Fast semi-iterative finite element Poisson solvers for Tensor Core GPUs

Dustin Ruda, Stefan Turek, Dirk Ribbrock, Peter Zajac

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

LS3 Oberseminar

July 17, 2023

technische universität dortmund

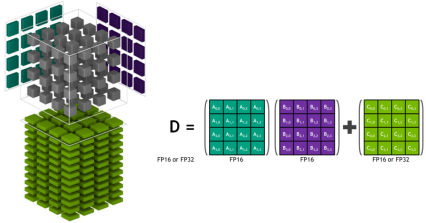
fakultät für mathematik

lehrstuhl für angewandte mathematik und numerik

Motivation

Tensor Cores (TC)

- Processing units by Nvidia specialized to accelerate AI applications
- Can perform matrix multiplications very fast
- Examples of TC GPUs: V100 (2017), A100 (2020), H100 (2023)
- Comparison: Fujitsu A64FX CPU (powering Fugaku)



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

	FP64	FP64 TC	FP32	FP32 TC	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
A64FX	3.4	-	6.8	-	13.5	-

TFlop/s peak rates (realistically achievable)

Motivation

- Similar technology by **AMD**: **Matrix Cores**
- Example AMD Instinct MI250X (2021); Peak rates:
  - FP64: 95.7 TFlop/s
  - FP32: 95.7 TFlop/s
  - FP16: 383 TFlop/s

Accelerator hardware in supercomputers

- 7 supercomputers in top 10 of TOP500 (June 2023) use Nvidia or AMD accelerator hardware

Rank	Name	Accelerator
1	Frontier	MI250X
2	Fugaku	-
3	LUMI	MI250X
4	Leonardo	A100
5	Summit	V100
6	Sierra	V100
7	Sunway TaihuLight	-
8	Perlmutter	A100
9	Selene	A100
10	Tianhe-2A	-

Motivation

Problem statement

- Consider **Poisson's equation** as very a common (sub-)problem in many 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**

Aim

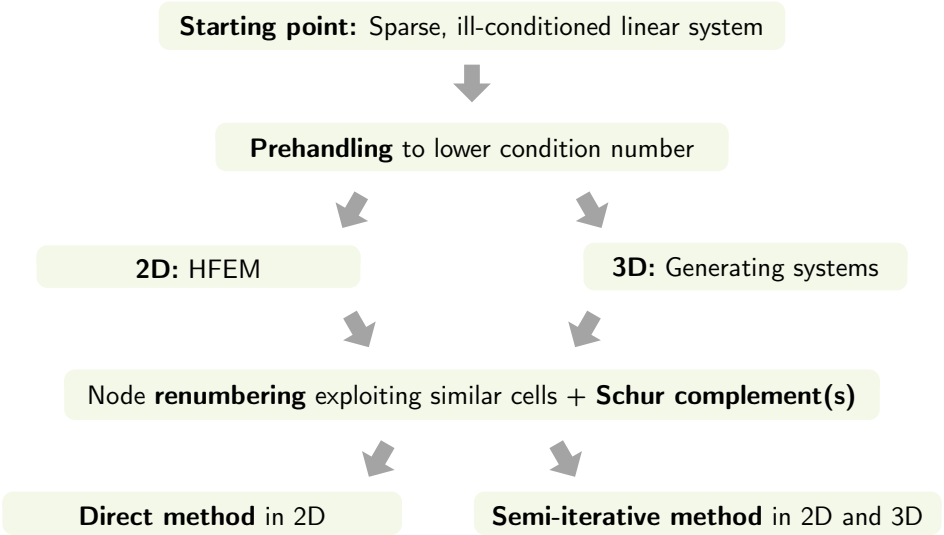
- Profitable use of TC hardware for linear systems in FE simulations (particularly for CFD) by constructing suitable hardware oriented solvers

Remark

- Global-in-time Navier–Stokes solver allows for solving pressure Poisson problem for all time steps at once $\rightarrow$ many right hand sides (RHS), resp., dense matrix as RHS[1]

[1] C. Lohmann, S. Turek: On the Design of Global-in-Time Newton-Multigrid-Pressure Schur Complement Solvers for Incompressible Flow Problems. J. Math. Fluid Mech. 25, 64 (2023)

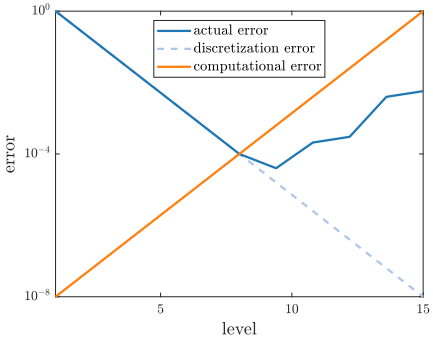
Basic procedure



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

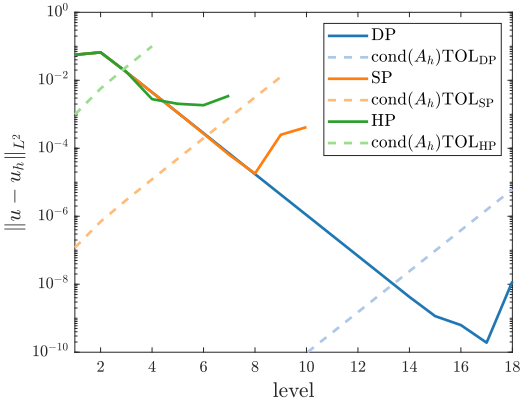
$$\Rightarrow h_{\text{crit}} \approx (\text{cond} \cdot \text{TOL})^{\frac{1}{2}}$$



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 $\mathrm{cond}(A) \neq \mathrm{cond}(\tilde{A})$

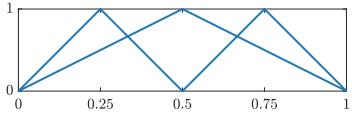
Central requirements for prehandling

- $\mathrm{cond}(\tilde{A}) \ll \mathrm{cond}(A)$
- $\tilde{A}$ 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** (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:

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

- $h_0 \approx \sqrt{h}$ good choice for variable coarse mesh size

HFEM in 2D – matrix density, condition and iteration numbers

Results on unit square (Q1):

$\frac{1}{h}$	$\frac{1}{h_0}$	$\frac{\mathrm{NNZ}}{N}$	cond	#iter (ichol)
1024	FEM	8.99	212,485.76	1,143 (395)
	16	20.32	39.87	49
	32	28.20	30.61	40
	64	59.95	22.47	31
2048	FEM	8.99	849,943.52	2,307 (771)
	32	23.33	39.98	48
	64	39.82	30.63	39
4096	FEM	9.00	3,399,774.60	4,658 (1,356)
	32	20.84	50.56	56
	64	29.24	40.00	47
	128	63.02	30.64	39

- Termination criterion in conjugate gradient method for HFEM based on retransformed residual to obtain comparable iteration numbers
- Low condition numbers + sparse matrices by means of prehandling with HFEM + Cholesky on coarse mesh
- Low iteration numbers by prehandling (even significantly lower than standard FE preconditioned by incomplete Cholesky)

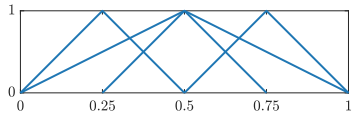
HFEM in 3D – matrix density, condition and iteration numbers

Results on unit cube (Q1):

		HFEM			HFEM + Cholesky		
$\frac{1}{h}$	$\frac{1}{h_0}$	$\frac{\text{NNZ}}{N}$	cond	#iter (ichol)	$\frac{\text{NNZ}}{N}$	cond	#iter
64	FEM	20.25	553.34	72 (36)	-	-	-
	4	52.87	247.41	89	55.45	221.80	88
	8	51.87	131.02	72	126.00	101.03	64
	16	48.01	79.53	54	1,310.97	39.32	37
128	FEM	20.62	2,213.40	145 (64)	-	-	-
	4	58.00	553.85	138	97.61	501.92	139
	8	57.50	299.33	113	96.59	240.98	100
	16	55.55	152.32	82	774.17	103.17	61
256	FEM	20.81	8,853.58	292 (127)	-	-	-
	8	61.04	657.87	172	81.08	542.41	155
	16	60.06	326.03	125	438.22	-	99
	32	57.49	300.52	101	-	-	-

- Decrease of condition and iteration numbers by HFEM in 3D not satisfactory
- Further prehandling by Cholesky decomposition on coarse grid leads to excessive fill in

Candidates for prehandling - Generating system



minimum example of generating systemLinear system in 1D

- Also dependent on sequence of refined meshes
- Generating system consists of nodal bases on **all** levels and thus, mesh points that do not exclusively belong to finest mesh are counted repeatedly
- Transformation matrix S is rectangular, $\tilde{A} = S^{\mathsf{T}} A S$ is symm. pos. semi-definite (sufficient for convergence of conjugate gradient 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 ($^4/_3$ in 2D)
- Jacobi preconditioner corresponds to BPX $\Rightarrow \operatorname{cond}(\tilde{A}) = \frac{\lambda_{\max}}{\lambda_{\min,>0}} = \mathcal{O}(1)$
- Gauss–Seidel-type iterative methods correspond to MG and SSOR precond. to MGCG

Generating systems in 2D & 3D – matrix density and iteration numbers

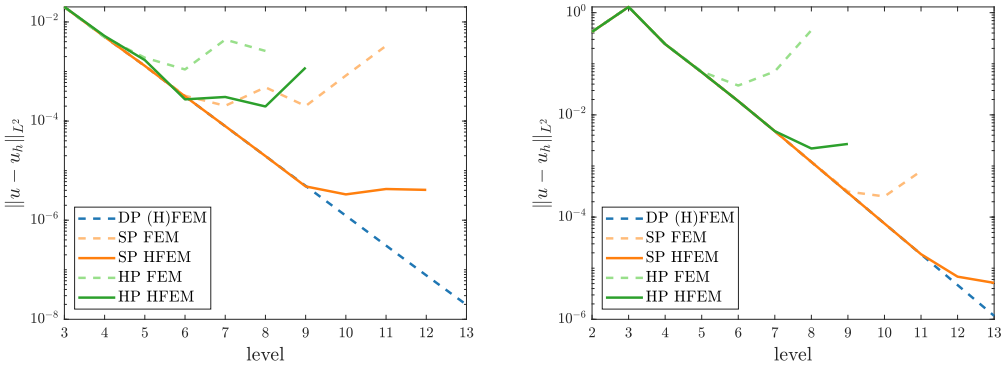
Results in 2D on unit square (Q1):

$\frac{1}{h}$	$\frac{1}{h_0}$	$\frac{\#\text{rows}(\tilde{A})}{N}$	$\frac{\text{NNZ}}{N}$	#iter (ichol)
1024	FEM	1.00	8.99	1,143 (395)
	16	1.33	40.00	31
	32	1.33	39.39	54
	64	1.33	38.07	96
2048	FEM	1.00	8.99	2,307 (771)
	32	1.33	40.29	57
	64	1.33	39.62	102
4096	FEM	1.00	9.00	4,658 (1,356)
	32	1.33	40.78	59
	64	1.33	40.44	107
	128	1.33	39.73	203

Results in 3D on unit cube (Q1):

$\frac{1}{h}$	$\frac{1}{h_0}$	$\frac{\#\text{rows}(\tilde{A})}{N}$	$\frac{\text{NNZ}}{N}$	#iter (ichol)
64	FEM	1.00	20.25	72 (37)
	4	1.13	74.14	15
	8	1.13	72.23	17
	16	1.13	65.43	23
128	FEM	1.00	20.62	145 (65)
	4	1.14	82.32	17
	8	1.14	81.35	19
	16	1.14	77.84	25
256	FEM	1.00	20.81	292 (128)
	8	1.14	87.13	20
	16	1.14	85.34	27
	32	1.14	80.80	47

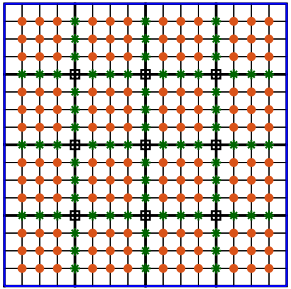
The effect of prehandling on accuracy in 2D



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 in 2D

- 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
- Subdivide nodes into:
  - $\mathcal{C}$: **C**oarse mesh nodes
  - $\mathcal{E}$: Nodes on **E**dges of the coarse mesh
  - $\mathcal{I}$: Nodes in the **I**nterior of the coarse mesh cells (cell by cell in same order)
- $Ax = b$ system after HFEM prehandling
- $A_{\mathcal{CC}} = I$ due to Cholesky on coarse mesh
- $A_{\mathcal{II}} =: C$ decomposes into $\#\text{macro cells} = \mathcal{O}(N^{1/2})$ independent blocks C_i
- Blocks are equal if corresponding to similar cells
- Only C grows like N $(= \#\mathrm{Dof})$



The direct method in 2D

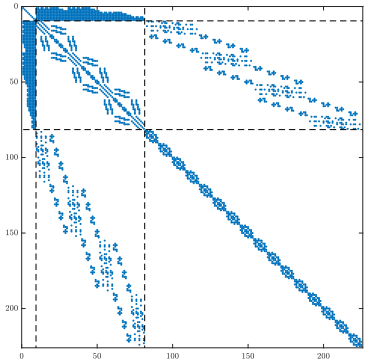
- In special cases (pure Poisson problem in 2D with Q1 on rectangular mesh or P1): No coupling between coarse mesh nodes and those inside macro cells, i.e., $A_{\mathcal{CI}} = 0$
- Matrix form:

$$\begin{pmatrix} I & A_{\mathcal{CE}} & 0 \\ A_{\mathcal{CE}}^{\mathsf{T}} & A_{\mathcal{EE}} & A_{\mathcal{EI}} \\ 0 & A_{\mathcal{EI}}^{\mathsf{T}} & C \end{pmatrix} \begin{pmatrix} x_{\mathcal{C}} \\ x_{\mathcal{E}} \\ x_{\mathcal{I}} \end{pmatrix} = \begin{pmatrix} b_{\mathcal{C}} \\ b_{\mathcal{E}} \\ b_{\mathcal{I}} \end{pmatrix}$$

- A **double Schur complement** yields:

Direct method

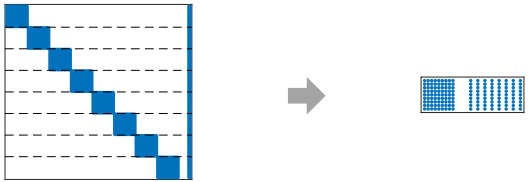
0) Compute inverse of
$$\Pi = A_{\mathcal{EE}} - A_{\mathcal{EI}} C^{-1} A_{\mathcal{EI}}^{\mathsf{T}} - A_{\mathcal{CE}} A_{\mathcal{CE}}^{\mathsf{T}}$$
1) $x_{\mathcal{E}} = \Pi^{-1} \left(b_{\mathcal{E}} - A_{\mathcal{CE}} b_{\mathcal{C}} - A_{\mathcal{EI}} C^{-1} b_{\mathcal{I}} \right)$
2) $x_{\mathcal{C}} = b_{\mathcal{C}} - A_{\mathcal{CE}} x_{\mathcal{E}}$
3) $x_{\mathcal{I}} = C^{-1} \left(b_{\mathcal{I}} - A_{\mathcal{EI}}^{\mathsf{T}} x_{\mathcal{E}} \right)$



- Matrices Π and C_i are well-conditioned (<50 on unit square with Q1; SP, HP possible)

Parenthesis: How to implement 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 transforming into dense matrix product (also for 1 RHS)



- Complexity $\mathcal{O}\left(N^{3/2}\right)$ because $\mathcal{O}(N^{1/2})$ RHS but very fast calculation by TC almost at peak performance
- **Current investigation**: Can different C_i be efficiently transformed into one as part of pre- and post-processing?

Direct method – recap

- Choice of $h_0 \approx \sqrt{h}$ as a good compromise yields complexity of $\approx 12N^{3/2}$
- Performance results on A100 on unit square in SP for many RHS:
  - up to 60 TFlop/s
  - 5,000 to 15,000 MDof/s (millions of unknowns solved per second)
- More arithmetic work ($\times 12$), but still much faster than standard MG solver on x64 AMD CPU (8 MDof/s for many rhs)
- Essentially equal results on unstructured coarse mesh ("flow around a square") with 3 distinct matrices C_i

Direct method – conclusion

Advantages of the direct method

- Highly performant because most operations are dense matrix multiplications also on unstructured coarse meshes (but higher memory consumption, depends on $\#C_i$)
- Almost no sensitivity to anisotropic meshes

Limitations of the direct method

- High storage requirement of $\mathcal{O}\left(N^{\frac{3}{2}}\right)$ due to Π^{-1}
- Limit of fine mesh size in our tests: $h = \frac{1}{1024}$ (on one GPU)
- Hardly applicable to 3D because storage requirement of $\mathcal{O}\left(N^{\frac{5}{3}}\right)$ and higher condition number with HFEM
- Requirement for simple form of the direct method: No coupling between nodes in $\mathcal{C}$ and $\mathcal{I}$ (coarse mesh nodes and nodes in interior of macro cells) only for Poisson's equation on P1 and rectangular Q1 meshes

- **We need a less memory consuming, more versatile variant, also working in 3D, that is still capable of exploiting Tensor Cores**

The semi-iterative method in 2D

- Again $Ax = b$ system after prehandling with HFEM + Cholesky decomposition on coarse mesh; now $A_{\mathcal{CI}} \neq 0$ possible
- **J**oin coarse mesh nodes and those on edges: $\mathcal{J} = \mathcal{C} \cup \mathcal{E} \rightarrow 2 \times 2$ block matrix

$$\begin{pmatrix} A_{\mathcal{JJ}} & A_{\mathcal{JI}} \\ A_{\mathcal{JI}}^{\mathsf{T}} & C \end{pmatrix} \begin{pmatrix} x_{\mathcal{J}} \\ x_{\mathcal{I}} \end{pmatrix} = \begin{pmatrix} b_{\mathcal{J}} \\ b_{\mathcal{I}} \end{pmatrix} \quad \Leftrightarrow: \quad \begin{pmatrix} A_1 & B \\ B^{\mathsf{T}} & C \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

- Applying a **single Schur complement** yields:

Semi-iterative method

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

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

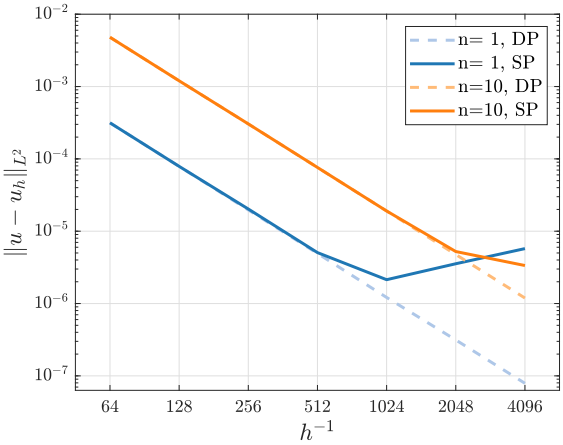
- $\hat{A}$ can be computed explicitly in 2D without excessive storage space ($\mathcal{O}(N^{1+\varepsilon})$) and is well-conditioned
- Majority of operations are still dense due to C^{-1} and $|\mathcal{J}| \ll |\mathcal{I}|$

The semi-iterative method in 3D

- In 3D: Same approach with slight differences
- $Ax = b$ system w.r.t. generating system because HFEM would lead to high condition numbers of $\hat{A}$ in 3D
- All basis functions on coarser levels are included in generating system $\rightarrow$ indices $\mathcal{I}$ can be chosen such that C_i are HFEM matrices (particularly invertible)
- All remaining indices (some of which correspond to the same mesh points) are stored in $\mathcal{J}$
- The semi-iterative algorithm resulting from Schur complement is the same as in 2D with HFEM
- $\hat{A}$ is positive semi-definite and has a lower generalized condition number
- Saving $\hat{A}$ explicitly would require a lot of storage $\rightarrow$ apply $\hat{A}$ implicitly as $A_1 - BC^{-1}B^{\mathsf{T}}$ in each iteration

Accuracy of the semi-iterative method

- Accuracy depends on smoothness of solution
- 2D: Loss of accuracy only for very smooth solution on high refinement levels
- 3D: No difference between DP and SP in considered range of mesh sizes



L^2-errors with semi-iterative method in 2D in DP and SP for differently smooth solutions

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}, D 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 favored

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:** $1 \times \hat{A}$ (3D: $A_1, B^{\mathsf{T}}, C^{-1}, B$) + 2 dot products + 3 axpy per iteration
  - Intermediate step: $1 \times B^{\mathsf{T}}$
  - **Direct Step:** $1 \times C^{-1}$

- Number of unknowns u solved iteratively: $\sim 2N^{3/4}$ in 2D, $\sim 3N^{5/6}$ in 3D
- Entire method in SP on A100
- Majority of the work: Dense matrix operations; Small part: sparse $\times$ dense and BLAS1
- **Own benchmark results on A100 in SP in GFlop/s** (given many RHS):

$\hat{A} \times$ dense	axpy	dot product	dense $\times$ dense	$B \times$ dense
2,375	227	321	150,000	1,200

Performance estimate

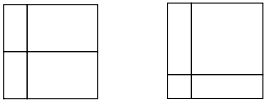
$\frac{1}{h}$	$\frac{1}{h_0}$	#iter	$\mathrm{cond}(C_i)$	$\frac{\mathrm{Flop}}{N}$ dense	$\frac{\mathrm{Flop}}{N}$ remainder	share dense	GFlop/s	MDof/s
1024	16	30	24	15,414	913	94.4%	31,445	**1,926**
	32	24	17	3,615	1,217	74.8%	8,206	1,698
2048	32	28	24	15,399	1,085	93.4%	27,919	**1,694**
	64	23	17	3,611	1,881	65.8%	6,333	1,153
4096	32	31	32	63,543	1,011	98.4%	74,476	1,154
	64	25	24	15,391	1,374	91.8%	23,636	**1,410**

$\frac{1}{h}$	$\frac{1}{h_0}$	#iter	$\mathrm{cond}(C_i)$	$\frac{\mathrm{Flop}}{N}$ dense	$\frac{\mathrm{Flop}}{N}$ remainder	share dense	GFlop/s	MDof/s
128	4	8	54	554,586	751	99.9%	121,307	218
	8	11	23	74,025	1,315	98.3%	36,831	**489**
	16	18	9	9,410	2,578	78.5%	3,799	317
256	8	11	54	712,513	1,131	99.8%	117,465	165
	16	18	23	112,550	2,295	98.0%	33,242	**289**
	32	35	9	17,205	5,294	76.5%	3,487	155

- Comparative result with **optimized MG** in C++-based FE software package (FEAT3) on AMD CPU in DP in **2D**: **8 MDof/s**

The semi-iterative method on anisotropic meshes in 2D

Anisotropy types in 2D: left x direction, right x,y direction



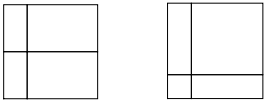
- 2 types of anisotropy on unit square in 2D
- Consider condition of C_i (averaged) and $\hat{A}$ and iteration numbers for increasing aspect ratios

			$1/2$			$1/4$			$1/16$		
	L	L_0	$\mathrm{cond}(C_i)$	$\mathrm{cond}(\hat{A})$	#it	$\mathrm{cond}(C_i)$	$\mathrm{cond}(\hat{A})$	#it	$\mathrm{cond}(C_i)$	$\mathrm{cond}(\hat{A})$	#it
x	10	4	23.9	28.3	28	35.0	39.2	35	108.1	111.8	48
		5	16.9	22.0	23	24.7	31.5	30	89.3	89.0	43
		6	11.1	16.4	18	16.2	24.3	24	67.2	70.5	37
x, y	10	4	23.9	28.3	28	40.2	52.3	40	303.4	197.7	71
		5	16.9	22.0	23	28.4	41.9	33	243.9	202.0	66
		6	11.1	16.4	18	18.7	32.6	28	146.7	199.0	61

- Max. $3\times$ iterations with conjugate gradient method on very anisotropic meshes
- Much higher effort with MG expected (more iterations and more expensive smoother than for example Jacobi necessary)

Preconditioning for the semi-iterative method in 2D

Anisotropy types in 2D: left x direction, right x,y direction



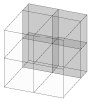
- Consider iteration numbers with **plain conjugate gradients** (pcg) vs. **incomplete Cholesky** decomposition (ichol) as preconditioner

			$^1/_2$		$^1/_4$		$^1/_{16}$	
	L	L_0	pcg	ichol	pcg	ichol	pcg	ichol
x	10	4	28	11	35	13	48	14
		5	23	8	30	10	43	14
		6	18	6	24	8	37	10
x, y	10	4	28	11	40	15	71	20
		5	23	8	33	12	66	18
		6	18	6	28	10	61	14

- Iteration numbers can be significantly reduced by means of preconditioning

The semi-iterative method on anisotropic meshes in 3D

			$1/2$		$1/4$		$1/16$	
	L	L_0	$\mathrm{cond}(C_i)$	#it	$\mathrm{cond}(C_i)$	#it	$\mathrm{cond}(C_i)$	#it
x	8	3	54.0	11	94.3	18	435.3	27
		4	22.5	18	39.6	35	220.2	44
		5	8.7	35	13.8	71	35.3	90
x, y	8	3	54.0	11	105.6	20	537.9	39
		4	22.5	18	45.1	38	281.9	52
		5	8.7	35	14.6	77	39.1	98
x, y, z	8	3	54.0	11	115.0	21	913.1	44
		4	22.5	18	51.0	41	448.9	58
		5	8.7	35	13.9	84	51.6	94



- Preconditioning without explicit matrix difficult, but might be possible

Conclusion and outlook

Conclusion

- It is possible to exploit Lower-Precision Accelerator Hardware for PDE computing
- A semi-iterative approach and the use of generating systems allows this for more general problems in 2D and 3D (compared to the fully direct variant)

Outlook

- Implementation of the complete semi-iterative method on GPU
- Deeper analysis of suitable preconditioners for the iterative step and initial guesses for the solution vector to reduce number of iterations
- Testing semi-iterative Method for other FE spaces and PDEs (Convection-Diffusion-Reaction)
- Investigate the feasibility of transforming different matrices C_i into one

References

Literature

- 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.13,16,17,18: Very fast finite element Poisson solvers on lower precision accelerator hardware: A proof of concept study for Nvidia Tesla V100, IJHPCA 36(4)