No. 671 **January 2024**

Fast Semi-Iterative Finite Element Poisson Solvers for Tensor Core GPUs Based on Prehandling

D. Ruda, S. Turek, D. Ribbrock

ISSN: 2190-1767

Fast Semi-Iterative Finite Element Poisson Solvers for Tensor Core GPUs Based on Prehandling

Dustin Ruda[0000−0003−4252−3099],
Stefan Turek[0000−0002−9740−6087] and
Dirk Ribbrock[0000−0002−2932−0513]

Abstract The impetus for the research presented in this work is provided by recent developments in the field of GPU computing. Nvidia GPUs that are equipped with Tensor Cores, such as the A100 or the latest H100, promise an immense computing power of 156 and 495 TFLOPS, respectively, but only for dense matrix operations carried out in single precision (with even higher rates in half precision), since this serves their actual purpose of accelerating AI training. It is shown that this performance can also be exploited to a large extent in the domain of matrix-based finite element methods for solving PDEs, if specially tailored, hardware-oriented methods are used. Such methods need to preserve sufficient accuracy, even if single precision is used, and mostly consist of dense matrix operations. A semi-iterative method for solving Poisson's equation in 2D and 3D based on prehandling, i.e., explicit preconditioning, by means of hierarchical finite elements or generating systems, that satisfies these requirements, is derived and analyzed. Actual benchmark results on an H100 allow the determination of optimal solver configurations in terms of performance, which ultimately exceeds that of a standard geometric multigrid solver on CPU.

1 Introduction

First, the hardware under consideration and the difficulties that arise when it is intended to be used for the numerical solution of PDEs using matrix-based finite elements are described. We focus on a particular type of Nvidia GPUs that are

Dustin Ruda, Stefan Turek, Dirk Ribbrock
TU Dortmund University, Department of Mathematics, Chair of Applied Mathematics and Numerics (LS3), Vogelpothsweg 87, 44227 Dortmund, Germany
e-mail: dustin.ruda@math.tu-dortmund.de
stefan.turek@math.tu-dortmund.de
dirk.ribbrock@math.tu-dortmund.de

Table 1 Peak rates in TFLOPS according to manufacturer specifications depending on precision (double (DP), single (SP), half precision (HP)) and use of Tensor Cores (TC).

model	DP	DP TC	SP	SP TC	HP	HP TC
V100	7.8	-	15.7	-	31.4	125
A100	9.7	19.5	19.5	156	78.0	312
H100	34.0	67.0	67.0	495	n/a	990

equipped with Tensor Cores. Their purpose is to accelerate AI training, which is why they are designed to perform the most expensive component, dense matrix multiplications, at very high speed, especially in lower (single or half) precision, since double precision is not necessary in this context. The three main representatives of this hardware are the V100 (2017), A100 (2020), and H100 (2023), with their peak performance in terms of TFLOPS listed in Table 1[1]. Their importance to high-performance computing becomes clear when the current TOP500 list (November 2023[2]) is considered. Six of the top ten systems are supported by V100, A100 or H100 GPUs. Two others include AMD Instinct MI250X accelerators, which draw their computing power from Matrix Cores, a technology similar to Tensor Cores.

Of course, it would be desirable to profitably use the high performance of Tensor Cores for PDE computing, but two main difficulties arise: The matrices resulting from finite element discretizations are sparse and therefore unsuitable to exploit Tensor Cores, and the use of lower precision leads to a loss of accuracy of the solution if the underlying stiffness matrix is ill-conditioned.

The latter is the case for Poisson's equation, to which we restrict ourselves for the scope of this work, since the spectral condition number of the resulting stiffness matrix with respect to a standard, nodal basis is $\kappa(A_h) = O(h^{-2})$, where h denotes the mesh size. According to perturbation theory of linear systems, the condition number is the amplification factor of the data error. Neglecting all other possible sources of error, the data error is given as the machine epsilon of the respective precision $\varepsilon_{\text{prec}}$. Hence, the computational error can be approximated as $\kappa(A_h)\varepsilon_{\text{prec}} \approx h^{-2}\varepsilon_{\text{prec}}$ and increases with increasing mesh resolution. In contrast, the discretization error, given as $O(h^2)$ for (bi-)linear finite elements, decreases. The actual error decreases only up to the point where the computational error becomes dominant. If a precision lower than double is used, this critical mesh size is already reached at a coarse stage of the mesh. To circumvent this and to allow the use of lower precision by reducing the computational error, we have introduced the concept of prehandling, an explicit way of preconditioning, or in other words a transformation of a linear system into an equivalent form with more advantageous properties. For a detailed description, see [1]. Let $A, \tilde{A} \in \mathbb{R}^{N\times N}$ be the matrix before and after prehandling, respectively.

[1] see https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf, https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet for datasheets

[2] see https://www.top500.org/lists/top500/2023/11/

For this approach to be sensible, three conditions must be met: A significant reduction of the condition number ($\kappa(\tilde{A}) \ll \kappa(A)$), preservation of sparsity and a fast transformation of the linear system (i.e., $N \log(N)$ operations).

One method whose suitability for prehandling in the 2D case has been demonstrated in previous research is hierarchical finite elements. More recently, the use of so-called generating systems has also proven effective in the 3D case.

1.1 Hierarchical Finite Elements

The hierarchical finite element method (HFEM) has been known since the 1980s and was mainly analyzed by H. Yserentant in [2]. It was shown that the stiffness matrix associated with Poisson's equation in 2D with respect to a hierarchical basis has a condition number $\kappa(\tilde{A}) = O\left((\log 1/h)^2\right)$. The method was proposed as a preconditioner for the conjugate gradient (CG) method. However, an explicit computation of the transformed system results in a matrix, that is slightly denser but still sparse. The construction of a hierarchical basis requires a sequence of uniformly refined meshes starting from a coarse mesh, on which a nodal basis is fixed. For each level of refinement, a nodal basis corresponding to the newly added nodes of that level is added. If the original linear system is $Ax = b$, we obtain the system with respect to a hierarchical basis $\tilde{A}\tilde{x} = \tilde{b}$ via $\tilde{A} = S^\mathsf{T} A S$ and $\tilde{b} = S^\mathsf{T} b$. The solution of the original problem is $x = S\tilde{x}$. The transformation matrix S is given as a product $S_j \dots S_1$ for j steps of refinement, where each factor S_k is a square version of the interpolation (or prolongation) operator from level $k-1$ to k. We use the Cholesky decomposition of $\tilde{A}$ restricted to the rows and columns corresponding to the coarse mesh for further prehandling to obtain even lower condition numbers while keeping the numerical effort and storage consumption low, since the number of coarse mesh nodes is usually small. For details, see [1, 2]. Indeed, a higher accuracy can be obtained by means of prehandling with hierarchical finite elements in 2D if the system is solved in single or half precision [1]. The results in 3D concerning the condition number of matrices with respect to hierarchical bases are less satisfactory because it is $\kappa(\tilde{A}) = O(h^{-1})$, as shown in [3].

1.2 Generating Systems

The use of generating systems, an approach similar to HFEM, was introduced by M. Griebel in the 1990s in [4, 5]. A generating system is again based on a hierarchy of gradually refined meshes and it is given as the union of the nodal bases on all levels. It is therefore not a basis and contains more functions than the nodal basis on the fine mesh. The stiffness matrix with respect to a generating system E is symmetric and positive semi-definite and its number of rows, or the size of the generating system N_E, grows by a factor of 2 in 1D, $4/3$ in 2D and $8/7$ in 3D compared to N. The

properties of E are sufficient for the convergence of the CG method to a non-unique solution. The great advantage of using generating systems is that preconditioners, which are otherwise difficult to implement and unsuitable for prehandling, have a very simple counterpart in this context. For instance, the BPX preconditioner of Bramble, Pasciak and Xu [6] corresponds to the Jacobi preconditioner if the representation with respect to a generating system is used. The symmetric Jacobi preconditioner can easily be used for prehandling by computing $E^J = \operatorname{diag}(E)^{-1/2} E \operatorname{diag}(E)^{-1/2}$ without any fill-in. Due to the equivalence with the BPX preconditioner, we have $\kappa(E^J) = O(1)$, which is independent of the dimension.

Analogous to HFEM, the transformation of the original linear system $Ax = b$ to the representation with respect to a generating system (comprising j levels) $Ex^E = b^E$ is possible by means of a transformation matrix $S = S_j \dots S_1$, that is now rectangular ($S \in \mathbb{R}^{N \times N_E}$). In order to describe the assembly of the factors S_k, we set some notations: Let n_k be the number of interior grid points on level k, so that $n_j = N$, and let $N_{E,k} = n_k + \dots + n_j$, so that $N_{E,0} = N_E$ and $N_{E,j} = N$, with the convention $N_{E,k} = 0$ for $k > j$. The S_k are constructed as 2×3 block matrices consisting of identity and zero blocks as well as interpolation (prolongation) operators $P_{k-1}^k \in \mathbb{R}^{n_k \times n_{k-1}}$ from level $k-1$ to k as

$$S_k = \begin{pmatrix} P_{k-1}^k & I_{n_k} & 0_{n_k \times N_{E,k+1}} \\ 0_{N_{E,k+1} \times n_{k-1}} & 0_{N_{E,k+1} \times n_k} & I_{N_{E,k+1}} \end{pmatrix} \in \mathbb{R}^{N_{E,k} \times N_{E,k-1}}.$$

Originally, S was intended to be used in its product form as an implicit preconditioner, but the explicit computation of E is possible without excessive storage consumption.

2 Derivation of a Hardware-Oriented Semi-Iterative Solver

In previous research, we provided a proof of concept that Tensor Cores can be exploited when solving Poisson's equation in 2D by means of a fully direct, HFEM-based method described in [7] achieving 60,000 TFLOPS in half and 46,000 TFLOPS in single precision on the V100 for multiple right-hand sides while preserving sufficient accuracy. Since this measure alone is not meaningful when comparing the method to a standard geometric multigrid solver with different complexity, we also consider the measure of millions of unknowns solved per second (MDof/s), in which the direct method is also significantly superior due to the benefit of Tensor Cores despite higher complexity. It has been shown in [8] that this method works equally well when extended to semi-structured grids.

The assumption of multiple right-hand sides or a dense matrix as right-hand side is advantageous with regard to the use of Tensor Cores and it has practical relevance, for example, in the context of global-in-time solvers for the Navier–Stokes equation [9], which require the solution of the pressure Poisson problem for all time steps at once with respect to the same matrix, given a fixed mesh. However, the direct method is limited to triangular meshes with linear finite elements and orthogonal

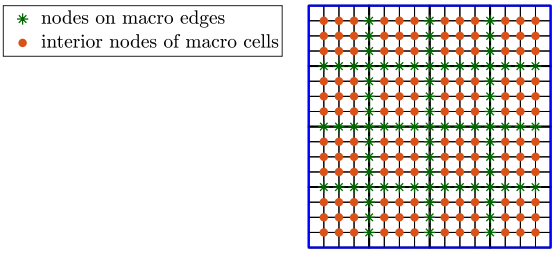

Fig. 1 Illustration of the two types of nodes $\mathcal{I}(\bullet)$ and $\mathcal{J}(*)$ on a uniformly refined coarse grid (bold lines) for the unit square with $h_0 = 1/4$ and $h = 1/16$, yielding 16 coarse mesh cells containing 9 nodes each, and Q_1 finite elements.

quadrilateral meshes with bilinear finite elements, and to the 2D case due to high storage consumption of $O\left(N^{3/2}\right)$ (the analogous method in 3D would require the storage of $O\left(N^{5/3}\right)$ non-zeros).

The aim is to derive a more versatile solver that also works in 3D, still consisting mainly of dense operations, whereby a higher share of sparse operations is accepted to reduce the storage cost. After a detailed analysis of all possible resulting variants – HFEM and generating systems are applicable in both 2D and 3D – it turned out that the more efficient and, in terms of conditioning, advantageous option in 2D is the one based on HFEM, and in 3D is the one based on generating systems, and therefore only these will be treated in the following.

First, we consider the 2D case. For simplicity, we denote the linear system resulting from prehandling via HFEM (and later generating systems) as $Ax = b$. As a first step, the unknowns are divided into two types: $\mathcal{I}$ is the set of all unknowns in the interior of the cells given by the coarse mesh, and $\mathcal{J}$ is the complement, i.e., those on the edges of the coarse mesh as shown in Fig. 1. We assume that $\mathcal{I}$ is ordered cell by cell and in the same geometric order within each cell. Note that for a fixed fine mesh, the cardinalities of both sets can be influenced by adjusting the refinement level of the coarse mesh. Renumbering the linear system accordingly and making some notational simplifications yields

$$\begin{pmatrix} A_{\mathcal{J}\mathcal{J}} & A_{\mathcal{J}\mathcal{I}} \\ A_{\mathcal{J}\mathcal{I}}^{\mathsf{T}} & A_{\mathcal{I}\mathcal{I}} \end{pmatrix} \begin{pmatrix} x_{\mathcal{J}} \\ x_{\mathcal{I}} \end{pmatrix} = \begin{pmatrix} b_{\mathcal{J}} \\ b_{\mathcal{I}} \end{pmatrix} \quad \Leftrightarrow: \quad \begin{pmatrix} A_1 & B \\ B^{\mathsf{T}} & C \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \tag{1}$$

Due to the numbering of $\mathcal{I}$, the matrix C decomposes into independent blocks C_i. The number of submatrices is equal to the number of coarse mesh cells and their row number is equal to the number of nodes in the interior of these cells. Furthermore, the C_i are equal, if they correspond to similar coarse mesh cells, and well-conditioned because they are HFEM matrices. These properties allow an explicit computation of the inverse C^{-1}, since only the submatrices C_i need to be inverted and saved (once for each group of similar cells), and an application of C^{-1} is transformed into small, dense matrix products, which can be performed very fast by means of Tensor Cores. The semi-iterative method is derived from system (1) by eliminating u applying a Schur complement approach. We obtain the algorithm

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

The computation of $\hat{A}$ and the transformation of the obtained solution components back to the representation with respect to the nodal basis by S as part of post-processing are performed in double precision. The iterative step 1 and the direct step 2 are carried out in single precision. The main part of the computational cost of the stated method consists of dense operations, complemented by a multiplication of the sparse matrix $\hat{A}$ with a dense matrix, axpy, and a dot product for each iteration in step 1, and intermediate steps consisting mainly of multiplications with B and B^T. The final composition of the method, and thus the complexity and the potential for Tensor Cores depends on the choice of the coarse mesh and is discussed in the following section.

Adapting the method to the 3D case requires some adjustments. In order to obtain the same block structure of the matrix C, a partitioning of the unknowns analogous to the 2D case is performed: $\mathcal{I}$ includes the nodes in the interior of the three-dimensional coarse mesh cells. For the nodes that do not belong exclusively to the fine mesh and therefore appear repeatedly in the generating system, only one index has to be chosen as an element of $\mathcal{I}$ in order to obtain invertible submatrices C_i. This could be done by simply choosing the indices corresponding to the fine mesh, which would lead to C_i being standard FEM matrices. However, a lower condition number is obtained if the indices are chosen in such a way that the C_i are related to a hierarchical basis. The remaining ones plus those on the edges and faces of the coarse mesh are contained in $\mathcal{J}$. The semi-iterative method then has the same form as given above. One difference is that the matrix $\hat{A}$ is positive semi-definite and that its explicit computation would result in a high storage consumption. Therefore, in the 3D case, the implicit form $A_1 - BC^{-1}B^\mathsf{T}$ is preferred, which can be used to compute the products arising in the CG method.

3 Numerical Results

The derived method is now analyzed in terms of storage requirements, expected performance on the H100, and the optimal coarse mesh size using the toy problem of the uniformly and equidistantly refined unit square or cube. For different combinations of fine and coarse mesh sizes (h and h_0) in 2D and 3D, the iteration numbers and the relevant properties of all matrices are known. Additional benchmark results on an H100 for the GFLOPS of all occurring operations in single precision (assuming many right-hand sides) allow a reliable estimate of the overall performance.

The main results for the 2D and 3D case are shown in Table 2 and Table 3, respectively. In each case, the first four rows show the storage requirement in terms of non-zero entries (NNZ) relative to N of the matrices used in the semi-iterative method and in total. Note that the matrix C_i^{-1} is expanded by zeros so that its number of rows and columns divides 16, since Tensor Cores work particularly fast in this case. Except for the case $(h, h_0) = (1/128, 1/4)$ in 3D, the total storage requirement is practically independent of h and moderate, compared to $9N$ in 2D and $27N$ in 3D, considering that the matrices of the semi-iterative method are stored in single

Table 2 Properties, benchmark results and performance model of the semi-iterative method in SP on H100 in 2D.

	$h = \frac{1}{1024}$		$h = \frac{1}{2048}$		$h = \frac{1}{4096}$	
	$h_0 = \frac{1}{16}$	$h_0 = \frac{1}{32}$	$h_0 = \frac{1}{32}$	$h_0 = \frac{1}{64}$	$h_0 = \frac{1}{32}$	$h_0 = \frac{1}{64}$
$\text{NNZ}(\hat{A})/N$	15.01	24.92	19.15	40.46	16.20	27.24
$\text{NNZ}(C_i^{-1})/N$	15.17	0.91	3.79	0.23	15.54	0.95
$\text{NNZ}(B)/N$	1.02	1.59	1.05	1.62	0.66	1.07
Total NNZ/N	31.19	27.42	23.99	42.30	32.39	29.25
iterations	30	24	28	23	31	25
share dense	94.44%	75.38%	93.46%	66.43%	98.43%	91.86%
GFLOPS C^{-1}	213,504	164,504	220,525	163,620	403,598	230,973
GFLOPS $\hat{A}$	3,743	3,566	3,115	2,404	3,204	2,479
GFLOPS B	2,107	2,083	2,138	2,070	2,138*	2,070*
GFLOPS B^{T}	698	948	713	948	713*	948*
Total GFLOPS	47,028	12,010	36,877	6,600	130,867	26,010
Total MDof/s	2,860	2,428	2,221	1,177	2,024	1,541

*estimate

Table 3 Properties, benchmark results and performance model of the semi-iterative method in SP on H100 in 3D.

	$h = \frac{1}{128}$			$h = \frac{1}{256}$		
	$h_0 = \frac{1}{4}$	$h_0 = \frac{1}{8}$	$h_0 = \frac{1}{16}$	$h_0 = \frac{1}{8}$	$h_0 = \frac{1}{16}$	$h_0 = \frac{1}{32}$
$\text{NNZ}(A_1)/N$	11.32	22.06	37.05	14.24	24.92	39.50
$\text{NNZ}(C_i^{-1})/N$	433.30	5.56	0.06	53.53	0.69	0.01
$\text{NNZ}(B)/N$	15.36	16.61	15.30	16.54	17.66	16.44
Total NNZ/N	459.98	44.23	52.41	84.31	43.27	55.95
iterations	8	11	18	11	18	35
share dense	99.86%	98.25%	79.30%	99.84%	98.00%	77.33%
GFLOPS C^{-1}	309,278	216,213	93,052	389,710	214,190	83,020
GFLOPS A_1	3,063	3,174	3,156	2,877	2,966	3,026
GFLOPS B	2,807	2,930	2,850	2,550	2,620	2,563
GFLOPS B^{T}	3,656	3,812	3,802	3,690	3,768	3,790
Total GFLOPS	265,263	87,605	10,520	310,909	78,200	9,374
Total MDof/s	478	1,162	842	436	680	400

precision. The number of unpreconditioned CG iterations of step 1 is also small. The row below shows the share of dense operations, i.e., multiplications with C^{-1}, in the total numerical cost of the method in FLOP, which is strongly influenced by the choice of h_0. The benchmark results of the most important operations performed with the actual matrices on an H100 follow. As expected, the highest performance of

83 up to 403 TFLOPS is observed for the dense part. The sparse times dense matrix multiplications are performed at 0.7 to 3.8 TFLOPS. The cost of matrix additions (144 GFLOPS) as well as axpy operations (287 GFLOPS) and dot products (479 GFLOPS) occurring in the CG method are almost negligible in comparison.

Knowing the numerical effort for each operation and the corresponding performance, it is possible to calculate the overall performance of the solver in terms of GFLOPS and MDof/s, as shown in the last two rows of the tables and determine the optimal coarse mesh size h_0. In the 2D examples, for each h, the value of h_0 yielding the highest overall performance in terms of GFLOPS is also associated with the most unknowns solved per second, namely for $(h, h_0) \in \{(1/1024, 1/16), (1/2048, 1/32), (1/4096, 1/32)\}$. In 3D, this is not the case. While the highest performance is obtained with the choices $(h, h_0) \in \{(1/128, 1/4), (1/256, 1/8)\}$, the highest value of MDof/s is obtained with one additional refinement step of the coarse mesh. This is because the effort for the very fast multiplications with the matrices C_i^{-1} increases, leading to high overall performance, but also to a higher complexity and thus to lower MDof rates. In 2D, this effect is not as strong due to the slower growth of the matrices C_i^{-1} as a function of h_0 (approx. $(h_0/h)^2$ in 2D instead of $(h_0/h)^3$ in 3D).

To better evaluate the results, it should be noted that the maximum value obtained with an optimized geometric multigrid solver in the C++-based finite element software package FEAT3[3] on CPU and in double precision, due to ill-conditioning without alternative, is approx. 8 MDof/s.

Finally, we want to examine the behavior of the solver if the underlying mesh does not consist of equal squares or cubes as previously assumed, but rectangles or cuboids with high aspect ratios. Commonly, the convergence behavior of iterative solvers like multigrid methods deteriorates in the case of such anisotropic meshes and sophisticated preconditioners are necessary. Again, we consider the unit square/cube, but in the first step of refinement, the central node is shifted along one or more coordinate axes by a constant distance. From then on, the mesh is refined uniformly. This yields the two (three) types of anisotropy in 2D (3D) depicted in Fig. 2 (Fig.3). The iteration numbers of the unpreconditioned CG method (stopping criterion: relative residual $< 10^{-6}$ (10^{-4}) in 2D (3D)) were determined and are shown in Table 4 for the 2D case and in Table 5 for the 3D case depending on the coordinates of the central node and the mesh size. Since there is no constant mesh size, we consider $\hat{h}$ and $\hat{h}_0$ denoting the reciprocal of the number of elements of the fine and coarse mesh, respectively, in one coordinate direction. Therefore the pairs of mesh sizes (h, h_0) in the isotropic and $(\hat{h}, \hat{h}_0)$ in the anisotropic case yield the same number of unknowns and sizes of $\hat{A}$, C_i, etc.

It can be seen that the iteration numbers increase with increasing anisotropy, but only moderately compared to the aspect and volume ratios of the elements, so that the performance of the solver is not severely restricted in this case, particularly as the direct part is not affected besides a higher number of different matrices C_i^{-1} that need to be stored.

[3] see https://www.mathematik.tu-dortmund.de/featflow/feat3

Fig. 2 Two types of anisotropy in 2D in directions x (left) and x,y (right).

Table 4 CG iterations on anisotropic meshes in 2D depending on coordinates of the central node.

		$\hat{h} = \frac{1}{1024}$		$\hat{h} = \frac{1}{2048}$		$\hat{h} = \frac{1}{4096}$	
x	y	$\hat{h}_0 = \frac{1}{16}$	$\hat{h}_0 = \frac{1}{32}$	$\hat{h}_0 = \frac{1}{32}$	$\hat{h}_0 = \frac{1}{64}$	$\hat{h}_0 = \frac{1}{32}$	$\hat{h}_0 = \frac{1}{64}$
1/2	1/2	28	23	27	21	30	22
1/4	1/2	35	30	33	27	36	30
1/8	1/2	40	34	37	31	40	34
1/16	1/2	48	43	46	40	50	42
1/4	1/4	40	33	37	31	41	35
1/8	1/8	50	43	48	41	54	44
1/16	1/16	71	66	71	65	76	70

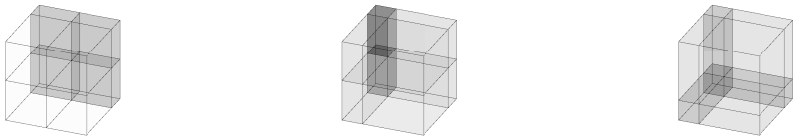

Fig. 3 Three types of anisotropy in 3D in directions x (left), x,y (center), and x,y,z (right).

Table 5 CG iterations on anisotropic meshes in 3D depending on coordinates of the central node.

			$\hat{h} = \frac{1}{128}$			$\hat{h} = \frac{1}{256}$		
x	y	z	$\hat{h}_0 = \frac{1}{4}$	$\hat{h}_0 = \frac{1}{8}$	$\hat{h}_0 = \frac{1}{16}$	$\hat{h}_0 = \frac{1}{8}$	$\hat{h}_0 = \frac{1}{16}$	$\hat{h}_0 = \frac{1}{32}$
1/2	1/2	1/2	8	11	18	11	18	35
1/4	1/2	1/2	11	18	34	18	35	71
1/8	1/2	1/2	16	21	42	21	43	87
1/16	1/2	1/2	21	28	44	27	44	90
1/4	1/4	1/2	14	20	38	20	38	77
1/8	1/8	1/2	24	30	45	27	46	94
1/16	1/16	1/2	32	42	54	39	52	98
1/4	1/4	1/4	15	21	41	21	41	84
1/8	1/8	1/8	27	33	47	31	47	96
1/16	1/16	1/16	41	52	64	44	58	94

4 Conclusion and Outlook

The aim was to develop a memory-efficient variant of the fully direct Poisson solver, which is suitable for the 3D case and still uses Tensor Cores in lower precision

profitably while maintaining sufficient accuracy. This is achieved by means of a hardware-oriented semi-iterative solver based on HFEM in 2D and generating systems in 3D, which is expected to compare favorably with a standard solver on CPU.

Tasks for future research include the implementation of the entire method on GPU, examining preconditioners and initial guesses for the solution to further lower the iteration numbers as well as the behavior of the method for other differential operators and meshes.

Acknowledgements This work was supported by the Federal Ministry of Education and Research (BMBF) through the project "StroemungsRaum" 16ME0706K, which is part of the initiative "Neue Methoden und Technologien für das Exascale-Höchstleistungsrechnen" (SCALEXA).

Comparative results have been created using the FEM software package FEAT3, whereby calculations have been carried out on the LiDO33 cluster at TU Dortmund University.

Evaluation access to an H100 GPU was kindly provided by Jülich Supercomputing Centre[4].

References

1. Ruda, D., Turek, S., Zajac, P., Ribbrock, D.: The Concept of Prehandling as Direct Preconditioning for Poisson-like Problems. In Vermolen, F.J., Vuik, C. (eds.) Numerical Mathematics and Advanced Applications ENUMATH 2019. Lecture Notes in Computational Science and Engineering, vol 139, pp. 1011–1019. Springer, Cham (2021) doi: 10.1007/978-3-030-55874-1_100
2. Yserentant, H.: On the multi-level splitting of finite element spaces. Numer. Math. **49**, 379–412 (1986) doi: 10.1007/BF01389538
3. Ong, M.E.G.: Hierarchical Basis Preconditioners in Three Dimensions. SIAM J. Sci. Comput. **18(2)**, 479–498 (1997) doi: 10.1137/S1064827594276539
4. Griebel, M.: Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen. In: Bock, H.G., Hackbusch, W., Rannacher, R. (eds.) Teubner Skripten zur Numerik. Vieweg+Teubner Verlag, Wiesbaden (1994) doi: 10.1007/978-3-322-89224-9
5. Griebel, M.: Multilevel Algorithms Considered as Iterative Methods on Semidefinite Systems. SIAM J.Sci.Comput. **15(3)**, 547–565 (1994) doi: 10.1137/0915036
6. Bramble, J.H., Pasciak, J.E., Xu, J.: Parallel Multilevel Preconditioners. Math. Comp. **55(191)**, 1–22 (1990) doi: 10.2307/2008789
7. Ruda, D., Turek, S., Ribbrock, D., Zajac, P.: Very fast finite element Poisson solvers on lower precision accelerator hardware: A proof of concept study for Nvidia Tesla V100. Int. J. High Perform. Comput. Appl. **36(4)**, 459–474 (2022) doi: 10.1177/10943420221084657
8. Ruda, D., Turek, S., Ribbrock, D., Zajac, P.: An extension of a very fast direct finite element Poisson solver on lower precision accelerator hardware towards semi-structured grids. In Kvamdal, T., et al. (eds.) ECCOMAS Congress 2022 – 8th European Congress on Computational Methods in Applied Sciences and Engineering (2022) doi: 10.23967/eccomas.2022.292
9. Lohmann, C., Turek, S.: On the Design of Global-in-Time Newton-Multigrid-Pressure Schur Complement Solvers for Incompressible Flow Problems. J. Math. Fluid Mech. **25(64)** (2023) doi: 10.1007/s00021-023-00807-6

[4] see https://www.fz-juelich.de/en/ias/jsc