Prehandling and Related Hardware-Oriented Finite Element PDE Solvers Enabling Lower Precision and Tensor Core Computations

Dustin Ruda, Stefan Turek, Dirk Ribbrock

Chair of Applied Mathematics and Numerics (LS3), TU Dortmund University

ENUMATH 2025, September 1–5, Heidelberg

September 4, 2025

technische universität dortmund

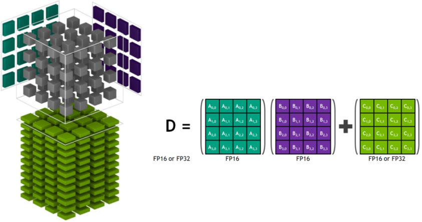
fakultät für mathematik

lehrstuhl für angewandte mathematik und numerik

Motivation – Hardware trends

Tensor Cores (TC)

- Processing units by Nvidia specialized to accelerate AI applications
- Can perform dense matrix multiplications very fast
- Examples of TC GPUs: V100 (2017), A100 (2020), H100 (2023), B200 (2024)



Schematic representation of fused multiply-add ($D = AB + C$) with 4×4 matrices on TC

	FP64	FP64 TC	FP32	FP32 TC / 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	n/a	40	n/a	1,100	n/a	2,250

TFlop/s peak rates (realistically achievable)

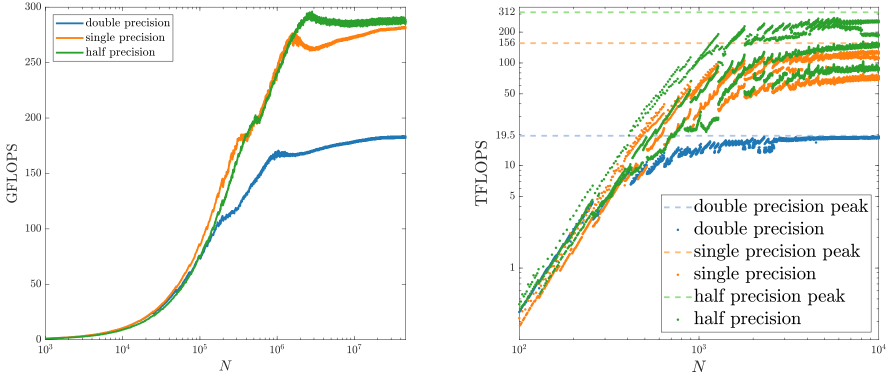
Motivation – Hardware trends

Rank	Name	Accelerator
1	El Capitan	MI300A
2	Frontier	MI250X
3	Aurora	Intel Max GPU
4	JUPITER Booster	GH200
5	Eagle	H100
6	HPC6	MI250X
7	Fugaku	–
8	Alps	GH200
9	LUMI	MI250X
10	Leonardo	A100

Accelerator hardware in supercomputers

- Technology similar to TC by AMD: Matrix Cores (MI250X, MI300A)
- 8 supercomputers in top 10 of TOP500 (June 2025) use Nvidia (4) or AMD (4) accelerator hardware

Motivation – Sparse vs. dense



GFLOPS on A100 for sparse 5-point stencil matrix (CSR) × vector (left) and TFLOPS for dense matrix × dense matrix (right)

Problem statement

- Consider **Poisson's equation**

$$-\Delta u = f \text{ on } \Omega \subset \mathbb{R}^d,\ d \in \{2,3\}$$

  as very a common (sub-)problem and bottleneck in many (e.g. CFD) applications
- TC GPUs have a performance potential of 100+ TFlop/s
- But it is only achievable in **lower precision** (SP or HP) and for **dense** matrix operations

↯ Both contradict basic principles of standard solvers (e.g. multigrid (MG)) for finite element (FE) simulations: Low precision might cause loss of **accuracy** due to high condition numbers ($\mathcal{O}(h^{-2})$) and FE matrices are **sparse**

Aims

- Profitable use of lower precision TC hardware for linear systems in matrix-based FE simulations by constructing suitable hardware oriented solvers

Two-step process:

- **Step 1:** Manipulate linear system to enable SP or HP while preserving sufficient accuracy
- **Step 2:** Adapt solver by densifying operations to leverage TC (large + sparse $\rightarrow$ small + dense)

Remark

- Also consider many right hand sides (RHS), resp., dense matrix as RHS ($AX = B$)
- Exemplary use case: Global-in-time Navier–Stokes solver $\rightarrow$ solve pressure Poisson problem for all time steps at once

Basic procedure

Starting point: Sparse, ill-conditioned linear system

Prehandling to lower condition number

2D: HFEM

3D: Generating systems

Node **renumbering** exploiting similar cells + **Schur complement(s)**

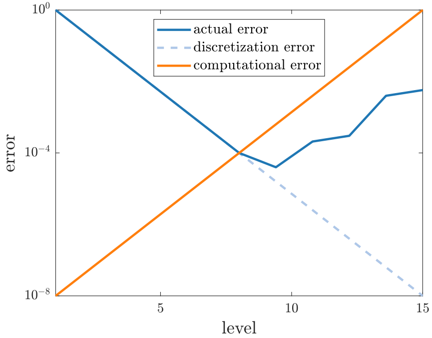
(**Direct method** in 2D)

Semi-iterative method in 2D and 3D

Prehandling – How to handle ill-conditioned Poisson problems

- Error consists of discretization and computational error: $u - \tilde{u}_h = (u - u_h) + (u_h - \tilde{u}_h)$
- **Discr. error:** $\|u - u_h\| = \mathcal{O}(h^{p+1})$
  - Depends on FE space and smoothness
  - Here for simplicity: $p = 1$
- **Comp. error:** $\|u_h - \tilde{u}_h\| \approx \text{cond} \cdot$ "data error"
  - Data error at least TOL of precision
  - Poisson: $\text{cond}(A_h) = \mathcal{O}(h^{-2})$
- Comp. error becomes dominant at h_{crit} at intersection of both errors
- Omit constant factors and equate

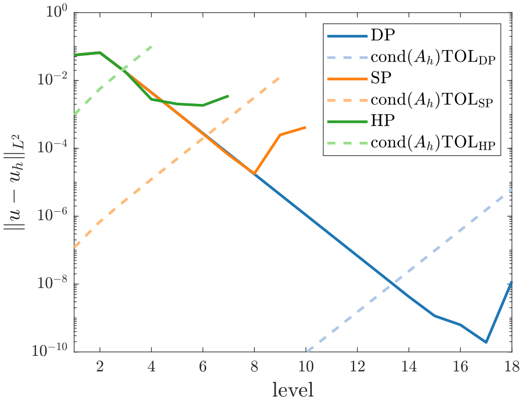
$$\Rightarrow h_{\text{crit}} \approx (\text{cond} \cdot \text{TOL})^{\frac{1}{2}}$$



Illustrative example of actual, discretization and computational error

Prehandling – How to handle ill-conditioned Poisson problems

- Critical mesh size: $h_{\text{crit}} \approx (\text{cond} \cdot \text{TOL})^{\frac{1}{2}}$
- Substitute $\text{cond} \approx h^{-2}$ for Poisson's equation $\Rightarrow h_{\text{crit}} \approx \text{TOL}^{\frac{1}{4}}$
- Example: $(\text{TOL}_{\text{DP}})^{\frac{1}{4}} = 2^{-13} \approx 10^{-3.9}$
  $(\text{TOL}_{\text{SP}})^{\frac{1}{4}} = 2^{-5.75} \approx 10^{-1.7}$
  $(\text{TOL}_{\text{HP}})^{\frac{1}{4}} = 2^{-2.5} \approx 10^{-0.8}$
- Wish: $\text{cond} = \mathcal{O}(1) \Rightarrow h \approx \text{TOL}^{\frac{1}{2}}$
  $\rightarrow$ SP (and even HP?) possible



Computational and actual L^2-error for 1D example

The concept of prehandling of linear systems

Basic idea

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

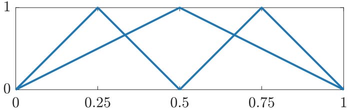
Central requirements for prehandling

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

Candidates for prehandling

- 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

Candidates for prehandling – HFEM

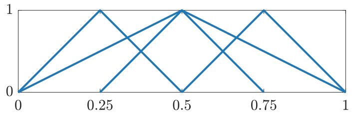


minimum example of hierarchical basis in 1D

- Sequence of refined meshes starting from coarse mesh (h_0) required
- Transformation matrix S is square, $\tilde{A} = S^\mathsf{T} A S$ is symm. positive-definite and sparse
- $\mathrm{cond}(\tilde{A}) = \mathcal{O}\left(\left(\log \frac{1}{h}\right)^2\right)$ (3D: $\mathcal{O}(h^{-1})$)
- Partial Cholesky decomposition on coarse mesh to lower cond:

$$\begin{pmatrix} \widetilde{A}_0 & 0 \\ 0 & \widetilde{D} \end{pmatrix} = L^\mathsf{T} L \to L^{-1} \widetilde{A} L^{-\mathsf{T}}$$

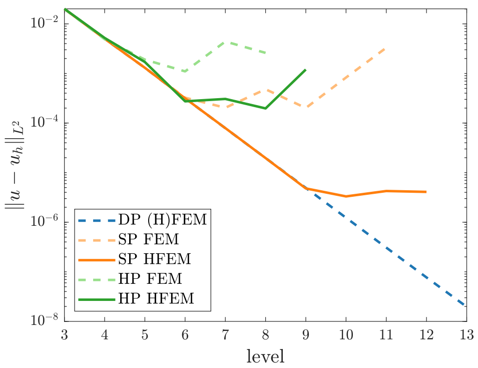
Candidates for prehandling – Generating systems

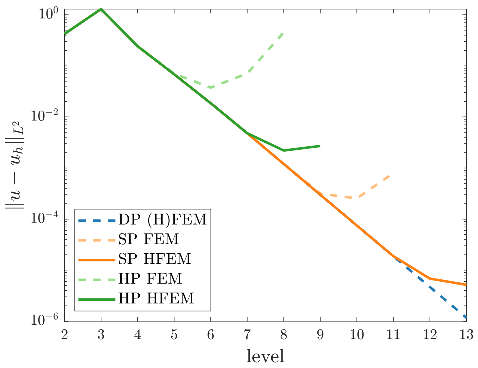


minimum example of generating system in 1D

- Also dependent on sequence of refined meshes
- GS consists of nodal bases on **all** levels and thus, some mesh points are counted repeatedly
- Transformation matrix S is rectangular, $\tilde{A} = S^{\mathsf{T}} A S$ is symm. pos. semi-definite (sufficient for convergence of CG method to non-unique solution) and sparse
- Transforming back by multiplication with S yields unique solution to original system
- Magnification factor of problem size: $^{8}/_{7}$ in 3D
- Jacobi preconditioner corresponds to BPX $\Rightarrow \mathrm{cond}(\tilde{A}) = \frac{\lambda_{\max}}{\lambda_{\min,>0}} = \mathcal{O}(1)$

The effect of prehandling on accuracy in 2D

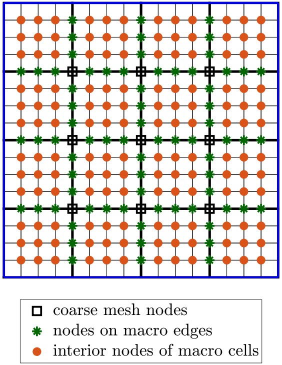


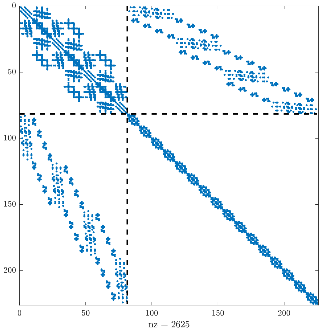


L^2-errors for different levels in 2D in DP, SP, HP without (dashed) and with (solid) **prehandling via HFEM** for **“very smooth”** (left) and **more oscillating** (right) solution

Solvers taylored for Tensor Core GPUs

- Low condition numbers by prehandling enable low precision but **matrices are still sparse**
- Construct solver consisting as much as possible on multiplications with dense matrices
- Same principle in 2D (HFEM) and 3D (GS): Subdivide nodes into:
  - nodes in the interior of the coarse mesh cells (cell by cell in same order)
  - "all remaining nodes" containing those on coarse mesh edges (+ repeated nodes of GS)





- Matrix form:

$$\begin{pmatrix} A_1 & B \\ B^\top & C \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

- C decomposes into independent blocks C_i
- Blocks are equal if corresponding to similar cells
- Only C grows like N $(= \#\text{Dof})$

Solvers taylored for Tensor Core GPUs

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

Semi-iterative Method

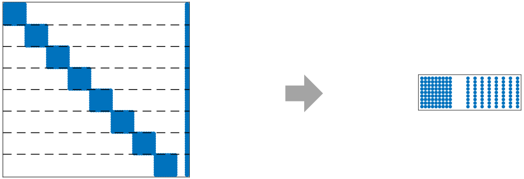
$$\underbrace{\left(A_1 - BC^{-1}B^\mathsf{T}\right)}_{\hat{A}} u = b_1 - BC^{-1}b_2 \qquad \text{(with CG)}$$

$$v = C^{-1}\left(b_2 - B^\mathsf{T}u\right)$$

- $\hat{A}$ can be computed explicitly (2D) or used implicitly (better option in 3D)
- Robust with respect to anisotropic meshes with high aspect ratios (moderate increase of iteration numbers)
- In special cases: Structure of $\hat{A}$ allows further Schur complement yielding the **direct method** with high memory usage, but highly performant (up to 60 TFlop/s)
  (Ruda, D. et al., Very fast finite element Poisson solvers on lower precision accelerator hardware: A proof of concept study for Nvidia Tesla V100, IJHPCA, 2022)

How to treat multiplications with C^{-1}

- Only C_i need to be inverted (once for each group of similar macro cells)
  $\longrightarrow C^{-1}$ block diagonal matrix with dense blocks C_i^{-1}
- C_i are small, well-conditioned HFEM matrices, $\mathcal{O}(N)$ storage for C_i^{-1}
- Efficient implementation by simultaneous computation



- Complexity $\mathcal{O}\left(N^{3/2}\right)$ but very fast calculation by TC almost at peak performance

Storage requirement of the semi-iterative method

- Exemplary toy problem: Poisson's 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
128	2.05	4	11.3	433.3	15.4	**460**
		8	22,1	5.6	16.6	**44**
		16	37.1	0.1	15.3	**52**
256	16.58	8	14.2	53.5	16.5	**84**
		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 DP)
- Explicit $\hat{A}$ in 3D: $400N - 1,400N \longrightarrow$ implicit variant preferred

Complexity and performance estimate

Semi-iterative Method

$$\underbrace{\left(A_1 - BC^{-1}B^\mathsf{T}\right)}_{\hat{A}} u = b_1 - BC^{-1}b_2$$

$$v = C^{-1}\left(b_2 - B^\mathsf{T}u\right)$$

- **Composition of the method:**
  - $1 \times B$, $1 \times C^{-1}$ to compute RHS
  - **Iterative step:** $1 \times \hat{A}$ (explicit or implicit) + 2 dot products + 3 axpy per iteration
  - Intermediate step: $1 \times B^\mathsf{T}$
  - **Direct Step:** $1 \times C^{-1}$

- Entire method in SP/TF32 on TC GPU
- Majority of the work: Dense matrix operations; Small part: sparse $\times$ dense and BLAS1
- Matrix properties, iteration numbers and benchmark results on A100 (H100) in SP for all occuring operations (given many RHS) $\longrightarrow$ **performance model**
- Metric beyond Flop/s for comparability: millions of unknowns solved per second (MDof/s)

Performance estimate

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

- Comparative result with **optimized MG** in C++-based FE software package (FEAT3) on AMD CPU in DP: $\leq$ **15 MDof/s**

Results on H100

- Results on **H100** GPU[1] ($\approx 3\times$peak rates of A100 in SP with TC)
- Speedup 1.3–1.9 for sparse×dense and 1.5–3.5 for dense×dense compared to A100
- **Comparison of MDof/s:**

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 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`

Prehandling for convection-diffusion

- So far: Poisson's equation; in general similar results for self-adjoint, positive definite, elliptic, linear, second order PDEs (proven)
- Study generating systems for stationary **convection-diffusion** problem numerically (in spite of lack of proof for $\mathrm{cond}(\tilde{A}) \ll \mathrm{cond}(A)$)

$$-\varepsilon \Delta u + b \cdot \nabla u = f \text{ on } \Omega \subset \mathbb{R}^2, \text{ diffusion coefficient } \varepsilon > 0, \text{ convection field } b : \mathbb{R}^2 \to \mathbb{R}^2$$

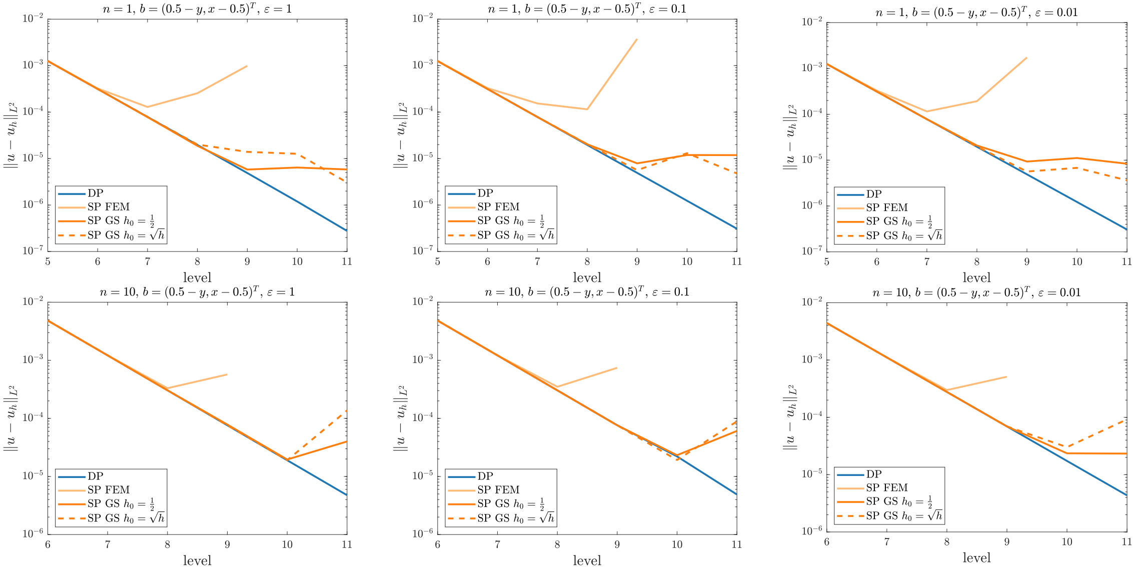
- Vortical convection $b(x, y) = (1/2 - y, x - 1/2)^{\mathsf{T}}$ moderate ($\varepsilon \geq 10^{-2}$) so that standard FE discetization is stable
- Consider GMRES preconditioned by Jacobi, Gauss–Seidel (GS), symmetric GS (SGS) and compare matrix density, iteration numbers and accuracy for standard FEM vs. generating systems

Prehandling for convection-diffusion: iteration numbers

$\frac{1}{h}$	$\frac{1}{h_0}$	$\frac{NNZ}{N}$	plain/Jacobi $\varepsilon = 1$	plain/Jacobi 10^{-1}	plain/Jacobi 10^{-2}	GS 1	GS 10^{-1}	GS 10^{-2}	SGS 1	SGS 10^{-1}	SGS 10^{-2}
512	FEM	9	629	678	1,010	728	850	1,797	260	279	527
	2	87	19	19	31	11	11	19	6	6	9
	4	84	20	21	32	11	11	19	6	6	10
	8	78	22	23	36	14	14	20	8	8	10
1024	FEM	9	1,259	1,356	2,018	1,462	1,657	3,694	516	556	1,031
	2	98	20	20	32	11	11	19	6	6	9
	4	95	21	22	34	11	11	19	6	6	10
	8	89	24	24	37	14	14	20	8	8	10
2048	FEM	9	2,517	2,712	4,034	2,861	3,182	7,397	1,031	1,111	2,050
	2	109	21	21	34	11	11	19	6	6	10
	4	106	23	23	36	11	11	20	6	6	10
	8	100	25	26	38	14	15	20	8	9	10

- Bottom line: Prehandling via generating systems works for moderate convection (**Step 1** ✓)
- Perfomance gain by low precision and sparse × dense (**Step 2** more challenging)

Prehandling for convection-diffusion: accuracy



Conclusion and outlook

Conclusion

- It is possible to exploit Lower-Precision and also Accelerator Hardware for PDE computing by prehandling and a related semi-iterative approach

Outlook

- Deeper analysis of suitable preconditioners for the iterative step and initial guesses for the solution vector to reduce number of iterations
- Investigate the feasibility of transforming different matrices C_i into one

References

Literature

- Ruda, D., Turek, S. & Ribbrock, D. (2025). Fast Semi-Iterative Finite Element Poisson Solvers for Tensor Core GPUs Based on Prehandling. Sequeira, A. et al., Lecture Notes in Computational Science and Engineering, 154, 320–330, Numerical Mathematics and Advanced Applications Enumath 2023, Vol 2., Springer,

  DOI: 10.1007/978-3-031-86169-7_33
- Ruda, D., Turek, S., Zajac, P. & Ribbrock, D. (2022). Very fast finite element Poisson solvers on lower precision accelerator hardware: A proof of concept study for Nvidia Tesla V100, International Journal of High Performance Computing Applications 36(4), pp.459–474,

  DOI: 10.1177/10943420221084657
- Ruda, D., Turek, S., Zajac, P. & Ribbrock, D. (2020). The Concept of Prehandling as Direct Preconditioning for Poisson-like Problems. Vermolen, F., Vuik, C., Lecture Notes in Computational Science and Engineering, 139, 1011-1019, Numerical Mathematics and Advanced Applications Enumath 2019, Springer,

  DOI: 10.1007/978-3-030-55874-1_100
- Yserentant, H. (1986). On the Multi-Level Splitting of Finite Element Spaces. Numer. Math., Vol. 49, pp. 379–412,

  DOI: 10.1007/BF01389538
- Griebel, M. (1994). Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen. Teubner Skripten zur Numerik.

  DOI: 10.1007/978-3-322-89224-9

Figures

- p.1: `https://developer.nvidia.com/blog/tensor-core-ai-performance-milestones/`
- pp. 6, 12: Ruda et al. (2022), IJHPCA 36(4) (see above)