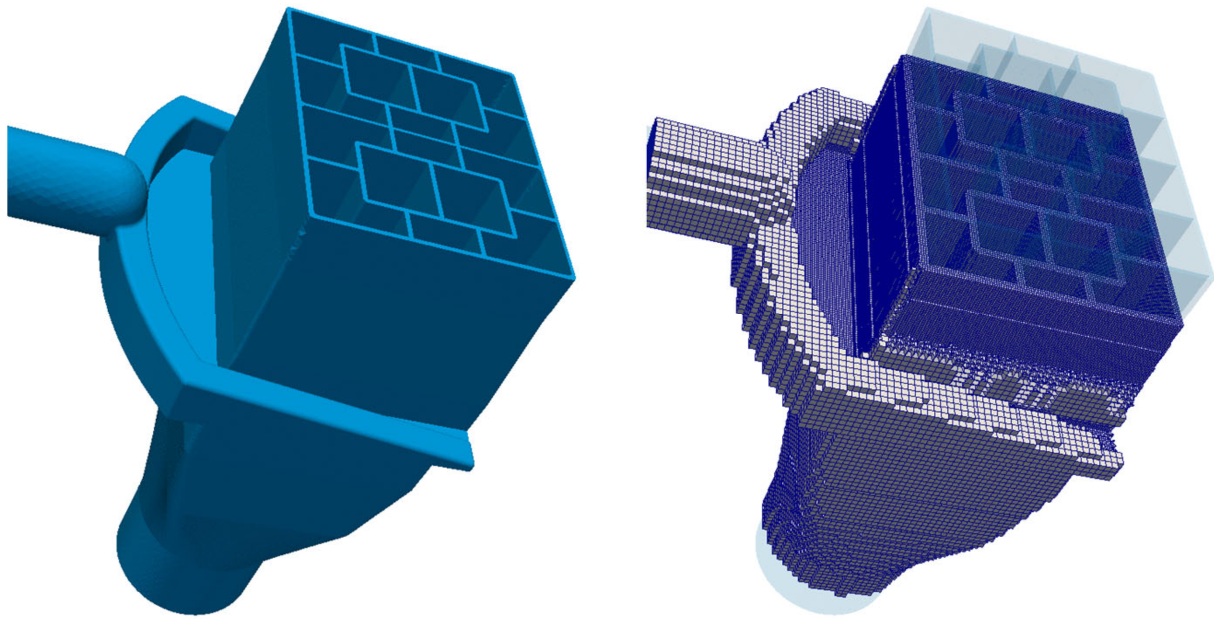
Prototypical „large scale“ CFD simulations (II)

Navier-Stokes equations

$$\varrho(u_t + u \cdot \nabla u) - \nu \Delta u + \nabla p = g, \quad \nabla \cdot u = 0$$

→ (many) **Oseen-type Problems**
→ (very many) **Poisson Problems**
→ (very many) **Convection-Diffusion Problems**

Particularly on (almost) arbitrarily complex geometries in realistic applications



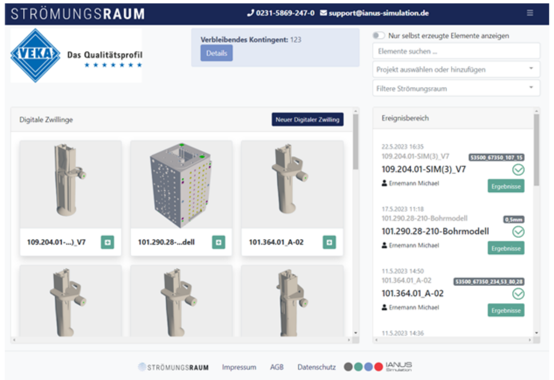
→ **Extrusion dies for polymer melts realized in *StrömungsRaum***
(software together with IANUS Simulation)

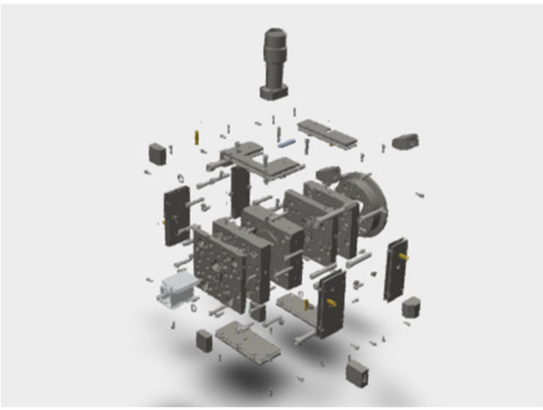
AIM: AUTOMATIC EXTRUSION SIMULATIONS IN *STRÖMUNGS**RAUM***

IANUS SIMULATION

1

PARAMETRIC GENERATION OF **DIGITAL TWINS**

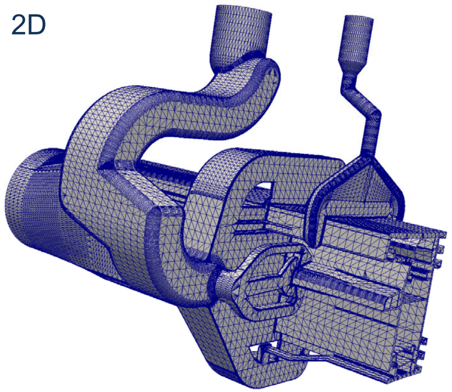




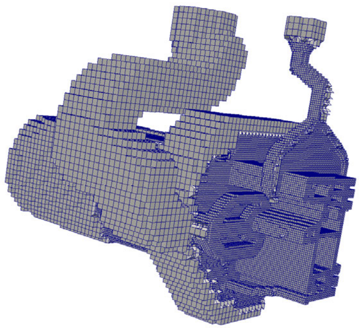
2

AUTOMATIC GENERATION **OF 2D SURFACES** AND 3D MESHES

2D



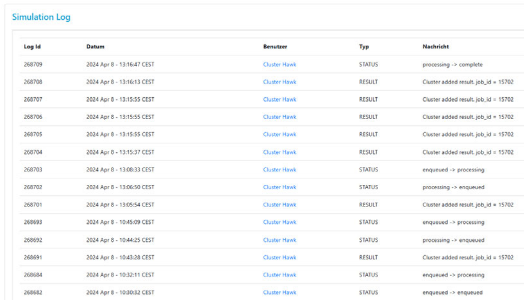
3D



3

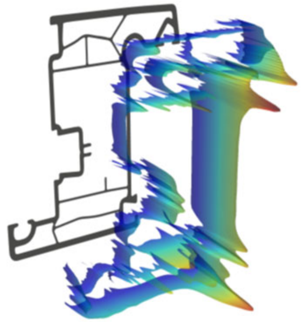
AUTOMATIC SIMULATION ON HPC HARDWARE

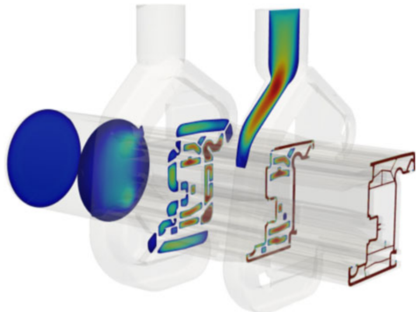
SUBMISSION AND LOGGING OF THE AUTOMATED SIMULATION SCRIPT



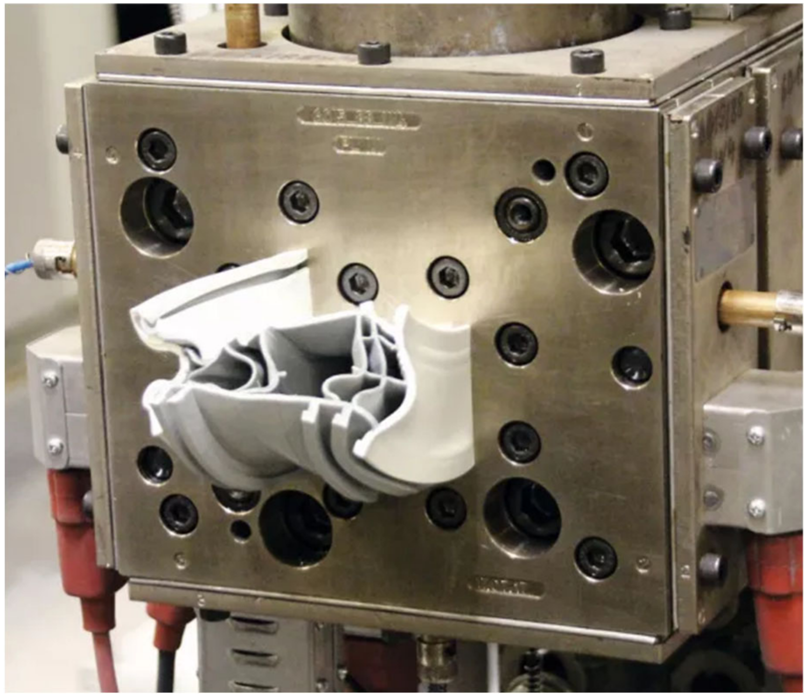
4

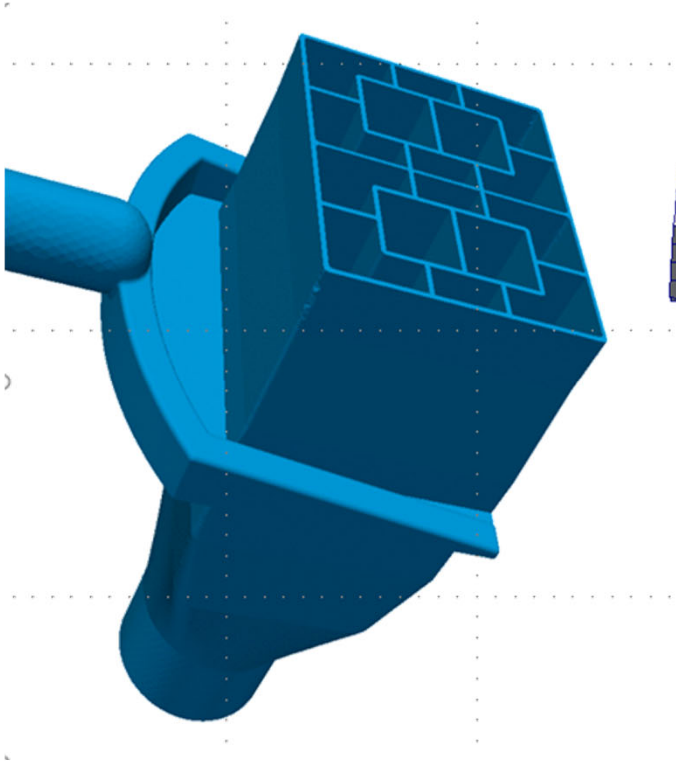
AUTOMATIC REPORTING OF RESULTS AND RECOMMENDATIONS

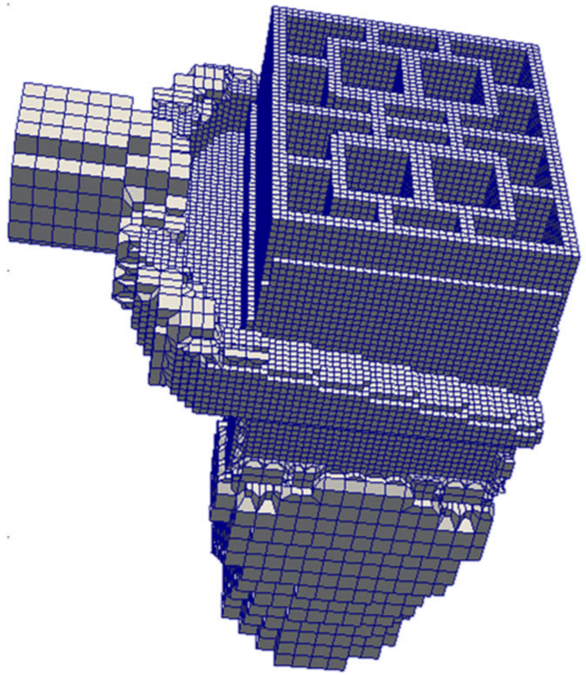




EXAMPLE: OPTIMIZATION OF EXTRUSION DIES





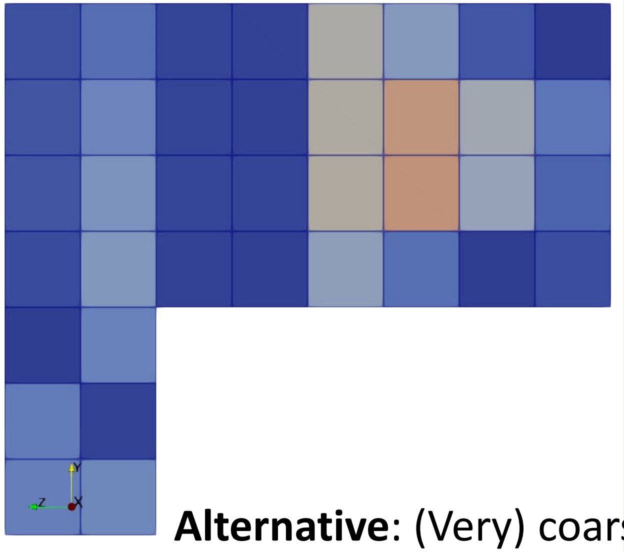


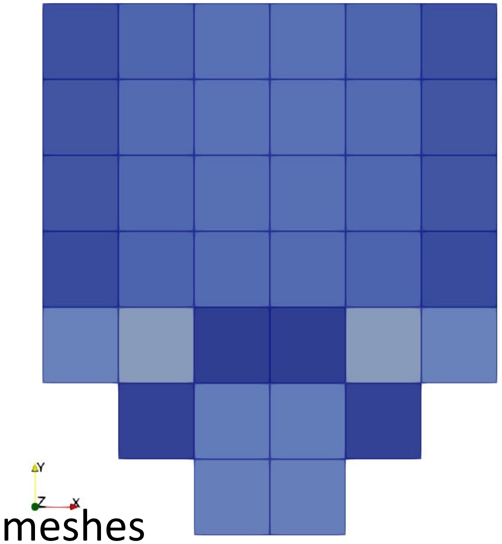
L1 : 68K elems	**~2M dofs**
L2 : 544K elems	~15M dofs
L3 : 4.35M elems	~117M dofs
L4 : 34.8M elems	**~940M dofs**

Problems:

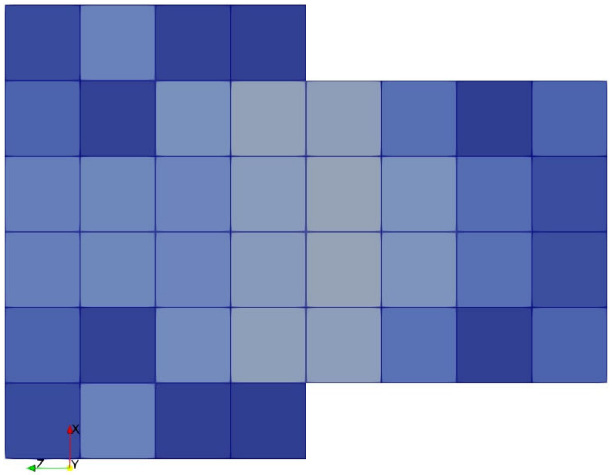
Highly resolved + very large **coarse meshes**

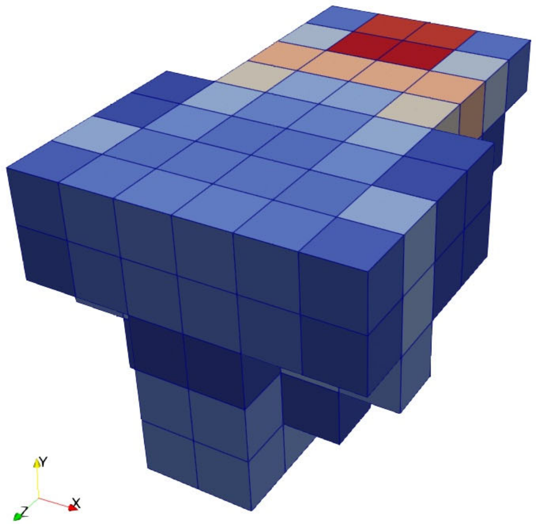
Many highly dimensional **problems (→ L5 !)**

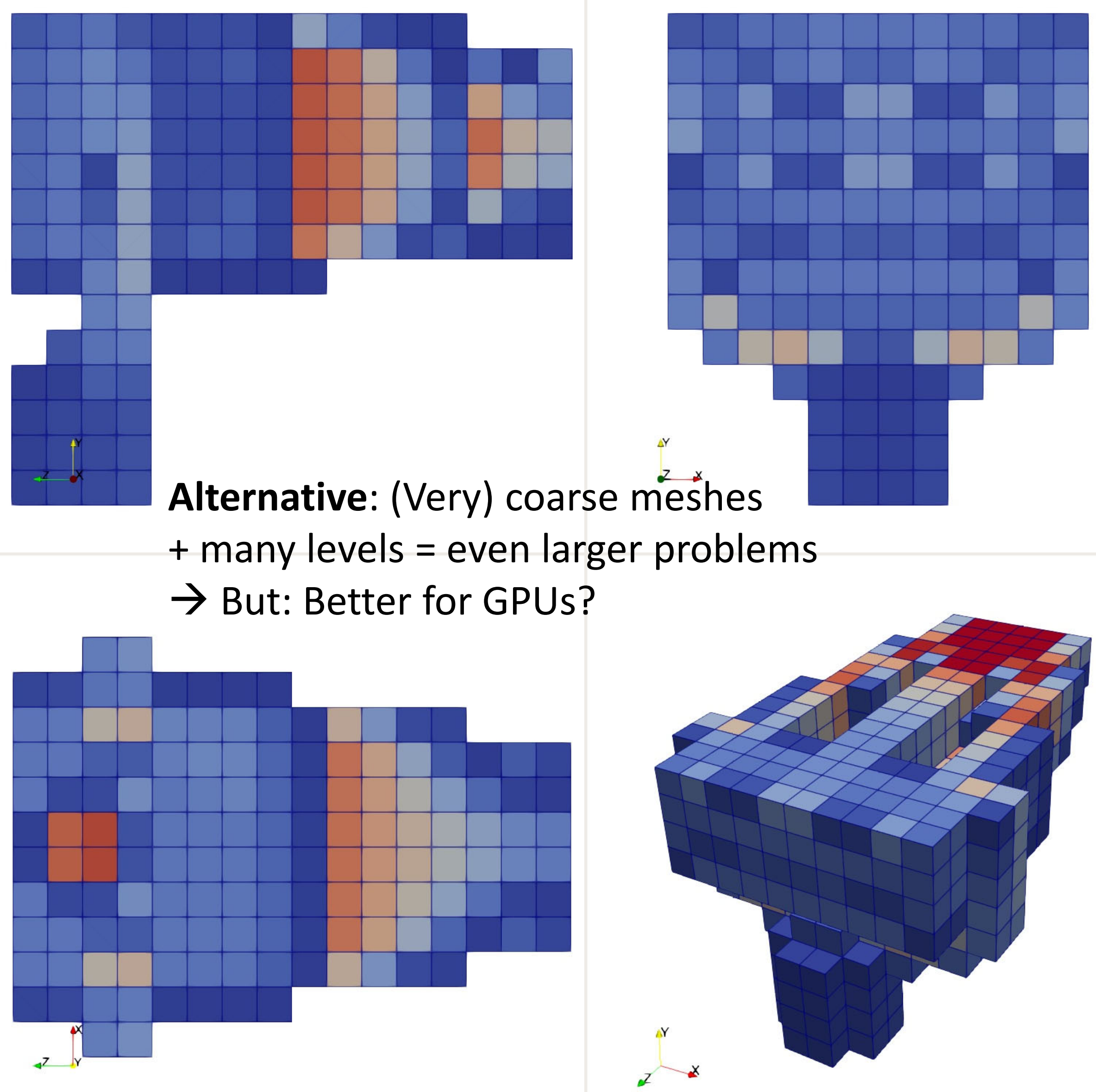




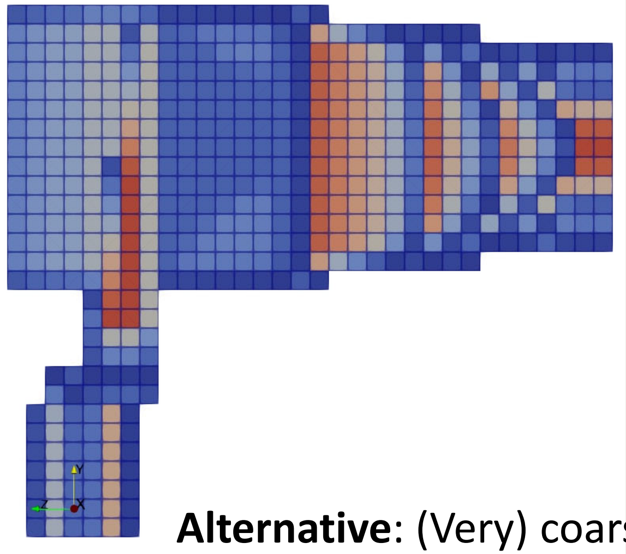
Alternative: (Very) coarse meshes
+ many levels = even larger problems
→ But: Better for GPUs?

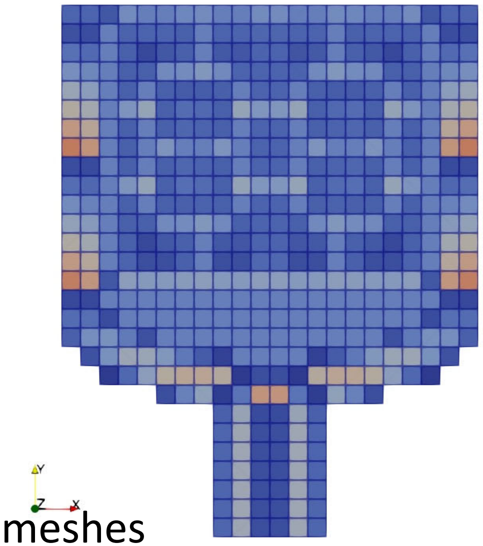




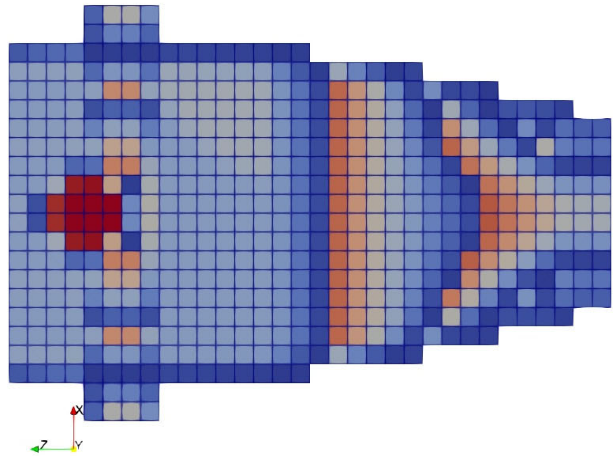


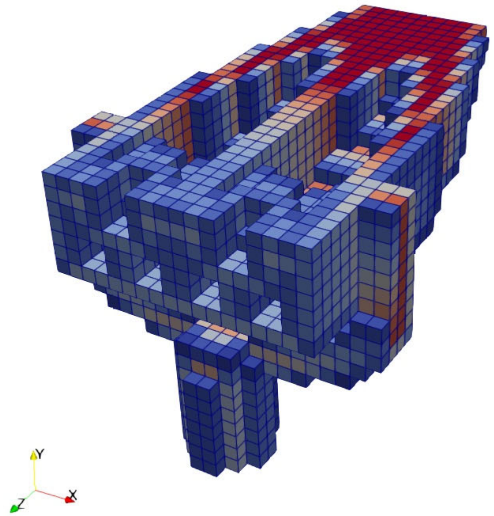
Alternative: (Very) coarse meshes
+ many levels = even larger problems
→ But: Better for GPUs?

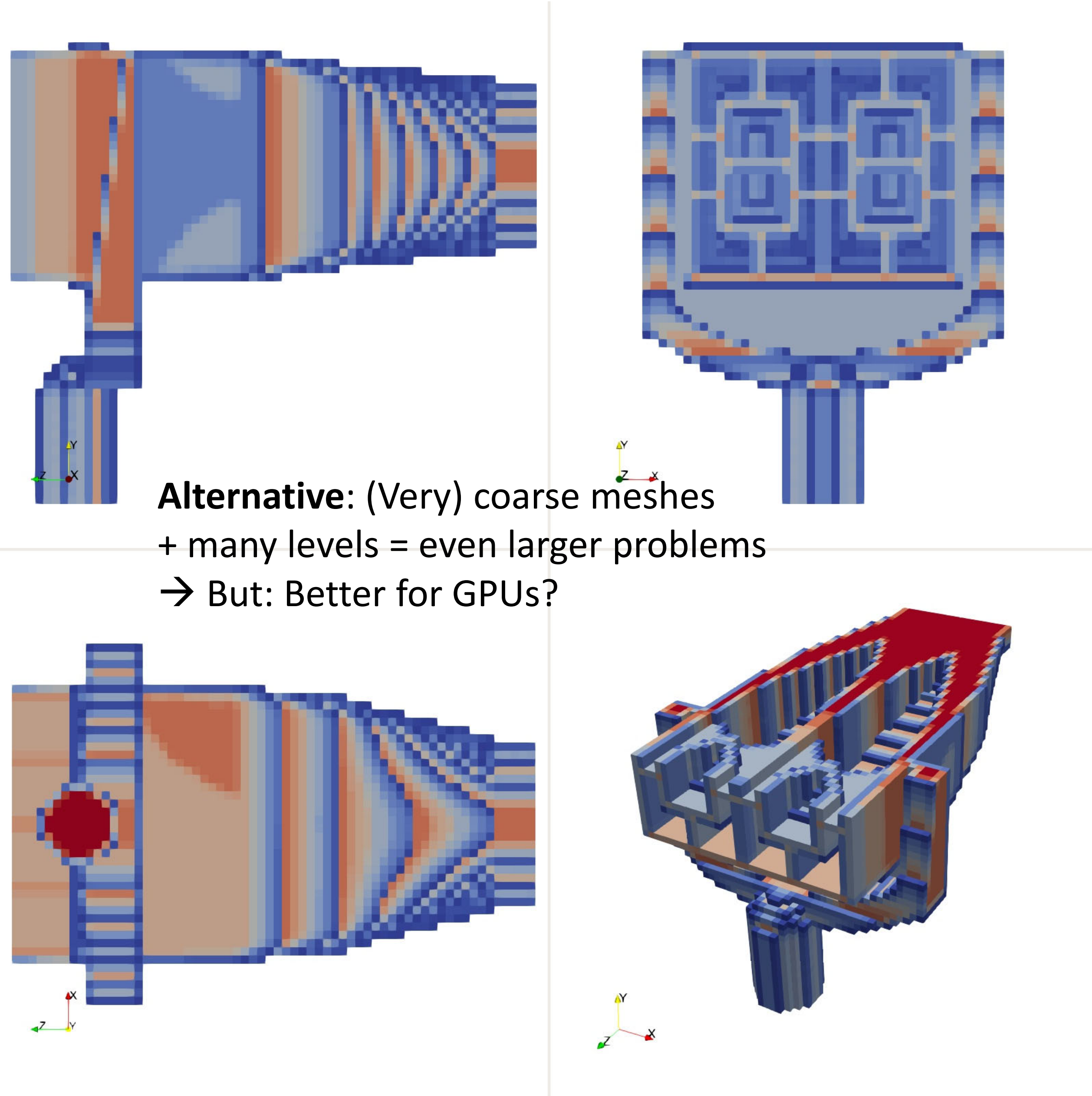




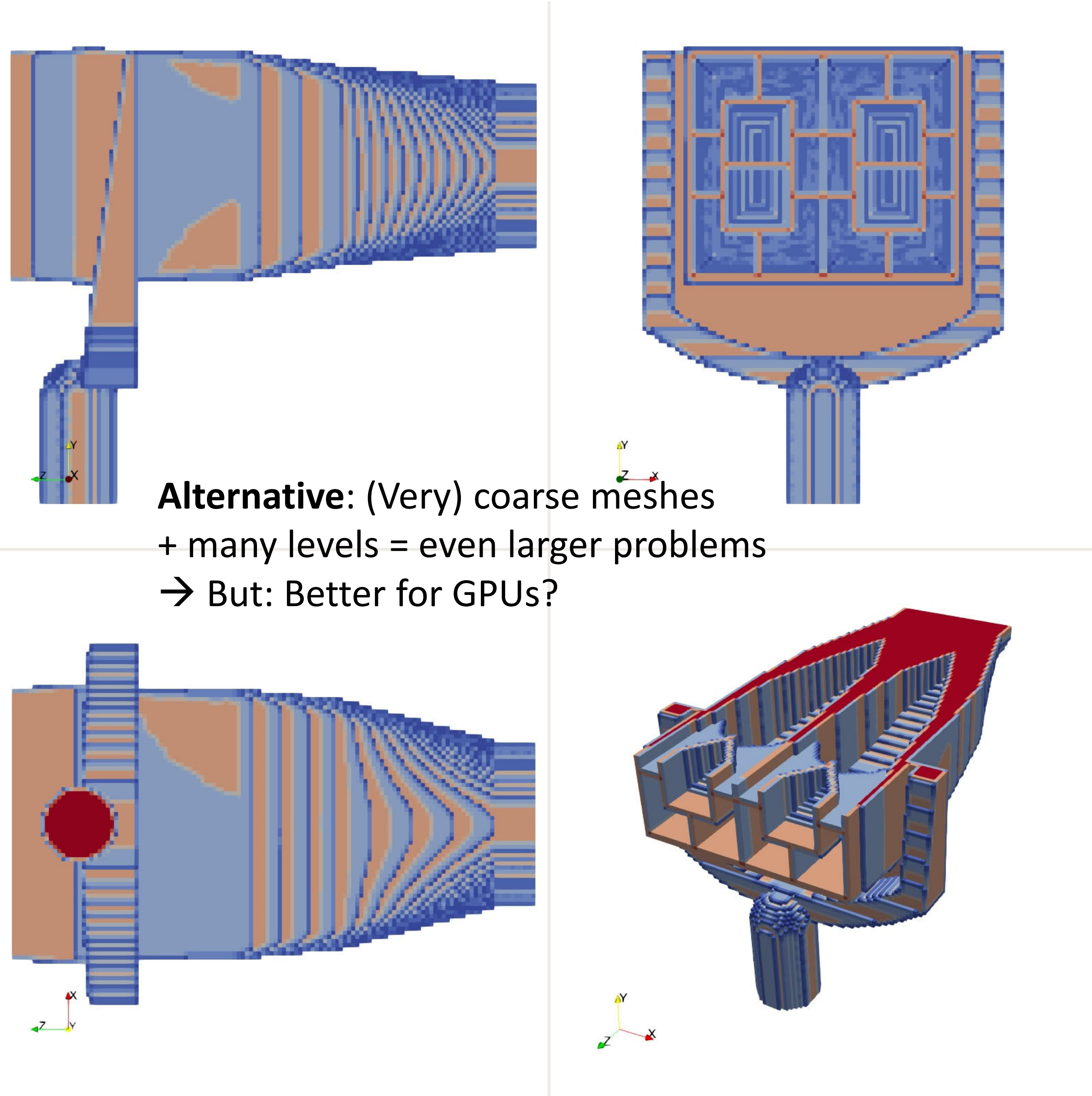
Alternative: (Very) coarse meshes
+ many levels = even larger problems
→ But: Better for GPUs?



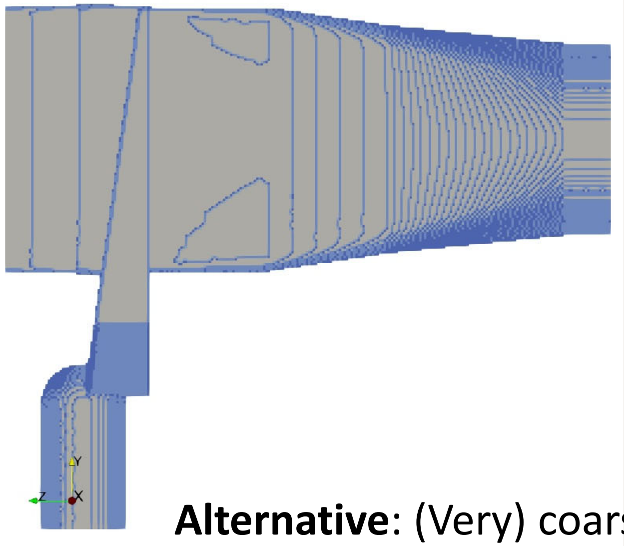


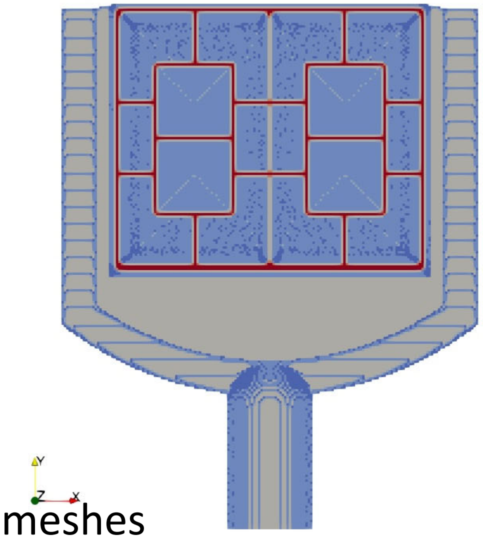


Alternative: (Very) coarse meshes
+ many levels = even larger problems
→ But: Better for GPUs?

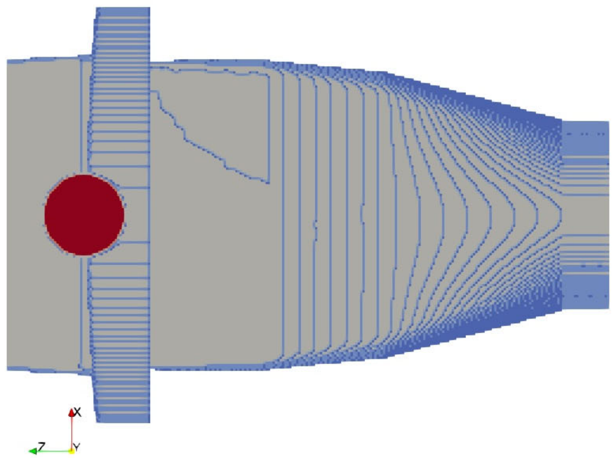


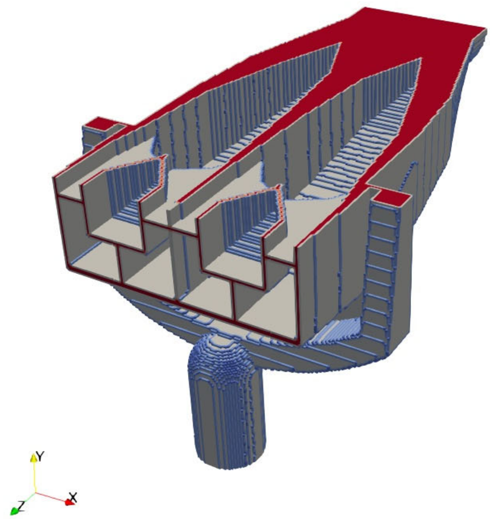
Alternative: (Very) coarse meshes
+ many levels = even larger problems
→ But: Better for GPUs?





Alternative: (Very) coarse meshes
+ many levels = even larger problems
→ But: Better for GPUs?

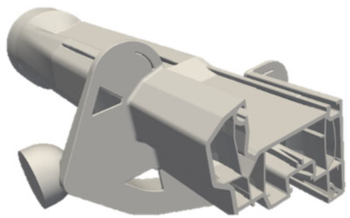


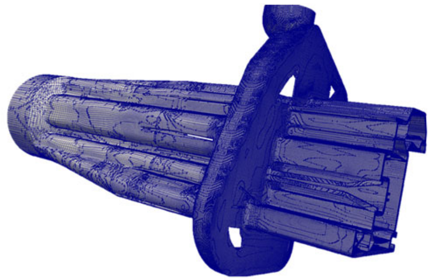


SIMULATIVE DESIGN OPTIMISATION

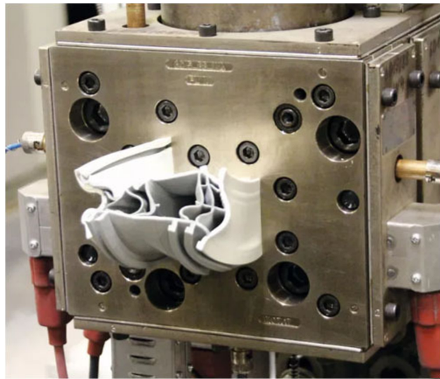
DIGITAL TWIN AND SIMULATION FOR OPTIMAL PROFILE DIE DESIGN

Fluid Domain

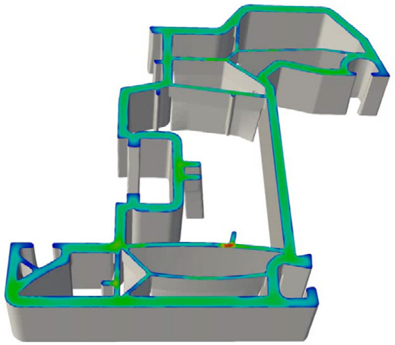




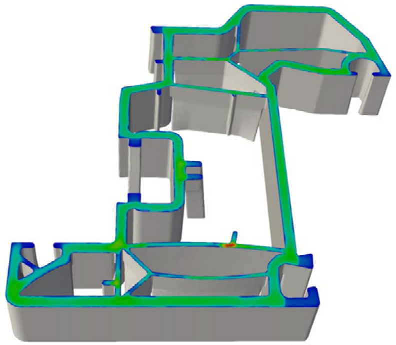
Mesh



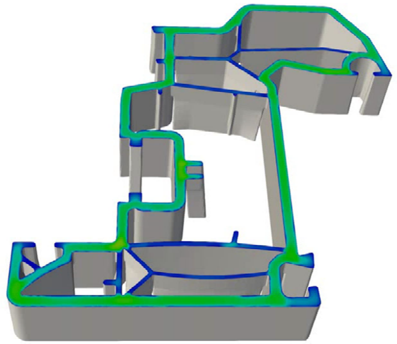
Prototype 1



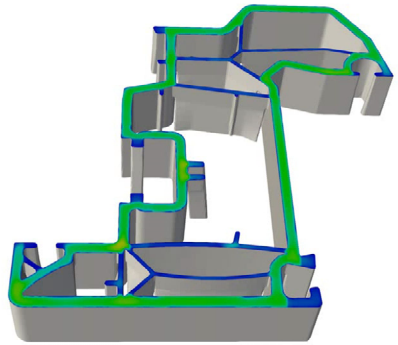
Prototype 2



Prototype 3



Prototype 4



Requires „large scale“ Methods & „large scale“ HPC Hardware

2 trends for HPC Hardware→ TOP500 November 24 (LINPACK)

Rank	System	#Cores	R_{max} (PFlop/s)	Accelerator
1	El Capitan	11,039,616	1,742	AMD MI300A
2	Frontier	9,066,176	1,353	AMD MI250X
3	Aurora	9,264,128	1,012	Intel Ponte Vecchio
4	Eagle	2,073,600	561	NVIDIA H100
5	HPC6	3,143,520	478	AMD MI250X
6	Fugaku	7,630,848	442	(A64FX)
7	Alps	2,121,600	435	NVIDIA GH200
8	LUMI	2,752,704	380	AMD MI250X
9	Leonardo	1,824,768	241	NVIDIA A100
10	Tuolumne	1,161,216	208	AMD MI300A

Exploiting massive parallelism: more than 1 million cores

Exploiting single node performance: special accelerators (NVIDIA, AMD)

→ HPC compute power = #nodes x #TFLOP/s per node

Problem 1: More cores → (Spatially) discretized problems „too small“

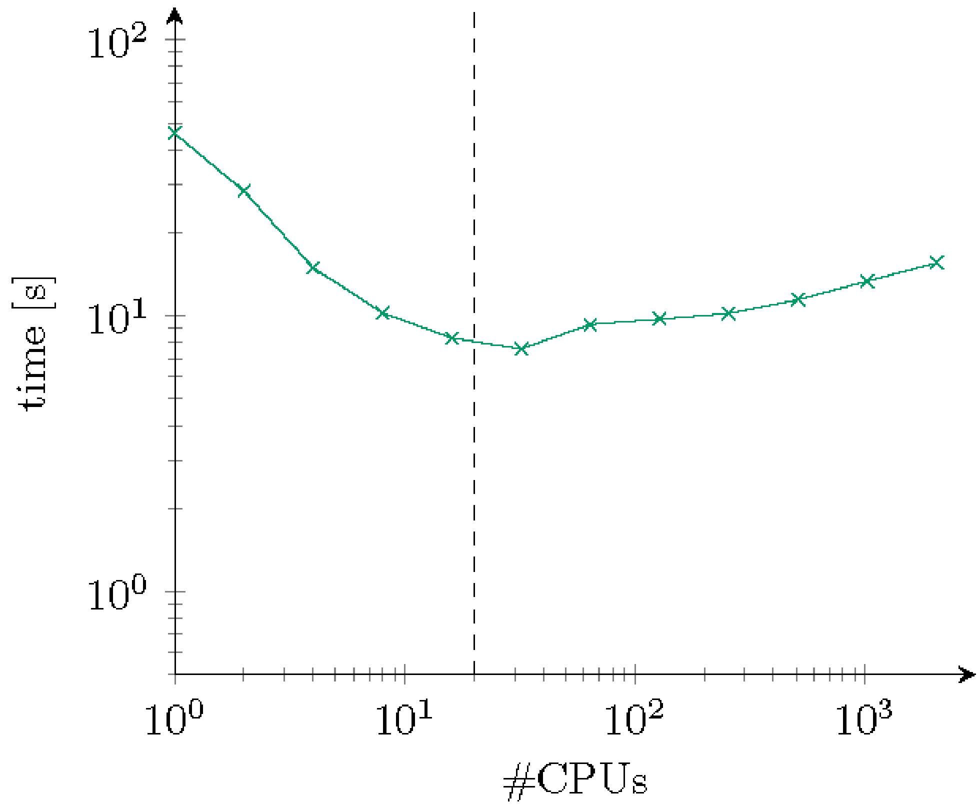
→ Example: Heat equation with multigrid

LiDO3 (2x Intel Xeon E5-2640v4 and 64GB memory per node, Infiniband QDR interconnect (40Gbps))

4 BiCGSTAB pre- and post-smoothing steps, V-cycle

level 5 in space, 2048 total time steps

→ more and more cores (#CPUs) for parallelization "only" in space do not help!



Solver time per iteration

<u>Problem 1: More cores</u> → (Spatially) discretized problems „too small“

→ **Example: Heat equation with multigrid**

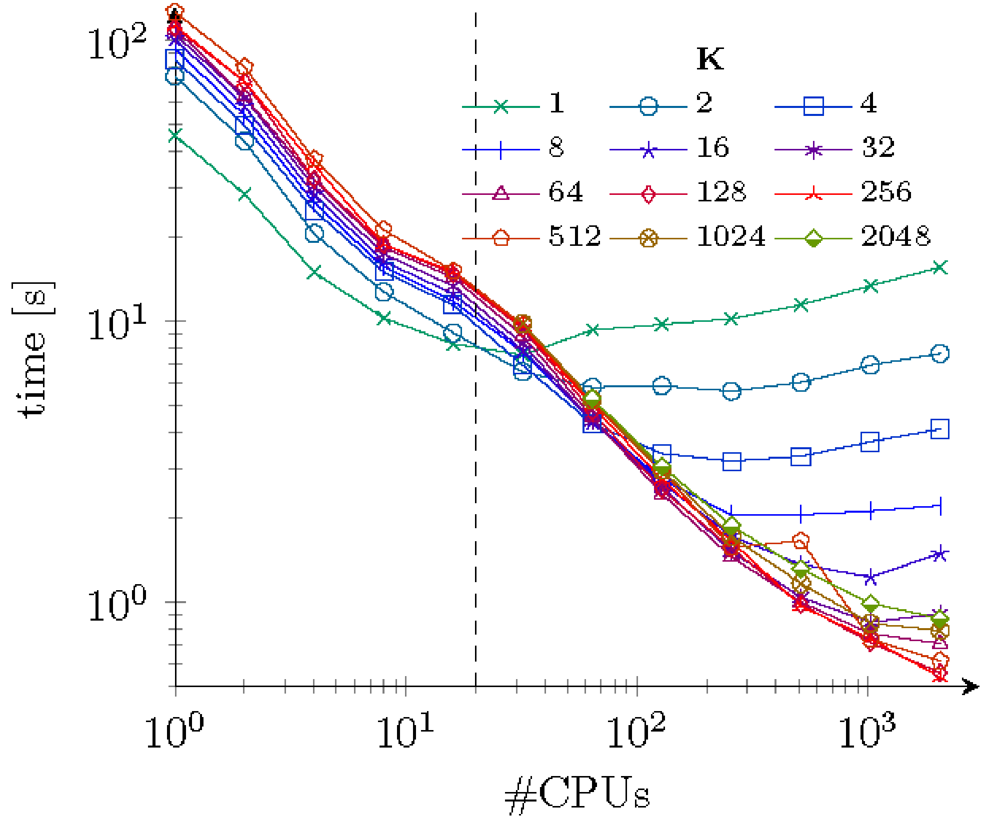
LiDO3 (2x Intel Xeon E5-2640v4 and 64GB memory per node, Infiniband QDR interconnect (40Gbps))

4 BiCGSTAB pre- and post-smoothing steps, V-cycle

level 5 in space, 2048 total time steps

→ **more and more cores (#CPUs) for parallelization “only” in space do not help!**

→ **Better scaling (for K blocked time steps) via Parallel/Simultaneous-in-Time Krylov-Multigrid**



Solver time per iteration

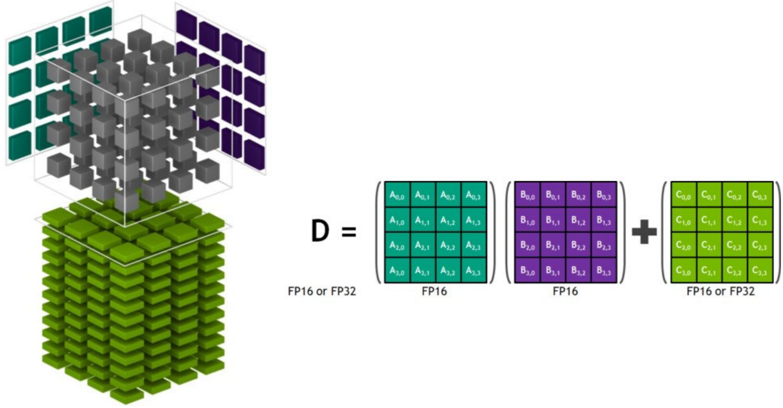
Trends for Accelerator Hardware → NVIDIA, AMD, Amazon

→ **Tensor Cores (TC) by NVIDIA**

- Originally developed to **accelerate AI applications**
- Perform (**dense**) matrix operations at very high speed

→ V100 (2017), A100 (2020), **H100** (2023), **B200** (2024)

- Alternatives: A64FX ARM, AMD MI250X, Trainium2



	FP64	FP64 TC	FP32	TF32	FP16	FP16 TC
V100	7.8	-	15.7	-	**31.4**	**125**
A100	**9.7**	**19.5**	**19.5**	**156**	**78**	**312**
H100	**34**	**67**	**67**	**495**	n/a	**990**
B200	**40**	-	**80**	**2,200**	n/a	**4,500**
MI300A	**61**	-	**122**	**490**	n/a	**981**
A64FX	3.4	-	6.8	-	13.5	-
Trainium2	-	-	181	667	667	-

(Hundreds of) TFlop/s peak rates (in FP32/TF32) → is it realistic.....for PDEs?

Metric for single node performance for PDEs?

Consider (optimal) **geometrical multigrid** for Poisson problems with appr. 10^9 **grid points** → „1000 FLOPs per grid point“

→ Full performance of **100 TF/s**: 0.01s on 1(!) node
→ If only 1% available = **1 TF/s**: 1s on 1(!) node

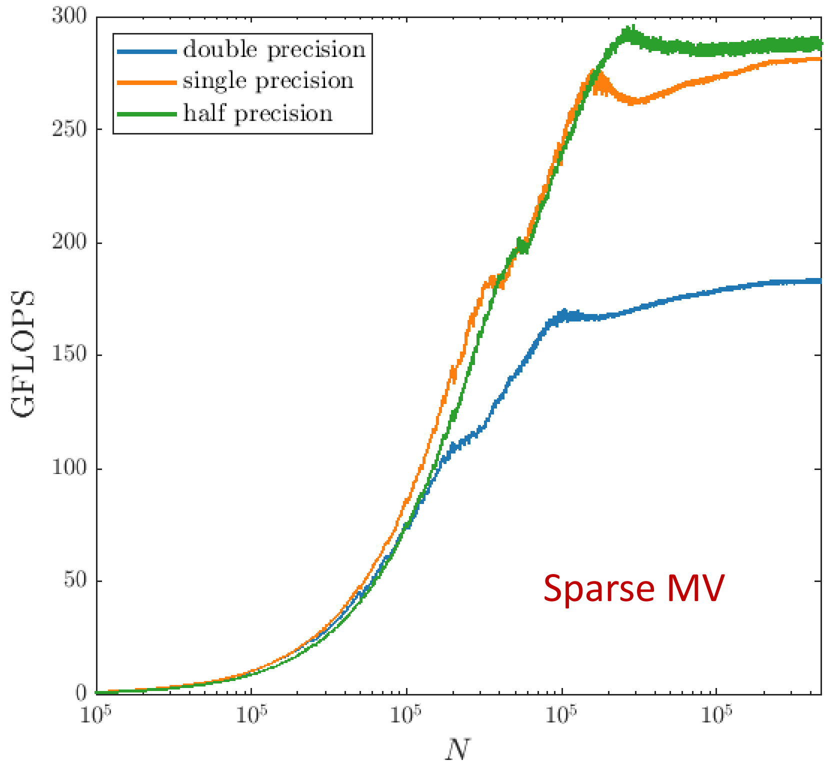
Or: „Solution speed“ **1000 MDOF/s**

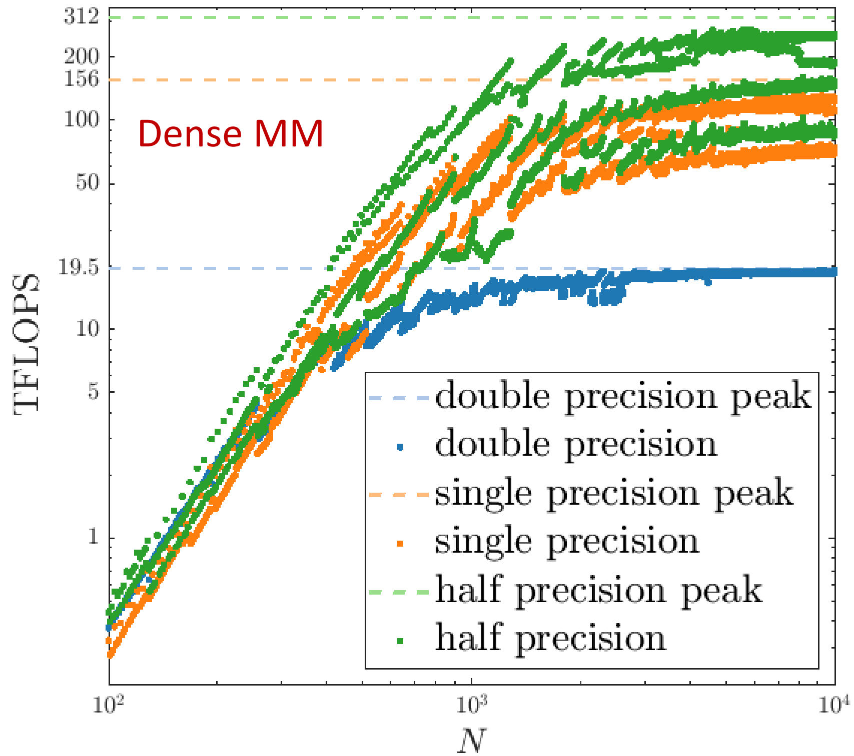
Let‘s be self-critical: How far are we away from exploiting the high single-node performance for such („optimal“) fast solvers???

SPECIFICATIONS


	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS
GPU Memory	32GB /16GB HBM2	
Memory Bandwidth	900GB/sec	
ECC	Yes	
Interconnect Bandwidth	32GB/sec	300GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

Problem 2: Sparse & multigrid vs. dense matrix operations (GEMM)





Only 300 GFLOP/s for sparse MV vs. dense MM (300 TFLOP/s) on A100

Only 10-20 MDOF/s on recent architectures (in FEAT3) with gMG

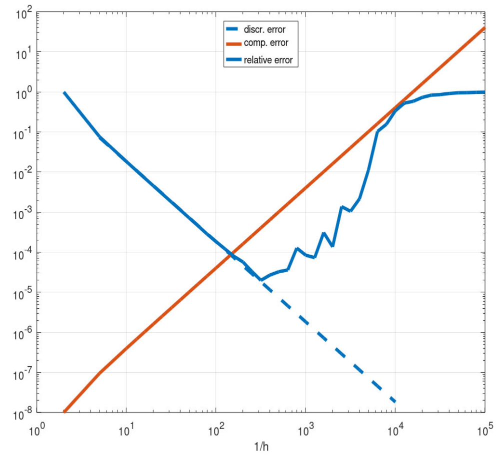
2 Trends for HPC Hardware→ TOP500 November 24 (HPCG)

Rank HPCG (HPL)	System	R_{max} (PFlop/s)	HPCG/HPL
1 (6)	Fugaku	16.0	3.6%
2 (2)	Frontier	14,1	1.0%
3 (3)	Aurora	5,6	0.6%
4 (8)	LUMI	4,6	1.2%
5 (7)	Alps	3,7	0.8%
6 (9)	Leonardo	3,1	1.3%
7 (19)	Perlmutter	1,9	2.4%
8 (14)	Sierra	1,8	1.9%
9 (23)	Selene	1,6	2.6%
10 (33)	JUWELS Booster Module	1,3	2.9%

→ Iterative sparse solvers: only appr. 1-4% of the available peak rates

<u>Problem 3a:</u> Lower precision hardware for Poisson problems?

→ **Split the error:** $u - \tilde{u}_h = u - u_h + u_h - \tilde{u}_h$



Discr. Error: $\|\boldsymbol{u} - \boldsymbol{u_h}\| = \mathcal{O}(\boldsymbol{h}^{p+1})$

→ depending on **FEM space** and **smoothness**

→ Here for simplicity: $\boldsymbol{p} = \mathbf{1}$

Comp. Error: $\|\tilde{\boldsymbol{u}}_h - \boldsymbol{u_h}\| \approx \boldsymbol{cond_h} \cdot$ "data error"

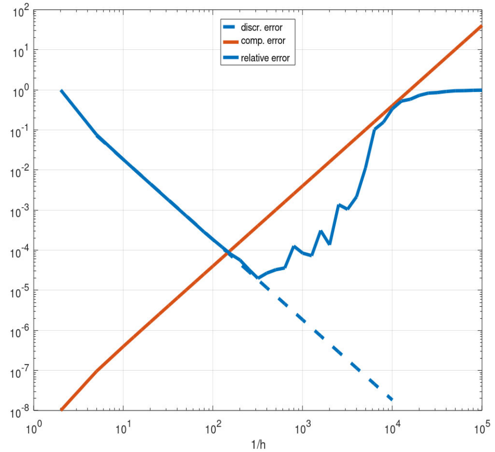
→ data error at least of size $\mathrm{TOL_{Prec}}$

→ $\boldsymbol{cond_h}(\mathbf{Poisson}) = \mathcal{O}(h^{-2})$

Discr. Error ≈ Comp. Error ⇒ $\boldsymbol{h} \approx \mathbf{TOL_{Prec}}^{1/4}$

Problem 3a: Lower precision hardware for Poisson problems?

→ **Split the error:** $u - \tilde{u}_h = u - u_h + u_h - \tilde{u}_h$



Discr. Error: $\|u - u_h\| = \mathcal{O}(h^{p+1})$

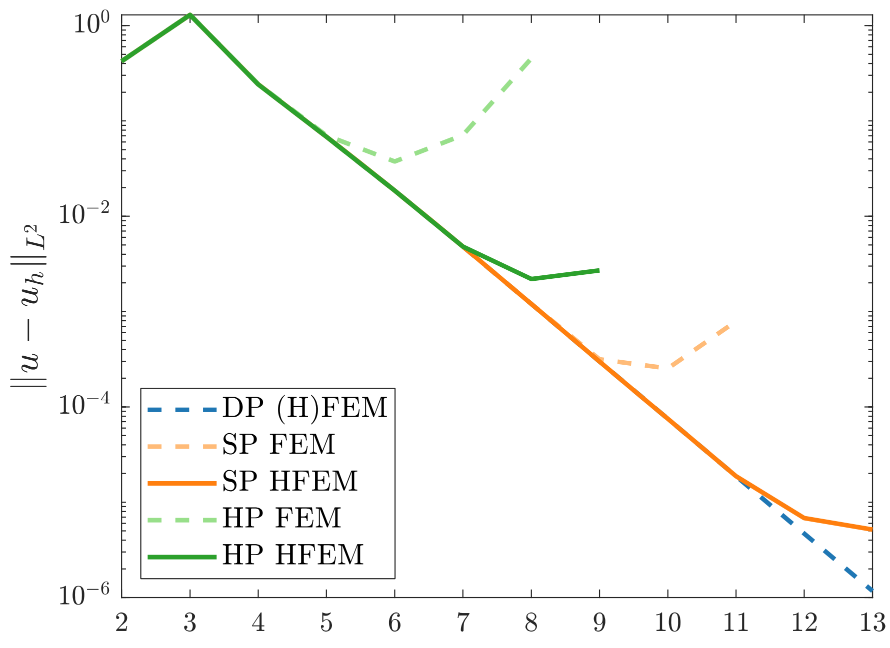
→ depending on **FEM space** and **smoothness**

→ Here for simplicity: $p = 1$

Comp. Error: $\|\tilde{u}_h - u_h\| \approx cond_h \cdot$ "data error"

→ data error at least of size $\mathrm{TOL}_{\mathrm{Prec}}$

→ $cond_h(\text{Poisson}) = \mathcal{O}(h^{-2})$



Discr. Error ≈ Comp. Error $\Rightarrow h \approx \mathrm{TOL}_{\mathrm{Prec}}{}^{1/4}$

Wish: $cond_h = \mathcal{O}(1) \quad \Rightarrow h \approx \mathrm{TOL}_{\mathrm{Prec}}{}^{1/2}$

→ **FP32/TF32 (and even FP16?)**

Preliminary summary regarding recent hardware trends

→ **Parallelization in space** is not enough….particularly for **very many time steps**

Global-in-time approaches* using *parallel-in-time,* resp., *simultaneous-in-time Krylov-multigrid solvers

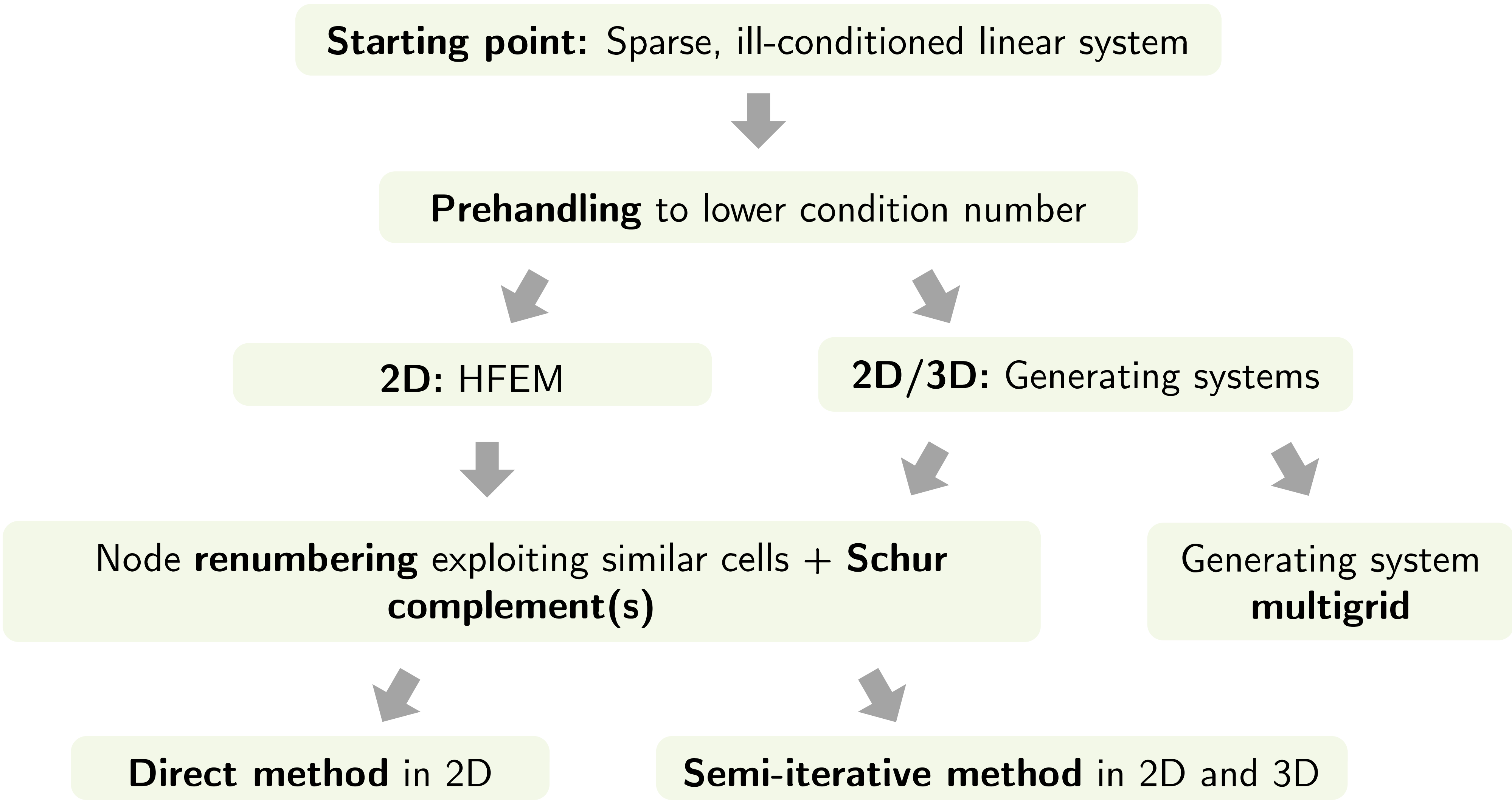
Summary regarding recent hardware trends

→ **Parallelization in space** is not enough….particularly for **very many time steps**

Global-in-time approaches* using *parallel-in-time, resp., simultaneous-in-time Krylov-multigrid solvers

→ Standard **sparse** matrix-vector operations (in **FP64)** quite **„slow“** on **GPUs**
→ **But: Lower precision** (**FP32, TF32, FP16**) often **not sufficiently accurate**
→ **But:** They do **not exploit** the **Tensor Cores!**

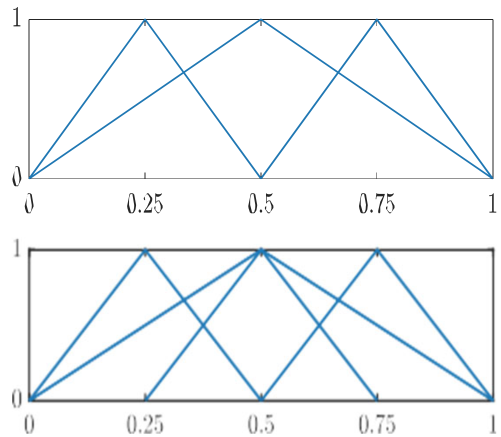
Lower precision & dense matrix operations on GPUs (with Tensor Cores) via Prehandling and special Schur Complement solvers with application to CFD


Starting point: Sparse, ill-conditioned linear system
Prehandling to lower condition number
2D: HFEM
2D/3D: Generating systems
Node **renumbering** exploiting similar cells + **Schur complement(s)**
Generating system **multigrid**
Direct method in 2D
Semi-iterative method in 2D and 3D

The concept of Prehandling for linear systems of equations

Basic idea

- Apply preconditioner **explicitly** to $Ax = b$
- Equivalent system $\tilde{A}\tilde{x} = \tilde{b}$, where $S^T AS$, $b^T = S^T b$, $x = S\tilde{x}$
- Both yield **same solution** in exact arithmetic, but accuracy (and iteration numbers) differ in practice because $\mathbf{cond}(\boldsymbol{A}) \neq \mathbf{cond}(\widetilde{\boldsymbol{A}})$



Central requirements for Prehandling

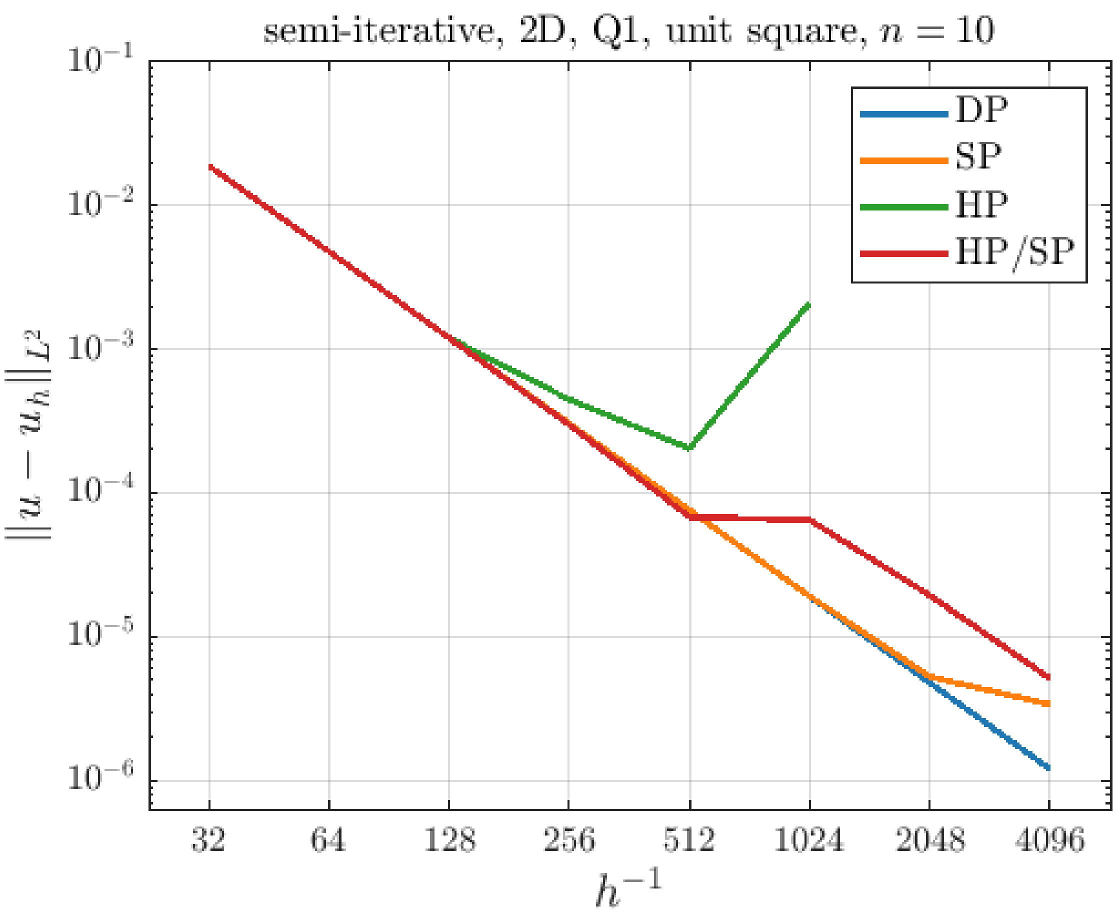
- $\mathbf{cond}(\widetilde{\boldsymbol{A}}) \ll \mathbf{cond}(\boldsymbol{A})$
- $\widetilde{\boldsymbol{A}}$ **is still sparse**
- **Transformation to** $\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{b}}$ **and** $\boldsymbol{x}$ **via** $\boldsymbol{S}$ **is fast (i.e.,** $\boldsymbol{\mathcal{O}}(\boldsymbol{N}\mathbf{log}\boldsymbol{N})$**)**

So far two candidates fulfill all requirements:

Hierarchical Finite Element Method (HFEM, Yserentant et al., 1980s) in 2D and **Generating Systems** (GS, Griebel et al., 1990s) in 2D and 3D

Prehandling via HFEM or Generating Systems

Example: (Typical) L_2 errors for Poisson problem for different levels in 2D



→ **same FEM solution on lower precision hardware is possible** (in the range of 1% error as typical for highly complex applications)

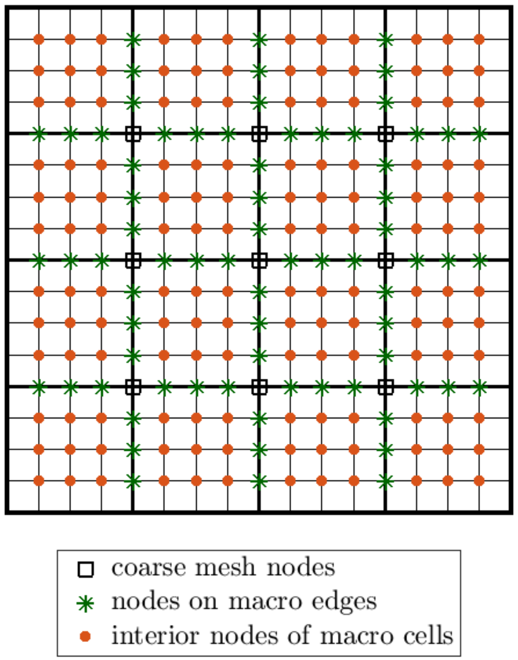
Next: How to exploit **Tensor Cores** via **sparse-dense Matrix** operations?

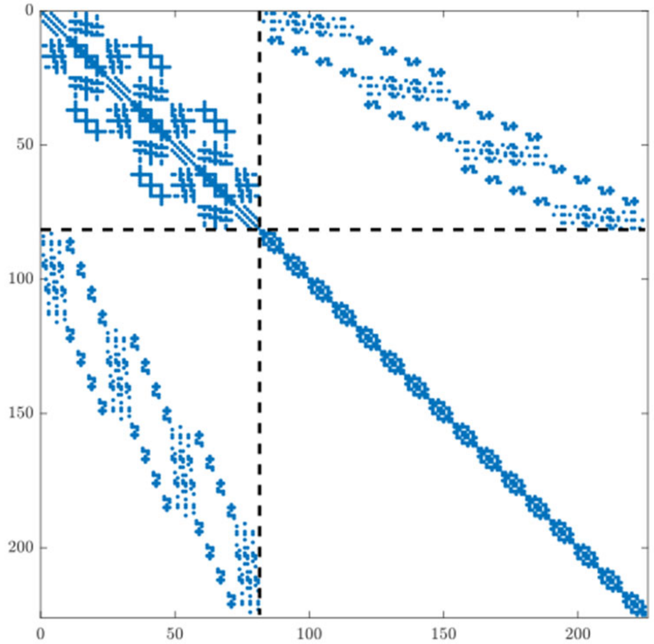
Schur Complement (SC) solvers taylored for Tensor Core GPUs

- Construct solver consisting as much as possible on **multiplications with dense matrices**
- Same principle in 2D (**HFEM**) and 3D (**GS**): **Subdivide nodes due to macro size h_0 into**

  a) nodes in the interior of the coarse mesh cells (cell by cell in same order)

  b) "all remaining nodes" containing those on coarse mesh edges (+ repeated nodes of GS)





- Matrix form: $\begin{pmatrix} A_1 & B \\ B^T & C \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$
- C decomposes into independent blocks C_i
- Blocks are equal if corres-ponding to similar cells
- Only C grows like N (= #Dofs)

Schur Complement (SC) solvers taylored for Tensor Core GPUs

- Applying Schur Complement to $\begin{pmatrix} A_1 & B \\ B^T & C \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ yields

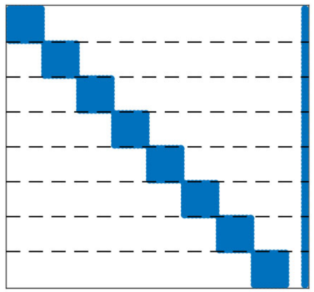
Semi-iterative Method

1) Solve $\hat{A}u = b_1 - BC^{-1}b_2$, where $\hat{A} = A_1 - BC^{-1}B^T$ with CG method
2) $v = C^{-1}(b_2 - B^T u)$

- $\hat{A}$ can be computed **explicitly in 2D** (then **direct** method) or used **implicitly** with **iterative** CG (better option in **3D**) & $\widehat{A}$ **well-conditioned**
- C_i are small, well-conditioned HFEM matrices

→ $\mathcal{O}(N)$ storage for ${C_i}^{-1}$

→ C^{-1} block diagonal matrix with dense blocks ${C_i}^{-1}$



Storage requirement of the semi-iterative method

- **Test problem: Poisson** equation on unit square/cube, equidistant Q1 mesh, variable coarse mesh size h_0
- Relevant for storage: C_i^{-1} , B and $\hat{A}$ in 2D / A_1 in 3D

2D: HFEM

$\frac{1}{h}$	$\frac{N}{10^6}$	$\frac{1}{h_0}$	$\hat{A}$	C_i^{-1}	B	total
1024	1.05	16	15	15.1	1.0	**31**
		32	25	0.9	1.6	**27**
2048	4.19	32	19	3.8	1.0	**24**
		64	40	0.2	1.6	**42**
4096	16.77	32	16	15.5	0.7	**32**
		64	27	0.9	1.0	**29**

3D: Generating Systems

$\frac{1}{h}$	$\frac{N}{10^6}$	$\frac{1}{h_0}$	A_1	C_i^{-1}	B	total
		4	11.3	433.3	15.4	**460**
128	2.05	8	22,1	5.6	16.6	**44**
		16	37.1	0.1	15.3	**52**
		8	14.2	53.5	16.5	**84**
256	16.58	16	24.9	0.7	17.7	**43**
		32	39.5	0.01	16.4	**56**

Number of nonzero entries relative to N

Moderate storage requirement for appropriate choice of h_0 compared to **9N** in 2D / **27N** in 3D with standard FEM (in **FP64**)

Performance estimate (in FP32/TF32 on A100)

2D:

$\frac{1}{h}$	$\frac{1}{h_0}$	#iter	cond(C_i)	total $\frac{\text{Flop}}{N}$	share dense	GFlop/s	MDof/s
1024	16	30	24	16,400	94.4%	27,400	**1,670**
	32	24	17	4,900	75.4%	6,700	1,360
2048	32	28	24	16,600	93.5%	21,600	**1,300**
	64	23	17	5,600	66.4%	4,100	730
4096	32	31	32	64,700	98.4%	58,700	910
	64	25	24	16,900	91.9%	15,600	**920**

3D:

$\frac{1}{h}$	$\frac{1}{h_0}$	#iter	cond(C_i)	total $\frac{\text{Flop}}{N}$	share dense	GFlop/s	MDof/s
128	4	8	54	555,300	99.9%	110,400	200
	8	11	23	75,400	98.3%	50,800	**670**
	16	18	9	12,500	79.3%	6,500	520
256	8	11	54	713,700	99.8%	107,500	150
	16	18	23	114,900	98.0%	47,400	**410**
	32	35	9	23,400	77.3%	6,100	260

Compare with results with optimized MG in C++-based FEM software package (FEAT) on AMD CPU in FP64: 10-20 MDOF/s

Results on A100 vs. H100

Mdof/s results on H100 (≈ 3×peak rates of A100 in **SP/TF32** with TC)

2D

$\frac{1}{h}$	$\frac{1}{h_0}$	A100	H100
1024	16	1,670	2,860
	32	1,360	2,430
2048	32	1,300	2,220
	64	730	1,180
4096	32	910	2,020
	64	920	1,540

3D

$\frac{1}{h}$	$\frac{1}{h_0}$	A100	H100
128	4	200	480
	8	670	1,160
	16	520	840
256	8	150	440
	16	410	680
	32	260	400

Typical "Hardware-oriented Numerics" approach: "optimal" configuration depends on problem(size) and hardware

[1]Kindly provided for use on JURECA by Forschungszentrum Jülich https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/jureca

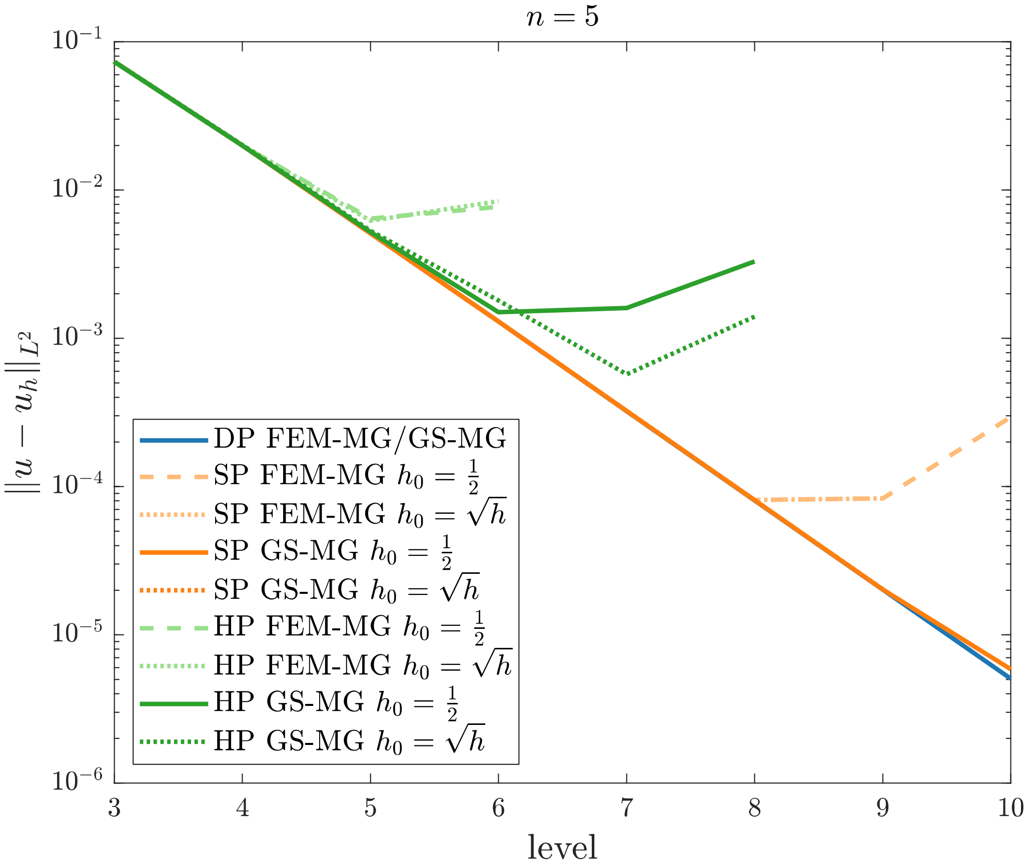
A multigrid method based on generating systems

- **Aim:** Combine advantages of GS (enabling lower precision) and multigrid (convergence properties & flexibility via different smoothers, cycles, …)

Basic idea

- Construct multigrid-like algorithm to solve system w.r.t. GS: $Ex^E = b^E$ ($x = Sx^E$)
- Level-wise structure (3-level example): $E = \begin{pmatrix} A_0 & (P_0^2)^T A P_1^2 & (P_0^2)^T A \\ (P_1^2)^T A P_0^2 & A_1 & (P_1^2)^T A \\ A P_0^2 & A P_1^2 & A \end{pmatrix}$
- In each iteration: Go through levels (block rows of E) according to V-, W- or F-cycle and
  - Compute residual r_i of current block row i
  - If $i = 0$: coarse grid solver, else apply ν smoothing steps w.r.t. A_i and r_i
- Equivalent to standard multigrid but underlying system given by GS matrix E
- Open question: How to modify algorithms in order to exploit Tensor Cores?

A multigrid method based on generating systems



Comparison of errors FEM-MG vs. GS-MG for Poisson's equation in 2D with smooth exact solution (V-cycle, Gauß-Seidel preconditioned Richardson smoother with 4 smoothing steps)

h	h_0	V-cycle				F-cycle			
		$\nu = 2$	$\nu = 4$	$\nu = 8$	$\nu = 16$	$\nu = 2$	$\nu = 4$	$\nu = 8$	$\nu = 16$
$\frac{1}{64}$	$1/2 - 1/16$	5	4	3	3	4	4	3	3
	$1/32$	4	4	3	3	4	4	3	3
$\frac{1}{256}$	$1/2 - 1/16$	5	4	3	3	4	3	3	3
	$1/32$	4	4	3	3	4	3	3	3
	$1/64 - 1/128$	4	3	3	3	4	3	3	3
$\frac{1}{1024}$	$1/2 - 1/16$	5	4	4	3	3	3	3	2
	$1/32$	5	4	3	3	3	3	3	2
	$1/64$	4	4	3	3	3	3	3	2
	$1/128 - 1/256$	4	3	3	3	3	3	3	2
	$1/512$	3	3	3	2	3	3	3	2
$\frac{1}{4096}$	$1/2 - 1/16$	5	4	4	3	3	3	2	2
	$1/32$	5	4	3	3	3	3	2	2
	$1/64$	4	4	3	3	3	3	2	2
	$1/128 - 1/512$	4	3	3	3	3	3	2	2
	$1/1024$	4	3	3	2	3	3	2	2
	$1/2048$	3	3	2	2	3	3	2	2

Iteration numbers of GS-MG for Poisson equation on unit square with Gauß-Seidel smoothing steps

Main questions:

How to exploit efficiently („that means with optimal computational and numerical efficiency“) not only Massively Parallel, but also Lower Precision Accelerator hardware for PDE problems?

Important components:

→ Prehandling & semi-iterative sparse-dense solvers in lower precision
→ Global-in-time Newton & Oseen (= linearized NSE) solvers
→ Parallel-in-time / Simultaneous-in-time Multigrid approaches

Sequential time-stepping:

$$\begin{pmatrix} \mathrm{A}_i & \mathrm{B} \\ \mathrm{B}^\top & \end{pmatrix} \begin{pmatrix} \mathrm{u}^{(n+1)} \\ \tilde{\mathrm{p}}^{(n+1)} \end{pmatrix} = \begin{pmatrix} \tilde{\mathrm{g}}^{(n+1)} - \mathrm{A}_e \mathrm{u}^{(n)} \\ \mathrm{f}^{(n+1)} \end{pmatrix}, \qquad n = 0, \ldots, K$$

Treating **K time steps simultaneously**:

$$\begin{pmatrix} \mathbf{A}_K & \mathbf{B}_K \\ \mathbf{B}_K^\top & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \tilde{\mathbf{p}} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{g}} \\ \mathbf{f} \end{pmatrix}$$

$$:\Longleftrightarrow \left(\begin{array}{cccc|cccc} \mathrm{A}_i & & & & \mathrm{B} & & & \\ \mathrm{A}_e & \mathrm{A}_i & & & & \mathrm{B} & & \\ & \ddots & \ddots & & & & \ddots & \\ & & \mathrm{A}_e & \mathrm{A}_i & & & & \mathrm{B} \\ \hline \mathrm{B}^\top & & & & & & & \\ & \mathrm{B}^\top & & & & & & \\ & & \ddots & & & & & \\ & & & \mathrm{B}^\top & & & & \end{array}\right) \left(\begin{array}{c} \mathrm{u}^{(1)} \\ \mathrm{u}^{(2)} \\ \vdots \\ \mathrm{u}^{(K)} \\ \hline \tilde{\mathrm{p}}^{(1)} \\ \tilde{\mathrm{p}}^{(2)} \\ \vdots \\ \tilde{\mathrm{p}}^{(K)} \end{array}\right) = \left(\begin{array}{c} \tilde{\mathrm{g}}^{(1)} - \mathrm{A}_e \mathrm{u}^{(0)} \\ \tilde{\mathrm{g}}^{(2)} \\ \vdots \\ \tilde{\mathrm{g}}^{(K)} \\ \hline \mathrm{f}^{(1)} \\ \mathrm{f}^{(2)} \\ \vdots \\ \mathrm{f}^{(K)} \end{array}\right)$$

Now: Apply **Pressure Schur Complement (PSC) techniques**

$$\begin{pmatrix} \mathbf{A}_K & \mathbf{B}_K \\ \mathbf{B}_K^\top & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \tilde{\mathbf{p}} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{g}} \\ \mathbf{f} \end{pmatrix}$$

$$\boxed{\mathbf{B}_K^\top \mathbf{A}_K^{-1} \mathbf{B}_K \tilde{\mathbf{p}} = \mathbf{B}_K^\top \mathbf{A}_K^{-1} \tilde{\mathbf{g}} - \mathbf{f}}$$

$$\mathbf{u} = \mathbf{A}_K^{-1}(\tilde{\mathbf{g}} - \mathbf{B}_K \tilde{\mathbf{p}})$$

Corresponding iterative solver:

$$\tilde{\mathbf{p}} \quad \mapsto \quad \tilde{\mathbf{p}} + \mathbf{q},$$

$$\mathbf{q} = \mathbf{C}_K^{-1}(\mathbf{B}_K^\top \tilde{\mathbf{u}} - \mathbf{f}), \qquad \tilde{\mathbf{u}} = \mathbf{A}_K^{-1}(\tilde{\mathbf{g}} - \mathbf{B}_K \tilde{\mathbf{p}})$$

Using PSC preconditioner $\mathbf{C}_K \approx \mathbf{P}_K = \mathbf{B}_K^\top \mathbf{A}_K^{-1} \mathbf{B}_K$

Preconditioners for Pressure Schur Complement iteration:

	PCD preconditioner[1]	LSC preconditioner
$\mathbf{C}_K^{-1}$	$(\mathrm{I}_K \otimes \mathrm{M}_p^{-1})\mathbf{A}_{K,p}(\mathrm{I}_K \otimes \hat{\mathrm{D}}_p^{-1})$	$\left(\mathrm{I}_K \otimes (\hat{\mathrm{D}}_p^{-1}\mathrm{B}^\top \mathrm{M}_u^{-1})\right)\mathbf{A}_K\left(\mathrm{I}_K \otimes (\mathrm{M}_u^{-1}\mathrm{B}\hat{\mathrm{D}}_p^{-1})\right)$
Effort	$1\times$Poisson & $1\times$mass	$2\times$Poisson & $2\times$mass

- **Parallel-/Simultaneous-in-time Multigrid** solvers for **nonsteady convection-diffusion-reaction** equations

- **Prehandling + Schur Complement Poisson** solvers for K **Poisson** problems, resp., 1 **Poisson** problem with **K right hand sides**

[1] Danieli et al. (2022)
Pressure Poisson matrix $\hat{\mathrm{D}}_p = \mathrm{B}^\top \mathrm{M}_u^{-1}\mathrm{B}$

Global-in-time Pressure Schur Complement Preconditioners

→ **Solve in EACH nonlinear iteration for ALL K time steps SIMULTANEOUSLY**

$$\left[\begin{array}{cccc|cccc|cccc|c}
S^1 & & & & & & & & & & & & u^1 \\
-M_l & S^2 & & & & & & & & & & & u^2 \\
& \ddots & \ddots & & & & 0 & & & & 0 & & \vdots \\
& & -M_l & S^{n+1} & & & & & & & & & u^{n+1} \\
\hline
& & & & S^1 & & & & & & & & v^1 \\
& & & & -M_l & S^2 & & & & & & & v^2 \\
& & 0 & & & \ddots & \ddots & & & & 0 & & \vdots \\
& & & & & & -M_l & S^{n+1} & & & & & v^{n+1} \\
\hline
-\frac{1}{k}B_1^T & & & & -\frac{1}{k}B_2^T & & & & C & & & & p^1 \\
\frac{1}{k}B_1^T & -\frac{1}{k}B_1^T & & & \frac{1}{k}B_2^T & -\frac{1}{k}B_2^T & & & & C & & & p^2 \\
& \ddots & \ddots & & & \ddots & \ddots & & & & \ddots & & \vdots \\
& & \frac{1}{k}B_1^T & -\frac{1}{k}B_1^T & & & \frac{1}{k}B_2^T & -\frac{1}{k}B_2^T & & & & C & p^{n+1}
\end{array}\right]$$

- **nonstationary convection-diffusion-reaction** equations for velocities

⇒ inner **Parallel-/Simultaneous-in-time Multigrid** solvers

- **K** x **Pressure-Poisson-like problems** $Cp^i = rhs^i$ with **many right hand sides**

⇒ **Prehandling + Schur Complement Poisson solver**

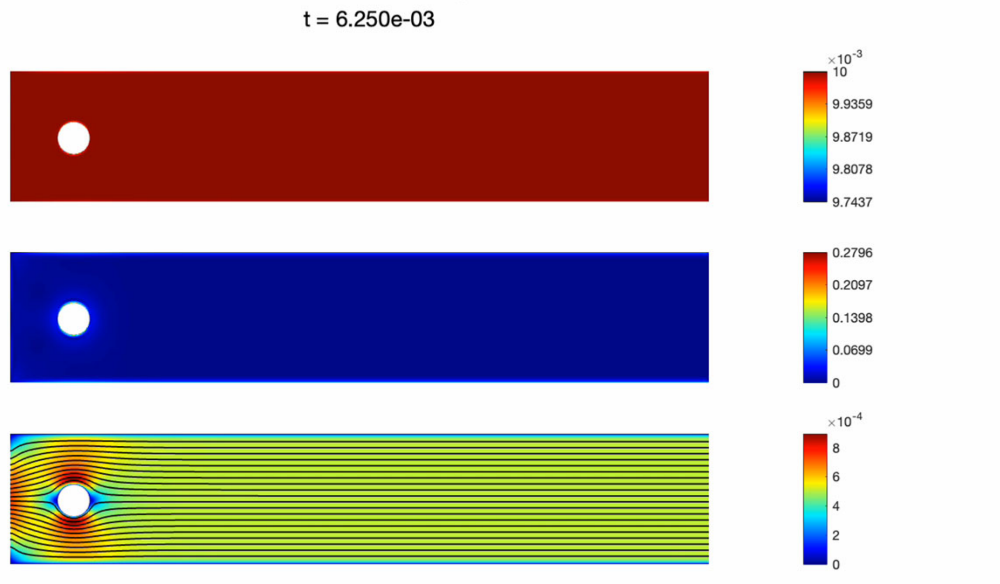
(Almost) Final component: Global-in-time (Picard-)Newton solver

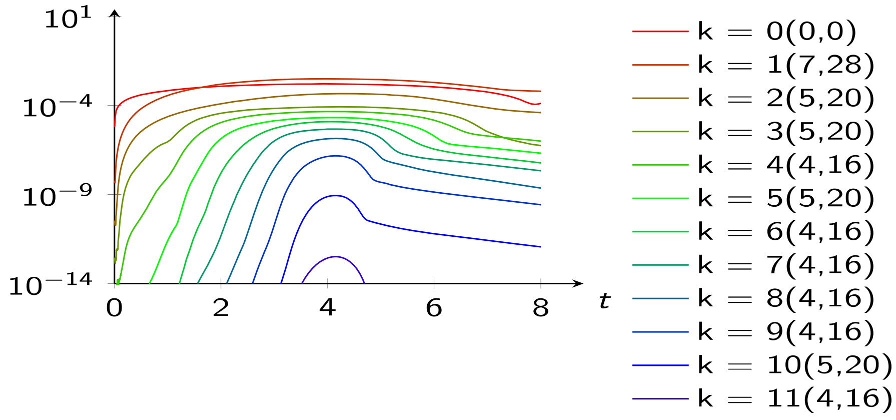
- Usual approach: apply a time discretization and solve a **nonlinear equation** in **each time step** with **solution** from **last time step** on right hand side
- **Problem: no solution for last time step available since global-in-time**

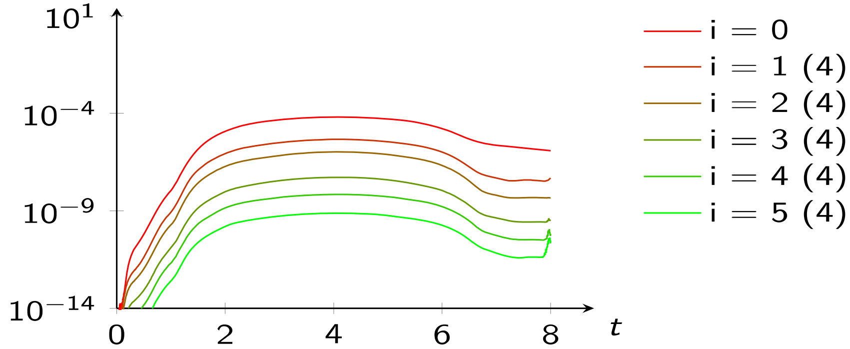
  - solve the nonlinear equation on the **complete space-time** domain
  - use a **global-in-time** (g-i-t) **Picard-Newton** method
  - **Jacobian** matrices correspond to **nonstationary Oseen** equations

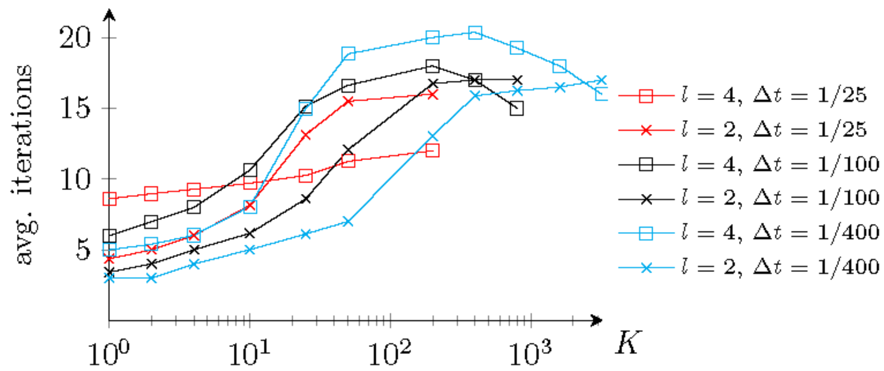
Idea: Quadratic convergence may lead to K-independent results

Proof of Concept: „FAC“ Benchmarks: BENCH3-Carreau Modell





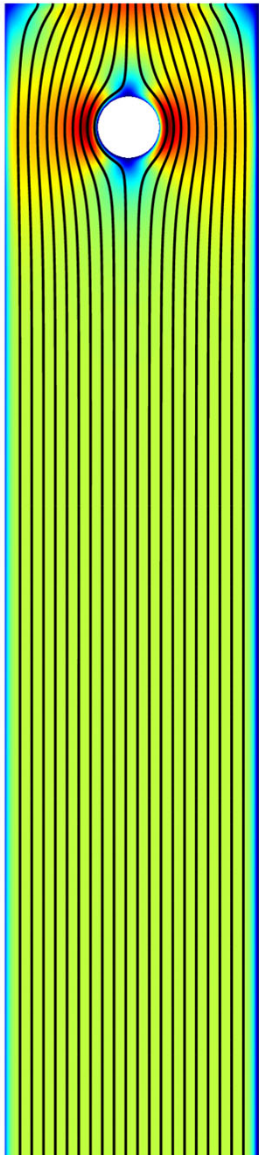


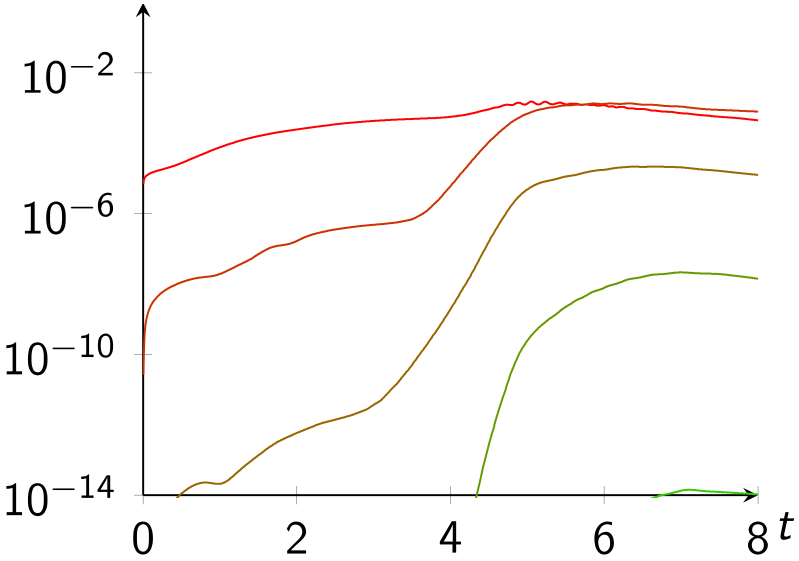


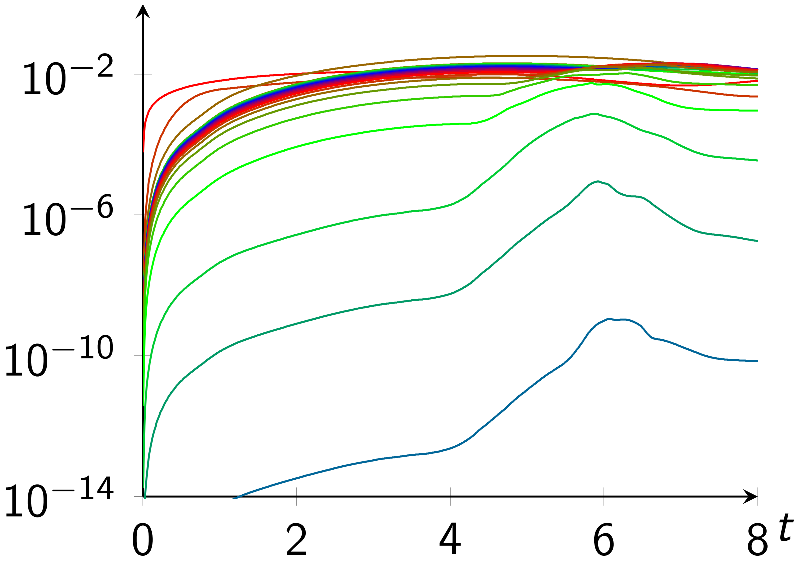
- → **11 g-i-t Newton steps** (3200 time steps with dt=1/400)
- → **4-7 g-i-t Oseen PSC steps** per Newton step
- → **4-20 SinT-MG steps** for convection-diffusion-reaction problems with **space-time dependent viscosity**

“No problem” for time-dependent & high viscosity problems with complex rheology (“polymer extrusion”) to apply g-i-t CFD solver

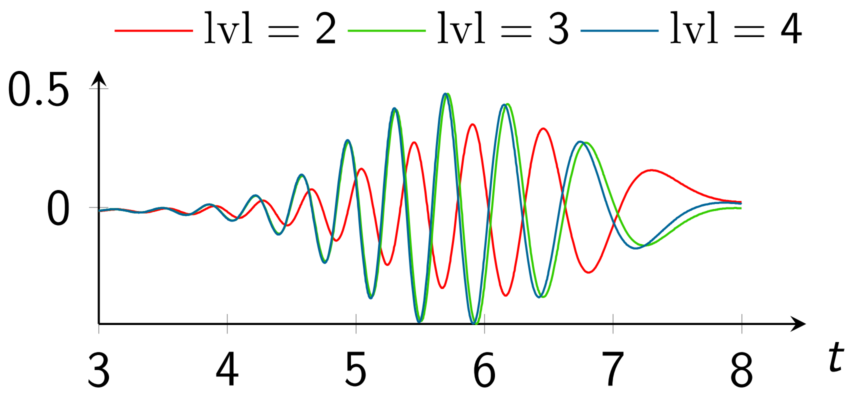
Proof of Concept: „FAC“ Benchmarks: BENCH3 – g-i-t Newton solver







→ **4 g-i-t undamped Newton iterations** (2048 time steps with dt=1/256) on level 4 started from level 3

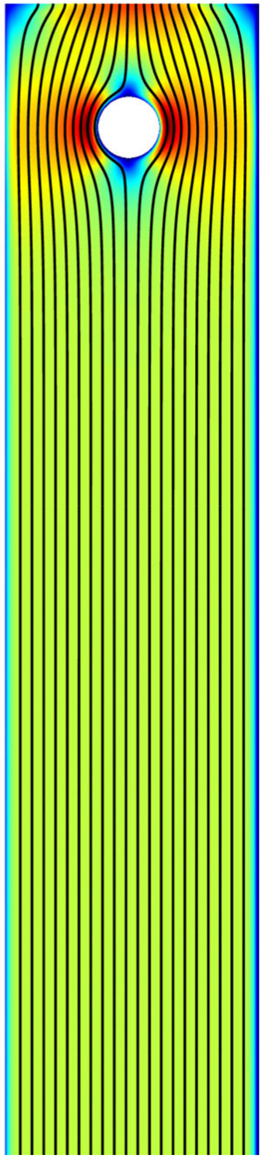


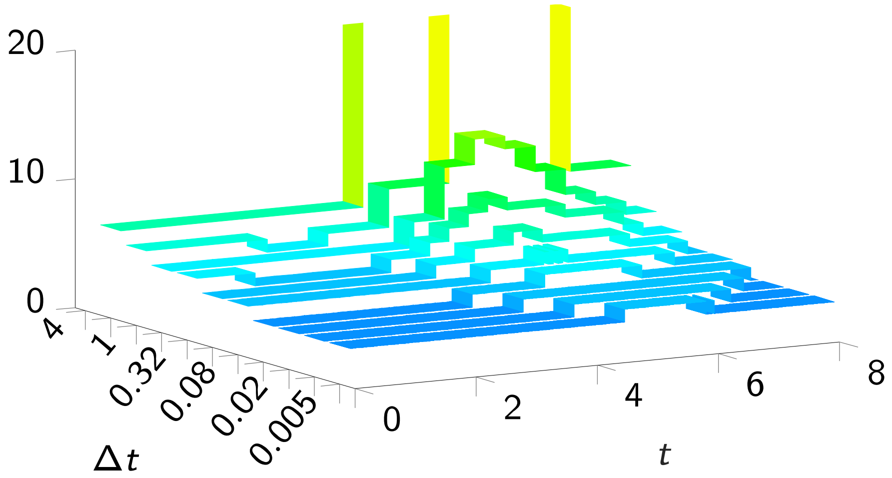
→ **24 g-i-t damped Newton iterations** (2048 time steps with dt=1/256) mainly due to “bad” starting values (finally: quadratic convergence)

Alternative:

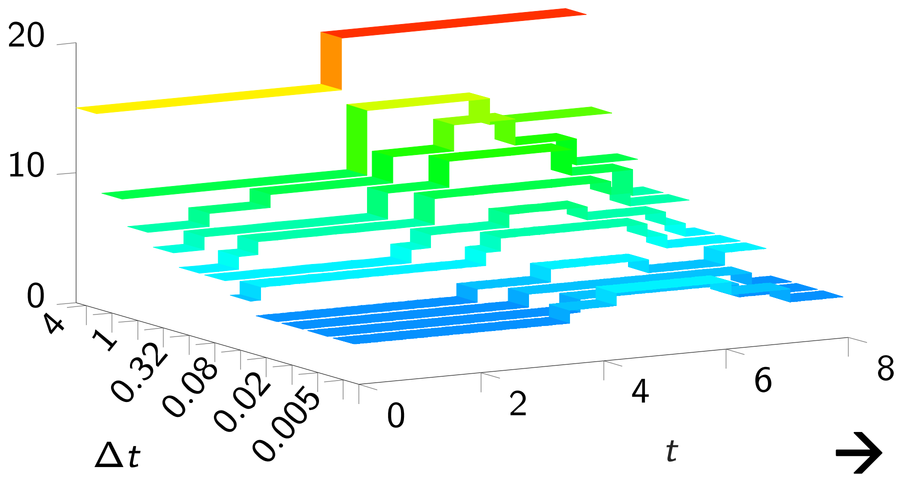
Adaptive **Picard-Newton** (Pollock, Rebholz et al.)

Proof of Concept: „FAC“ Benchmarks: BENCH3 – g-i-t Picard-Newton





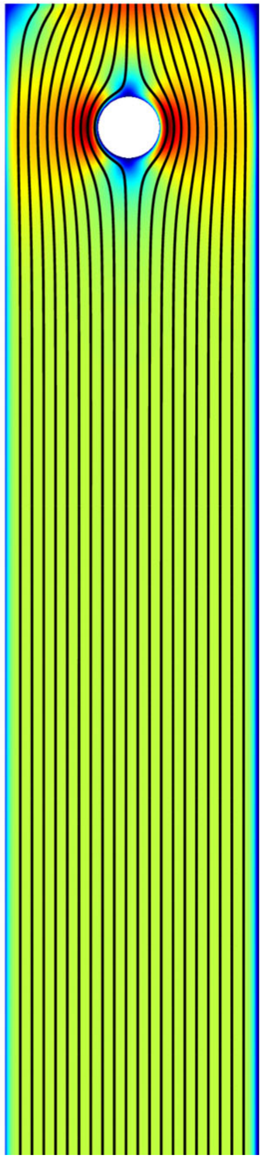
	mean	min	max
$\Delta t = 4$	–	–	–
$\Delta t = 1$	–	5	–
$\Delta t = 0.32$	5.76	4	9
$\Delta t = 0.08$	4.41	4	6
$\Delta t = 0.02$	3.46	3	4
$\Delta t = 0.005$	3.17	3	4

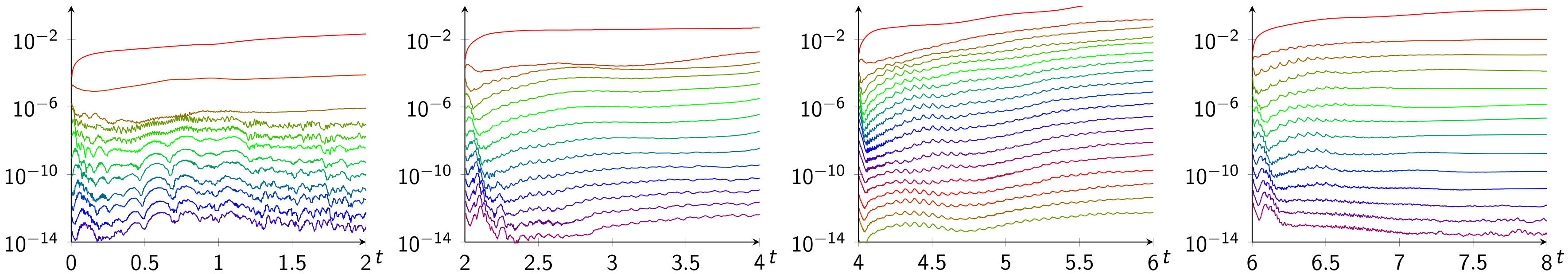


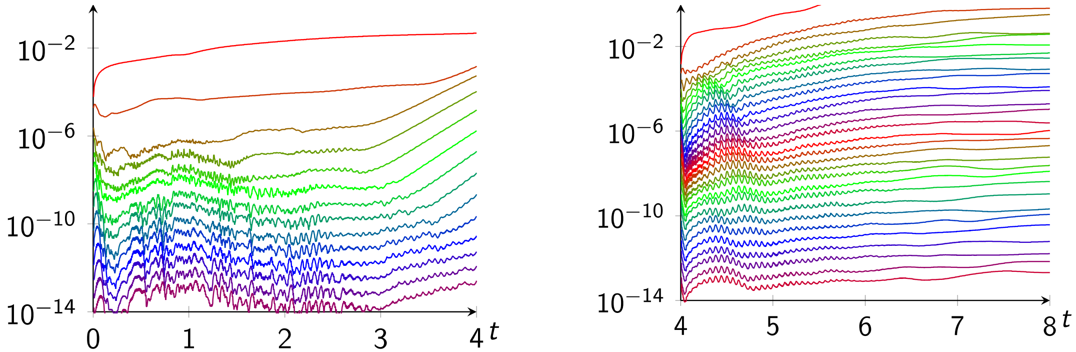
	mean	min	max
$\Delta t = 4$	17.00	15	19
$\Delta t = 1$	9.63	7	13
$\Delta t = 0.32$	7.56	5	9
$\Delta t = 0.08$	5.65	4	7
$\Delta t = 0.02$	3.46	3	4
$\Delta t = 0.005$	3.68	3	5

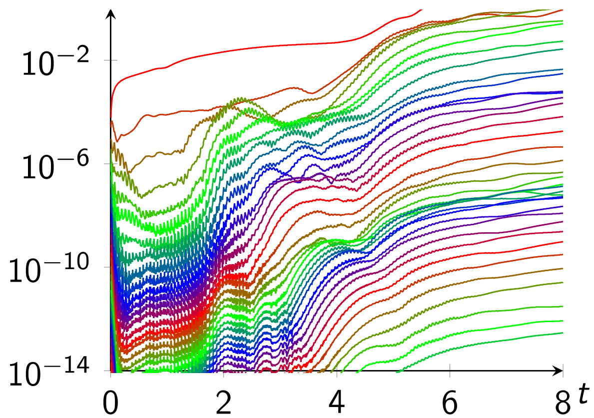
→ Anderson Acceleration for **Picard-Newton** ?

Proof of Concept: „FAC“ Benchmarks: BENCH3 – g-i-t Oseen solver









→ **g-i-t Oseen MG solver** depends on blocked time steps K

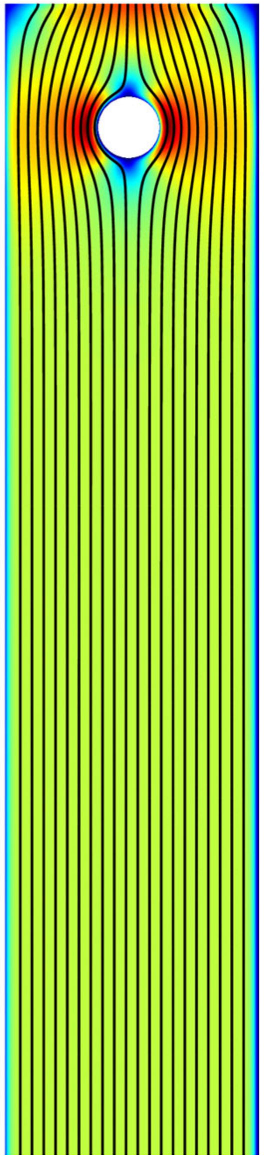
How to improve the solution behavior to be independent of K In the case of higher Re???

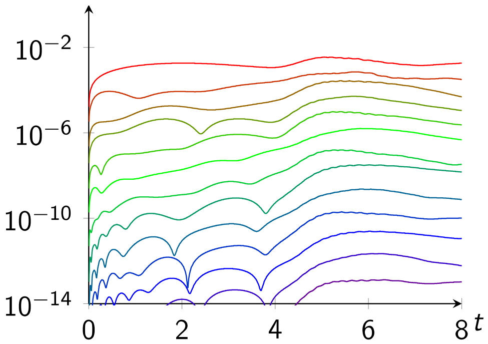
→ **Augmented Lagrange (AL)**

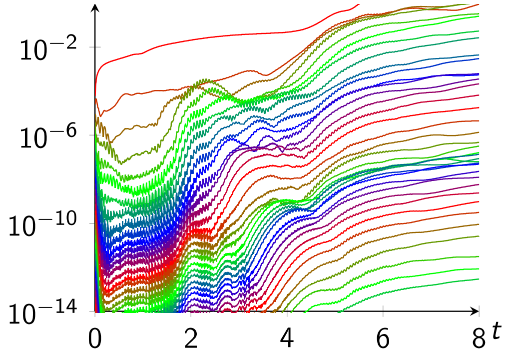
→ **Coupled Vanka**

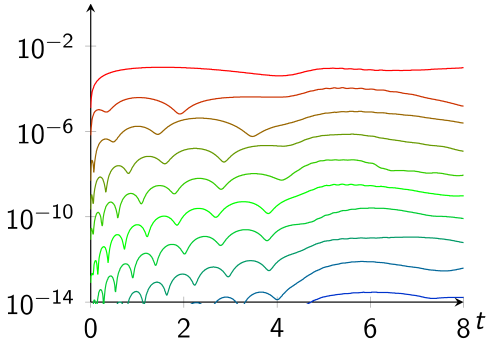
→ **Discretely Div.-free (DDF)**

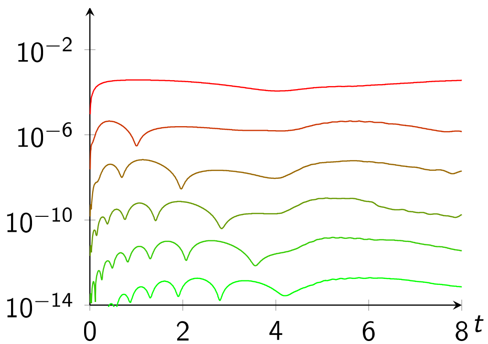
Proof of Concept: „FAC“ Benchmarks: BENCH3 – g-i-t AL-Oseen solver











→Very **moderate number** of **g-i-t Oseen solver steps** per Newton step if **AL stabilization** parameter is large enough **since PSC preconditioner gets exact**

→No need for multigrid! → PGMRES

→**But: Special MG solvers** for “velocity problems” required!

Augmented momentum equation

$$\mathrm{A}_i \mathrm{u}^{(n+1)} + \mathrm{B}\tilde{\mathrm{p}}^{(n+1)} = \tilde{\mathrm{g}}^{(n+1)} + \gamma \delta t \mathrm{B} \mathrm{W}^{-1} (\mathrm{f}^{(n+1)} - \mathrm{B}^\top \mathrm{u}^{(n+1)})$$

Using $\mathrm{W} = \mathrm{M}_p$ and $\gamma > 0$

Stabilized system of equations:

$$\begin{pmatrix} \mathrm{A}_i + \gamma \delta t \mathrm{B} \mathrm{M}_p^{-1} \mathrm{B}^\top & \mathrm{B} \\ \mathrm{B}^\top & \end{pmatrix} \begin{pmatrix} \mathrm{u}^{(n+1)} \\ \tilde{\mathrm{p}}^{(n+1)} \end{pmatrix} = \begin{pmatrix} \tilde{\mathrm{g}}^{(n+1)} - \mathrm{A}_e \mathrm{u}^{(n)} + \gamma \delta t \mathrm{B} \mathrm{M}_p^{-1} \mathrm{f}^{(n+1)} \\ \mathrm{f}^{(n+1)} \end{pmatrix}$$

Sherman-Morrison-Woodbury identity guarantees[1]

$$\boxed{\mathrm{P}_{i,\gamma}^{-1} = \mathrm{P}_i^{-1} + \gamma \delta t \mathrm{M}_p^{-1} \approx \mathrm{C}_i^{-1} + \gamma \delta t \mathrm{M}_p^{-1}}$$

[1] Benzi and Olshanskii (2006) and Wechsung (2019)

$$
\begin{pmatrix} \mathbf{A}_K + \gamma\delta t \mathbf{B}_K \mathbf{M}_{K,p}^{-1} \mathbf{B}_K^\top & \mathbf{B}_K \\ \mathbf{B}_K^\top & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \tilde{\mathbf{p}} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{g}} + \gamma\delta t \mathbf{B}_K \mathbf{M}_{K,p}^{-1} \mathbf{f} \\ \mathbf{f} \end{pmatrix}
$$

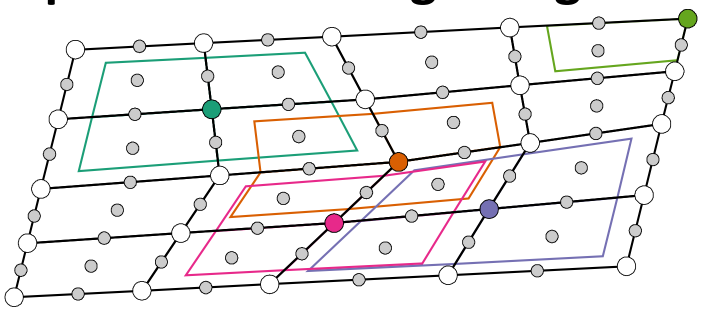
$$
:\Longleftrightarrow \left(\begin{array}{cccc|cccc}
\mathrm{A}_{i,\gamma} & & & & \mathrm{B} & & & \\
\mathrm{A}_e & \mathrm{A}_{i,\gamma} & & & & \mathrm{B} & & \\
& \ddots & \ddots & & & & \ddots & \\
& & \mathrm{A}_e & \mathrm{A}_{i,\gamma} & & & & \mathrm{B} \\
\hline
\mathrm{B}^\top & & & & & & & \\
& \mathrm{B}^\top & & & & & & \\
& & \ddots & & & & & \\
& & & \mathrm{B}^\top & & & &
\end{array}\right)
\left(\begin{array}{c}
\mathrm{u}^{(1)} \\ \mathrm{u}^{(2)} \\ \vdots \\ \mathrm{u}^{(K)} \\ \hline \tilde{\mathrm{p}}^{(1)} \\ \tilde{\mathrm{p}}^{(2)} \\ \vdots \\ \tilde{\mathrm{p}}^{(K)}
\end{array}\right)
=
\left(\begin{array}{c}
\tilde{\mathrm{g}}^{(1)} - \mathrm{A}_e \mathrm{u}^{(0)} + \gamma\delta t \mathrm{B}\mathrm{M}_p^{-1}\mathrm{f}^{(1)} \\
\tilde{\mathrm{g}}^{(2)} + \gamma\delta t \mathrm{B}\mathrm{M}_p^{-1}\mathrm{f}^{(2)} \\
\vdots \\
\tilde{\mathrm{g}}^{(K)} + \gamma\delta t \mathrm{B}\mathrm{M}_p^{-1}\mathrm{f}^{(K)} \\
\hline
\mathrm{f}^{(1)} \\ \mathrm{f}^{(2)} \\ \vdots \\ \mathrm{f}^{(K)}
\end{array}\right)
$$

Using $\mathrm{A}_{i,\gamma} = \mathrm{A}_i + \gamma\delta t \mathrm{B}\mathrm{M}_p^{-1}\mathrm{B}^\top$

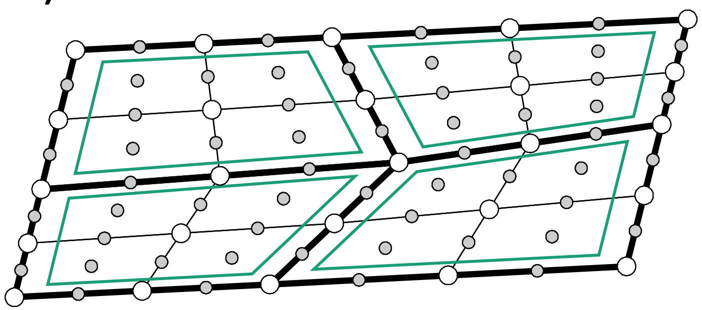
$$
\boxed{\mathbf{P}_{K,\gamma}^{-1} = \mathbf{P}_K^{-1} + \gamma\delta t \mathbf{M}_{K,p}^{-1} \approx \mathbf{C}_K^{-1} + \gamma\delta t \mathbf{M}_{K,p}^{-1}}
$$

$$\begin{pmatrix} A_i + \gamma\delta t B M_p^{-1} B^\top & B \\ B^\top & \end{pmatrix} \begin{pmatrix} u^{(n+1)} \\ \tilde{p}^{(n+1)} \end{pmatrix} = \begin{pmatrix} \tilde{g}^{(n+1)} - A_e u^{(n)} + \gamma\delta t B M_p^{-1} f^{(n+1)} \\ f^{(n+1)} \end{pmatrix}$$

Specialized multigrid algorithm:



a) Smoother.



b) Prolongation.

Standard multigrid using Jacobi smoother:

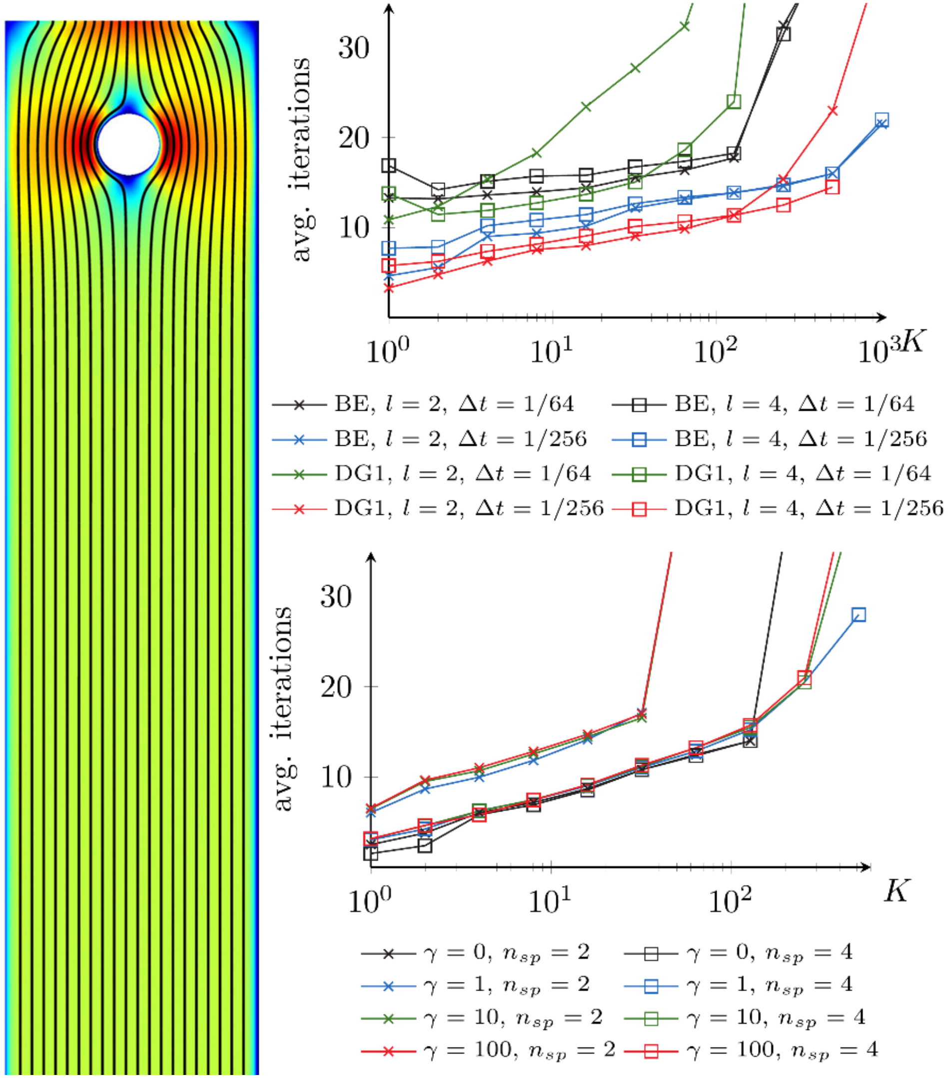
lvl	”$t^{\neq 1}$ ╲ “	0	$10^{\neq 2}$	$10^{\neq 1}$	10^0	10^2	10^4
1	50	2	4	11	79	≠	≠
2	100	2	6	23	204	≠	≠
3	200	3	8	38	380	≠	≠
4	400	3	10	56	≠	≠	≠

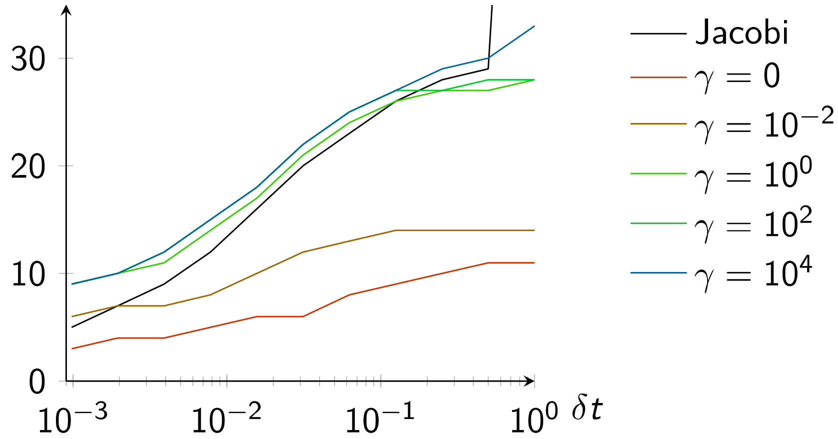
Specialized multigrid:

lvl	”$t^{\neq 1}$ ╲ “	0	$10^{\neq 2}$	$10^{\neq 1}$	10^0	10^2	10^4
1	50	2	2	3	4	5	5
2	100	2	3	4	4	6	5
3	200	2	3	5	5	6	6
4	400	2	4	5	6	6	6

[1]Wechsung (2019)

Proof of Concept: „FAC“ Benchmarks: BENCH3 – PinT-MG solvers for nonstationary convection-diffusion-reaction problems

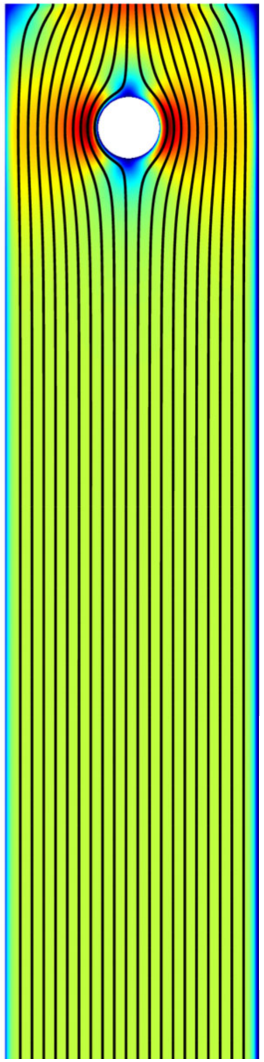


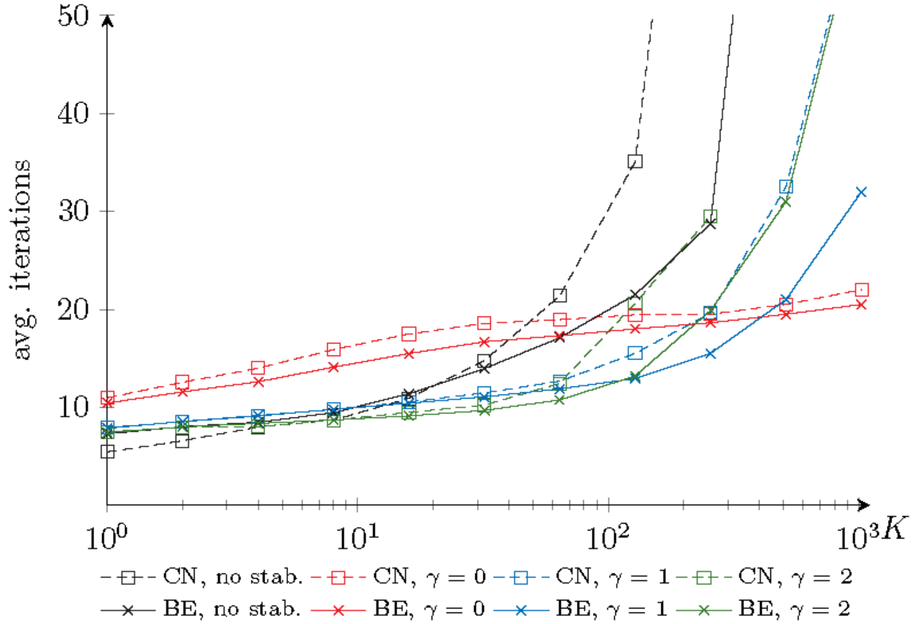


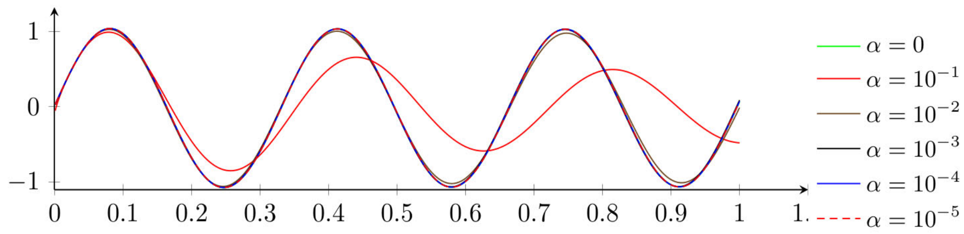
Robust behavior with special **Patch-Jacobi Smoother**

→**our recent candidate for HPC realization**

Proof of Concept: „FAC“ Benchmarks: BENCH3 – SinT-MG solvers for nonstationary convection-diffusion-reaction problems





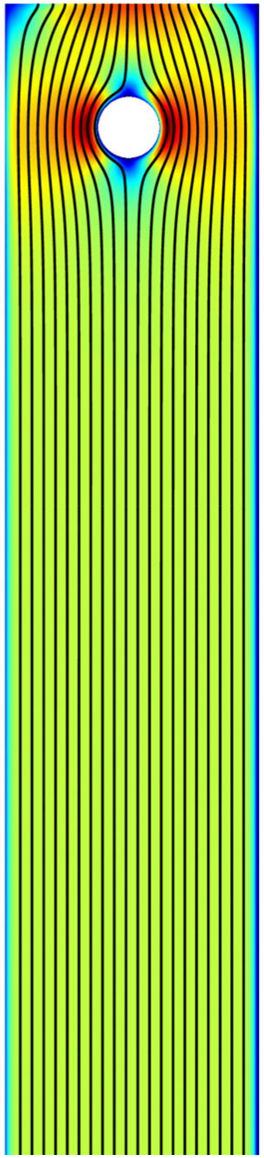


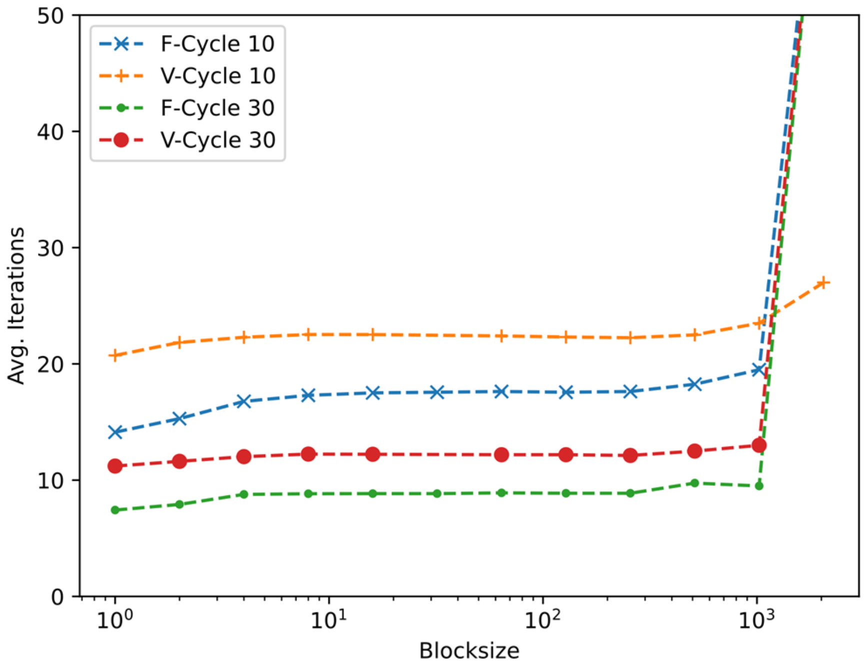
Robust behavior ONLY with appropriate **stabilization** as usual for **SinT-MG** with **block-Jacobi smoothers,** otherwise only for **moderate K**

→ **Stabilization** (here: VMS type) necessary for solver AND for accuracy & robustness

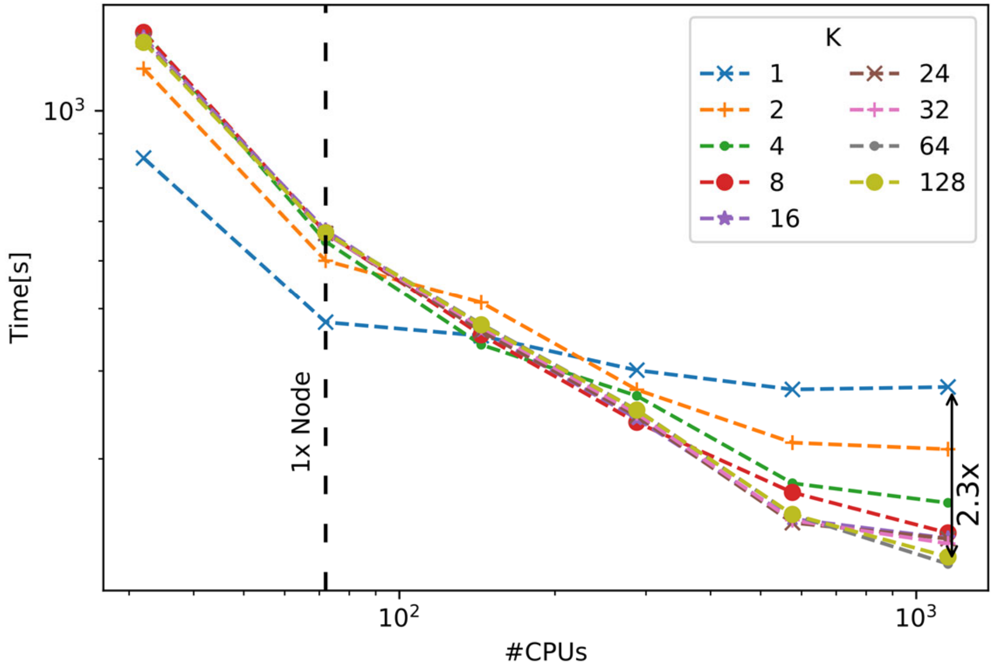
→ Similar behavior for **AL**

Proof of Concept: „FAC“ Benchmarks: BENCH3 – SinT Vanka MG solver





Comparison of different smoothing steps to reach tolerance of 10^{-10} for Re_{max}=100.



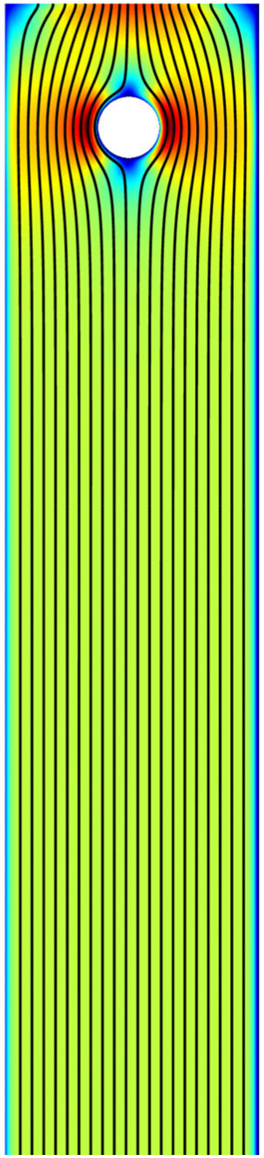
Time to solve last Newton step for Re_{max}=100.

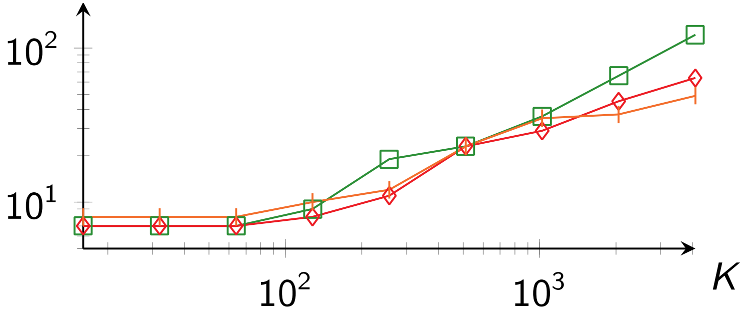
(Also) Very interesting candidate for **nonstationary problems** with **larger time step sizes**

→Optimal realization!

→Complex meshes?

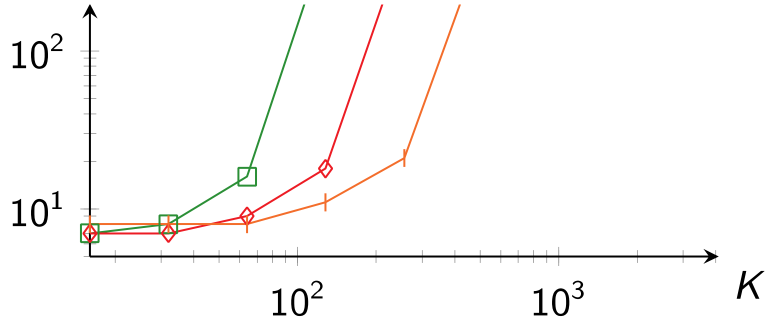
Proof of Concept: „FAC“ Benchmarks: BENCH3 – g-i-t DDF-Oseen solver



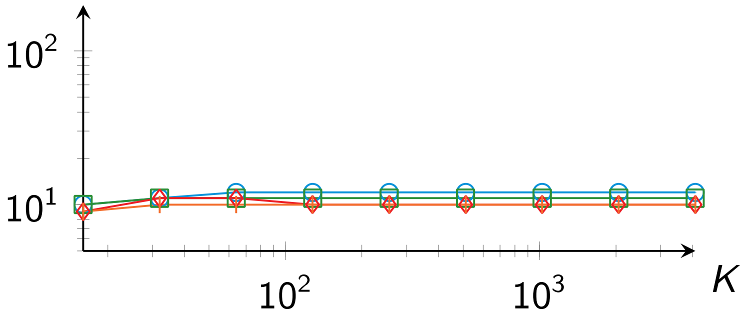


Final Newton iteration, $\mathrm{Re}_{\max} = 20$.

→ SinT

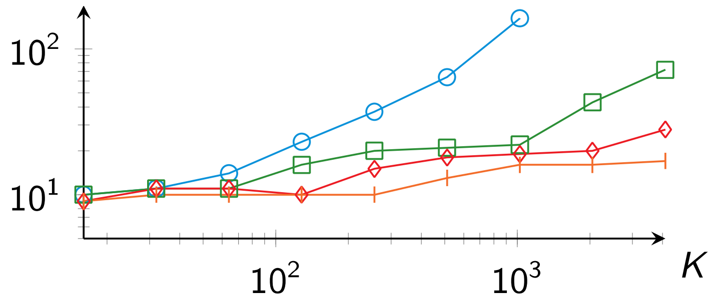


Final Newton iteration, $\mathrm{Re}_{\max} = 100$.



Final Newton iteration, $\mathrm{Re}_{\max} = 20$.

→ **PinT**



Final Newton iteration, $\mathrm{Re}_{\max} = 100$.

Very interesting candidate for **special FEM spaces** and for **large number** of **time steps K**

→3D realization by Chr. Lohmann (first since Thomasset and Hecht in 80s?......)

Can we use Massively Parallel & Lower Precision Accelerator Hardware for Extreme Scale High Performance Computing for Flow Problems of Industrial Relevance?

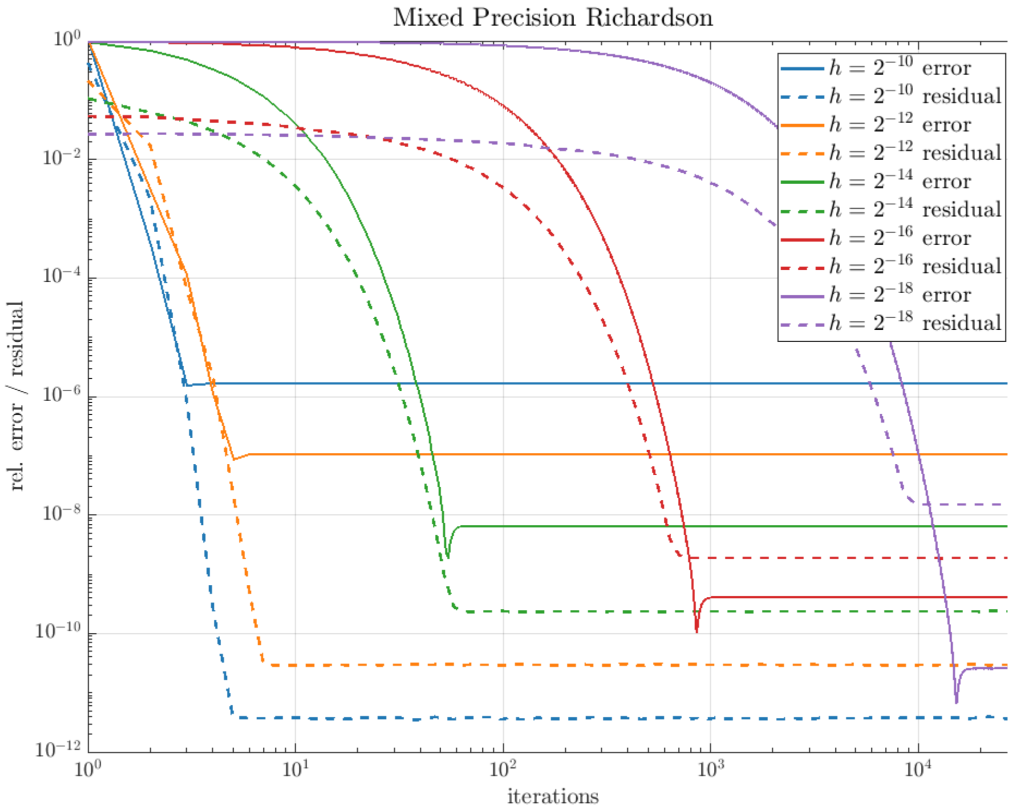
Yes, it seems to be possible!

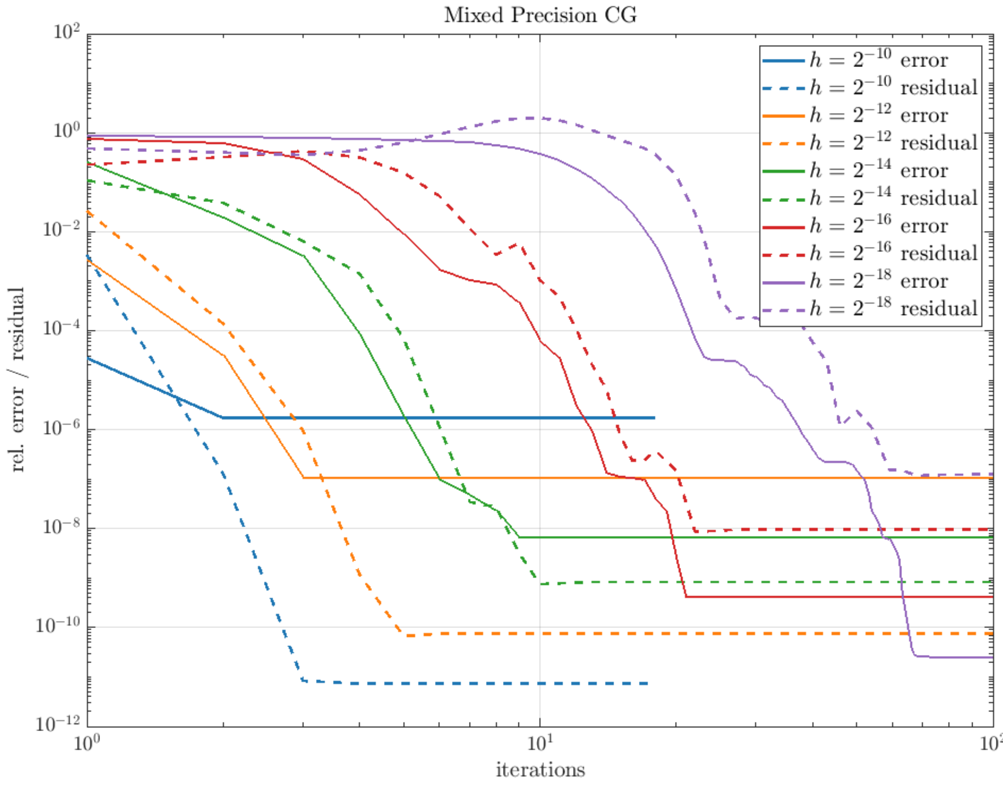
Can we use Massively Parallel & Lower Precision Accelerator Hardware for Extreme Scale High Performance Computing for Flow Problems of Industrial Relevance?

Or: Yes, it must be possible!

Problem 3b: Lower precision hardware for Poisson problems?

→Standard approach: Mixed Precision Defect Correction





Wish: Directly solve in FP32/TF32

Complexity and performance estimate (in detail, on A100)

- Exemplary case: 3D unit cube, $h = {}^{1}/_{256}$ ($N \approx 16.6 \times 10^6$), $h_0 \in \{{}^{1}/_{16}, {}^{1}/_{32}\}$

	$^1/_{16}$	$^1/_{32}$
Total number of Flop relative to N	115,000	23,400
Multiplications with C^{-1}	98%	77%
GFlop/s	116,000	57,000
Multiplications with A_1	0.8%	11.8%
GFlop/s	1,700	1,730
Multiplications with B and B^T	1.2%	10.1%
GFlop/s	1,600 / 2,600	1,500 / 2,600
BLAS1 (axpy, dot products, additions)	0.05%	0.8%
GFlop/s	130 – 380	130 – 380
Total GFlop/s	47,000	6,000
Total MDof/s	**412**	**260**