International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Research Paper

The International Journal of High
Performance Computing Applications
2022, Vol. 36(4) 459–474
© The Author(s) 2022
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420221084657
journals.sagepub.com/home/hpc
SAGE

Very fast finite element Poisson solvers on lower precision accelerator hardware: A proof of concept study for Nvidia Tesla V100

Dustin Ruda, Stefan Turek, Dirk Ribbrock and Peter Zajac

Abstract

Recently, accelerator hardware in the form of graphics cards including Tensor Cores, specialized for AI, has significantly gained importance in the domain of high-performance computing. For example, NVIDIA's Tesla V100 promises a computing power of up to 125 TFLOP/s achieved by Tensor Cores, but only if half precision floating point format is used. We describe the difficulties and discrepancy between theoretical and actual computing power if one seeks to use such hardware for numerical simulations, that is, solving partial differential equations with a matrix-based finite element method, with numerical examples. If certain requirements, namely low condition numbers and many dense matrix operations, are met, the indicated high performance can be reached without an excessive loss of accuracy. A new method to solve linear systems arising from Poisson's equation in 2D that meets these requirements, based on "prehandling" by means of hierarchical finite elements and an additional Schur complement approach, is presented and analyzed. We provide numerical results illustrating the computational performance of this method and compare it to a commonly used (geometric) multigrid solver on standard hardware. It turns out that we can exploit nearly the full computational power of Tensor Cores and achieve a significant speed-up compared to the standard methodology without losing accuracy.

Keywords

Accelerator hardware, tensor core GPUs, NVIDIA V100, prehandling, hierarchical finite elements, Poisson's equation

1. Motivation and related work

Accelerator hardware, in particular current GPUs equipped with Tensor Cores tailored for AI applications, is an increasingly important component of state-of-the-art computer systems, which is used to boost their computing power. For instance, the NVIDIA Tesla V100 Tensor Core GPU, which we consider in this work, reaches up to 7.8 TFLOP/s in double precision and 125 TFLOP/s in half precision due to the usability of Tensor Cores according to the manufacturer specifications (NVIDIA, 2020). Of course, it is desirable to apply the V100 or similar GPUs in the context of numerical simulation, or more precisely, to solve linear systems resulting from the discretization of partial differential equations (PDEs), for example, via finite element methods (FEM), that are associated with continuum mechanics. But when trying to do so, one encounters the problems explained in more detail in the following two sections, namely the risk of a deteriorating error if low precision is used in conjunction with high condition numbers, and the presence of sparse matrices, for which Tensor Cores cannot be fully utilized. Consequently, the question arises whether it is possible to implement basic components of finite element simulations on modern

Institute for Applied Mathematics (LS III), TU Dortmund University, Dortmund, Germany

Corresponding author:
Dustin Ruda, Institute for Applied Mathematics (LS III), TU Dortmund University, Vogelpothsweg 87, D-44227 Dortmund 44227, Germany.
Email: dustin.ruda@math.tu-dortmund.de

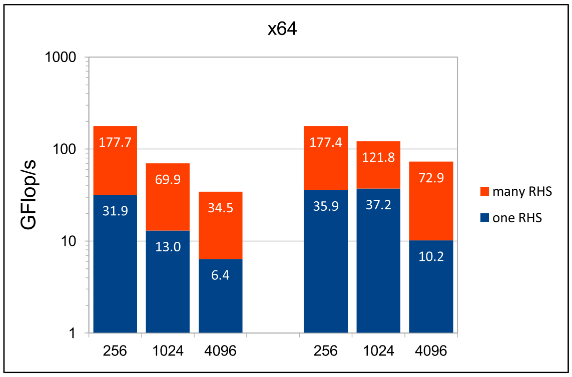

Figure 1. GFLOP/s for sparse matrix-vector multiplication (one RHS) and sparse matrix-matrix multiplication (many RHS) depending on h^{-1} in DP and SP (left and right three columns, respectively) on the AMD CPU.

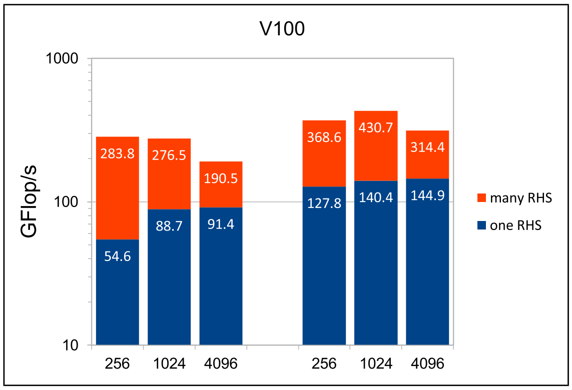

Figure 2. GFLOP/s for sparse matrix-vector multiplication (one RHS) and sparse matrix-matrix multiplication (many RHS) depending on h^{-1} in DP and SP (left and right three columns, respectively) on the V100 GPU.

accelerator hardware while measurably exploiting its high computing power.

Recent work using other methods also addresses this issue. For instance, Oo and Vogel (2020) investigate how profitably the V100 can be used in the context of a mixed-precision approach for Poisson's equation, that is, an iterative refinement algorithm with a low-precision multigrid preconditioning involving large, sparse matrices and observe a speed-up by a factor of 2.5 when solving the linear system. Since the operations with sparse matrices are memory-bounded, the peak rates of the V100 are not approached. Another recent paper on this subject is a comprehensive survey, that summarizes some of the latest results on mixed-precision numerical (dense and sparse) linear algebra routines (Abdelfattah et al., 2021). Moreover, a mixed-precision iterative refinement solver for dense matrices with a bounded condition number, that employs an LU factorization in low precision, by Haidar et al. (2020) should be mentioned. The algorithm is accelerated by a factor of four to five compared to the completely double-precision method when running on a Tensor Core GPU. For the case of sparse matrices, there are fewer practical results concerning the use of Tensor Cores, which is not surprising given their special suitability for dense linear algebra. Instead, the focus is on format decoupling and compression and their application to preconditioners for iterative solvers and multigrid methods.

An alternative approach is matrix-free methods. In the article by Kronbichler and Ljungkvist (2019), for example, a matrix-free multigrid algorithm to solve Poisson's equation on NVIDIA's Pascal P100 GPU is examined and compared with a CPU implementation, resulting in a speedup by a factor of 1.5–2. The advantages of using single precision are highlighted, but this methodology also likely offers little potential for use with Tensor Core GPUs, if using half precision is even a consideration for reasons concerning the conditioning.

The objective of this proof-of-concept study is to develop hardware-oriented algorithms for solving Poisson's equation in 2D, which is oftentimes a bottleneck in numerical simulations, on current GPUs and to demonstrate that computations can be accelerated by orders of magnitude in comparison to standard methods on standard hardware given by a geometric multigrid method on multicore CPUs while preserving the required accuracy.

2. An example of modern accelerator hardware

We take a closer look at the two sets of hardware compared in this study. On the one hand, the aforementioned NVIDIA Tesla V100 SXM2 as an example of current GPUs offers 7.8 TFLOP/s in double (DP), 15.7 TFLOP/s in single (SP), 31.3 TFLOP/s in half precision (HP), and 125 TFLOP/s in half precision in conjunction with Tensor Cores. The memory bandwidth is 900 GB/s (NVIDIA, 2020). On the other hand, we consider an x64 architecture in the form of the AMD EPYC 7542 CPU with the following specifications: 32 cores per CPU, 128 MB L3-Cache, 1.5 TFLOP/s in double, 3 TFLOP/s in single precision (half precision is not supported) and a memory bandwidth of 205 GB/s (AMD, 2021) representing modern standard computers used in computer centers, for example.

It should be noted that the stated computer performance rates are peak rates. They can be achieved when tasks with very high arithmetic intensity including dense linear algebra (BLAS3 operations) such as direct solvers for linear systems of equations (with typically cubic complexity) are executed. However, finite element discretizations result in very sparse stiffness matrices and thus standard iterative solvers, for example, Krylov (multigrid) methods, particularly require sparse matrix-vector multiplications. Therefore, the computational performance is bounded by the memory bandwidth.

As a test model, we consider Poisson's equation $-\Delta u = f$ in 2D on the unit square $\overline{\Omega} = [0, 1]^2$ discretized by bilinear (Q1) finite elements on an equidistant mesh of grid width h. The corresponding stiffness matrix is stored in compressed sparse row (CSR) format (Saad, 2003). In the further course of this study, we also cover the subject of general triangular (P1) meshes. Both above hardware configurations were used to compute sparse matrix-vector and matrix-matrix products, whereby the second can be understood as a component when solving the linear system with respect to a number of different right-hand sides (RHS). The resulting GFLOP/s rates are depicted in Figures 1 and 2. It becomes obvious that the actual performance is far below the peak rates. For h = 1/1024 (which corresponds to approx. 1 million unknowns), the V100 GPU is only four to six times faster than the AMD CPU.

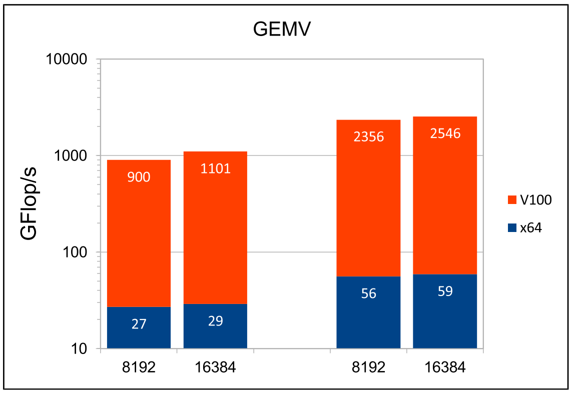

Figure 3. GFLOP/s for dense matrix-vector multiplication (GEMV) depending on h^{-1} in DP and SP (first and second set of columns from left, respectively) one the AMD CPU and the V100 GPU.

We also remark that our results for optimized matrix-free, more specifically stencil-based, operations on the V100 in the context of a geometric multigrid algorithm show a speedup factor of about 5–10 compared to a CSR approach. Thus, the order of one TFLOP/s is achievable, but this is still far below the peak rate (Poelstra, 2019).

If, instead, one performs calculations with dense matrices, that is, dense matrix-vector and matrix-matrix multiplications, a strong increase in achievable computing power can be observed, as the results shown in Figures 3 and 4 indicate. Note that half precision is not supported on the x64 architecture. The exceptionally high rates in half precision on the Tensor Cores of the V100 demonstrate their performance potential.

In a nutshell, in this section we have demonstrated the large gap between peak performance on the one hand and actual performance in the setting of finite element applications, that are usually based on sparse matrix-vector operations in double precision, on the other hand. In addition, it has become evident that there is a high potential with regard to numerical simulations of corresponding PDEs in continuum mechanics if low precision, and thus Tensor Cores, as well as dense matrix operations, can be used. To exploit this potential, it is necessary to ensure that the use of low precision does not lead to a significant overall loss of accuracy and to develop solution methods that include dense matrix-vector or matrix-matrix applications.

In the following sections, we pursue both requirements and introduce possible solutions. The next section treats the concept of prehandling (Ruda, 2020; Ruda et al., 2021) for Poisson's equation, which is an essential component in many numerical simulations. It enables the use of lower precision when solving the linear system with respect to the stiffness matrix while preserving results comparable to those achieved in double precision in terms of accuracy. It

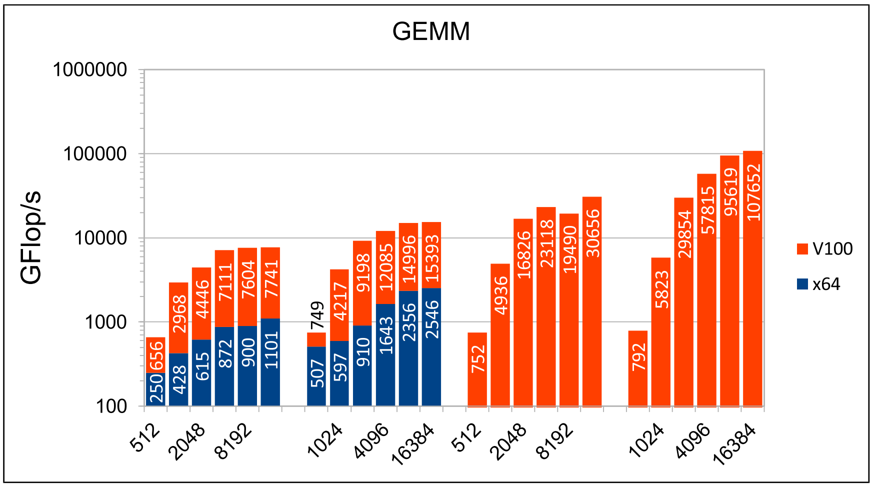

Figure 4. GFLOP/s for dense matrix-matrix multiplication (GEMM) depending on h^{-1} in DP and SP (first and second set of columns from left, respectively) on the AMD CPU and the V100 GPU and in the case of the V100 in HP without and with Tensor Cores (third and fourth set of columns from left, respectively).

can be realized via hierarchical finite elements (Yserentant, 1986) and this approach is then extended to a Schur complement like solution method including multiplications with dense (part) matrices so that Tensor Cores can be fully used. Finally, we provide first numerical results for this particular solver serving as a proof-of-concept that GPUs with Tensor Cores can in fact be appropriately used by means of adjusted hardware-oriented techniques.

3. Prehandling as a concept for stiffness matrices in lower precision

The primary objective of prehandling is to reduce the condition number of the stiffness matrix A_h corresponding to elliptic PDEs, Poisson's equation in this case. The necessity will become clear if the following subdivision of the error obtained by a finite element discretization (with mesh width h) given by the difference between the exact solution u and the actual numerical solution $\tilde{u}_h$ is considered

$$u - \tilde{u}_h = (u - u_h) + \left(u_h - \tilde{u}_h\right) \tag{1}$$

with the exact solution to the discrete problem u_h. It shows that the overall error consists of the discretization error $u - u_h$ satisfying $\left\|u - u_h\right\|_{L^2} = O(h^2)$ if (bi)linear shape functions are chosen and the computational error $u_h - \tilde{u}_h$ caused by roundoff that is characterized by $\left\|u_h - \tilde{u}_h\right\| \approx \mathrm{TOL} \cdot \kappa(A_h)$, whereby the machine accuracy TOL ($9.8 \cdot 10^{-4}$ in half, $1.2 \cdot 10^{-7}$ in single and $2.2 \cdot 10^{-16}$ in double precision) is a lower bound for the data error and the spectral condition number of the stiffness matrix is given by $\kappa(A_h) = O(h^{-2})$ in the case of Poisson's equation.

The opposite trends of both error types cause that the mesh width h may not be chosen too small in order to prevent the computational error from becoming dominant. This happens if the mesh width undercuts a certain critical value. In the above example, the intersection of discretization and computational error is at $h \approx \sqrt[4]{\mathrm{TOL}}$, so that the critical grid width is asymptotically given by $O(\sqrt[4]{\mathrm{TOL}})$. We show an illustrative and a practical example of this phenomenon in Figures 5 and 6. Yet, only if PDEs are to be solved on a fine mesh resulting in large linear systems, the use of high-performance computers with accelerator hardware is necessary and reasonable.

It is desirable to reduce the condition number of the stiffness matrix to widen the range of appropriate grid widths and even allow for the use of lower precision. This can be achieved by explicitly transforming the original linear system $A_h x_h = b_h$ into an equivalent form $\tilde{A}_h \tilde{x}_h = \tilde{b}_h$, $x_h = B\tilde{x}_h$, the method of prehandling, whereby we require the properties:

1. Strong decrease of the condition number, $\kappa(\tilde{A}_h) \ll \kappa(A_h)$.
2. The matrix $\tilde{A}_h$ is only moderately less sparse than A_h.
3. The transformation can be realized efficiently (for instance, in $O(N \log N)$ operations for N unknowns).

Provided that exact arithmetic is used, solvers in their (implicitly) preconditioned variants would yield the same results as the (explicitly) prehandled forms, but it turns out that the results can differ significantly in finite precision, in particular in case of ill-conditioned stiffness matrices. Well known preconditioners (matrix splitting methods, ILU, SPAI, etc.) that are directly applied to the stiffness matrix are not appropriate options for prehandling since they violate the requirements 1 and 2. The hierarchical finite element method (HFEM) meets the demands for Poisson's equation, at least in 1D and 2D. Below we outline the basic idea, remarkable properties, and practical aspects of this method.

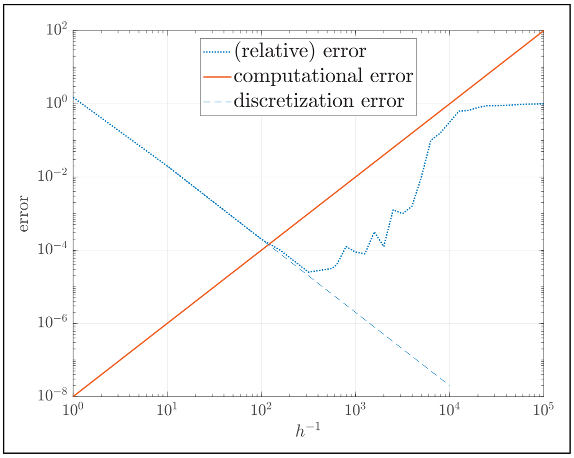

Figure 5. Illustrative course of total, computational, and discretization error in SP in the case of Poisson's equation with bilinear finite elements.

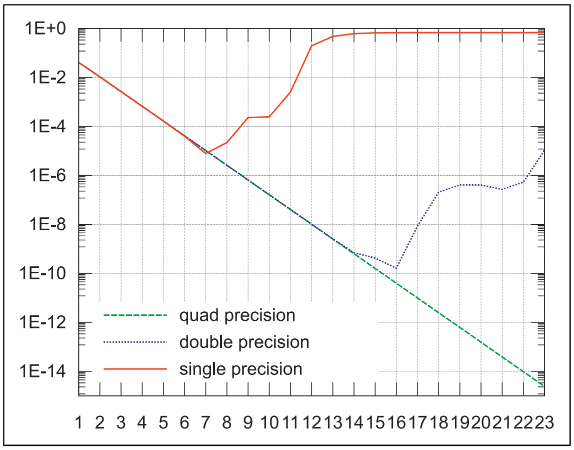

Figure 6. Actual L2-error for single, double, and quad precision with standard finite elements in 1D depending on the refinement level (h = $2^{-\text{level}}$, so level 10 corresponds to h = 1/1024).

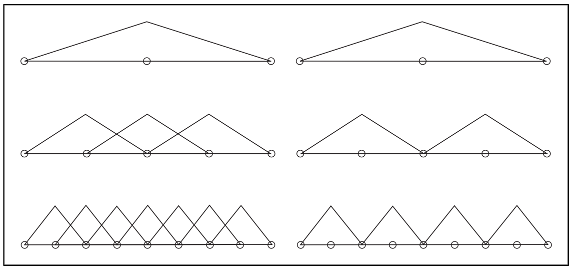

Figure 7. Left: nodal bases; right: hierarchical bases (only newly added basis functions on the respective meshes) in 1D. Source: Deuflhard et al. (1989).

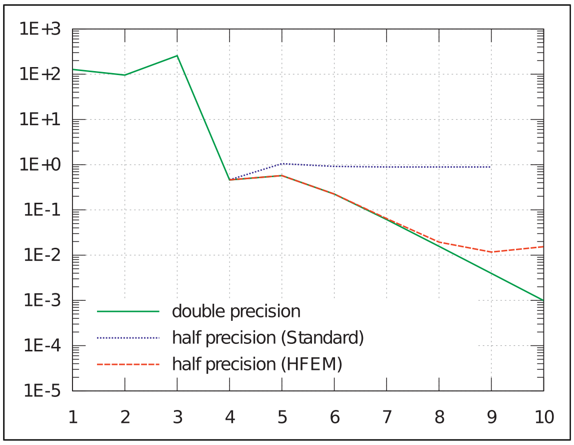

Figure 8. L2-errors depending on the refinement level without (dotted graph) and with prehandling via hierarchical finite elements (dashed graph) in HP, respectively, in comparison to the error of the reference solution obtained in DP (solid graph), which is independent of prehandling in this range of levels. A strongly oscillating exact solution to the continuous 2D Poisson problem was chosen. Again, h = $2^{-\text{level}}$. Note that the error deviates at level 5 without prehandling and at level 8 or 9 with prehandling.

It was established and analyzed by H. Yserentant et al. in the 1980s, for example, in (Yserentant, 1986). The prerequisite is an initial triangulation (level 0) consisting of triangular or quadrilateral elements that is successively refined yielding a nested sequence of finite element spaces. Here we restrict ourselves to (bi)linear Q1 or P1 finite element discretizations. Instead of a nodal basis, we use a hierarchical basis, which comprises shape functions on each level as shown in Figure 7 in the one-dimensional case. It is straightforward to generalize this idea to higher dimensions. We focus on the two-dimensional case.

It seems that the assembly of the stiffness matrix and the right-hand side with respect to a hierarchical basis is more complex due to the greater support of the basis functions on lower levels, but it is in fact not necessary to compute them: If the stiffness matrix and right-hand side with respect to a nodal basis are known, we can obtain the respective structures with respect to a hierarchical basis by means of a transformation via the matrix $S = S_j S_{j-1} \ldots S_1$. Each factor S_k in this product corresponds to one step of refinement and can be understood as a typical prolongation from level $k-1$ to k as known from geometric multigrid methods. In other words, multiplying a coefficient vector with the matrix S_k yields the values of the basis functions on the (k-1)st level at the nodes of the kth level. Hence, the matrices S_k are identity matrices with additional entries in the rows whose indices correlate with the newly added nodes of level k. In practice, assuming P1 finite elements and uniform refinement of the initial grid (i.e., subdividing each triangle into four congruent triangles), every added level-k-node with index i is adjacent to two level-($k-1$)-nodes, $n_1(i)$ and $n_2(i)$, so that we have

$$S_k\left(i, n_j(i)\right) = \frac{1}{2}, \quad j = 1, 2 \tag{2}$$

In the case of Q1 finite elements, uniform refinement leads to four edge midpoints with two adjacent nodes on the coarser grid, respectively, and one element midpoint with four adjacent nodes. As a result, the manipulation of the rows of S_k must be adapted as follows

$$S_k\left(i, n_j(i)\right) = \begin{cases} \frac{1}{2}, & j = 1, 2, \quad \text{if } x_i \text{ is edge midpoint} \\ \frac{1}{4}, & j = 1, \ldots, 4, \quad \text{if } x_i \text{ is element midpoint} \end{cases} \tag{3}$$

The arising matrix S is consequently a sparse block unit lower-triangular matrix and the transformed linear system of equations is given as $\tilde{A}_h \tilde{x}_h = \tilde{b}_h$, where

$$\tilde{A}_h = S^{\mathrm{T}} A_h S, \quad \tilde{b}_h = S^{\mathrm{T}} b_h \tag{4}$$

and the solution with respect to the nodal basis is regained via $x_h = S\tilde{x}_h$.

For better results in terms of condition numbers, Yserentant (1986) and Deuflhard et al. (1989) suggest an additional Cholesky decomposition on the initial grid, which can also be directly applied to the stiffness matrix. Let $\tilde{A}_h^0$ be the part of the stiffness matrix with respect to a hierarchical basis that corresponds to the nodes on level 0. If the nodes are numbered level-wise, this is the top left block of $\tilde{A}_h$. If the remaining part of the matrix is denoted by $\tilde{A}_h^{1,\ldots,j}$, we compute the Cholesky decomposition

$$\begin{pmatrix} \tilde{A}_h^0 & 0 \\ 0 & \operatorname{diag}\left(\tilde{A}_h^{1,\ldots,j}\right) \end{pmatrix} = LL^{\mathrm{T}} \tag{5}$$

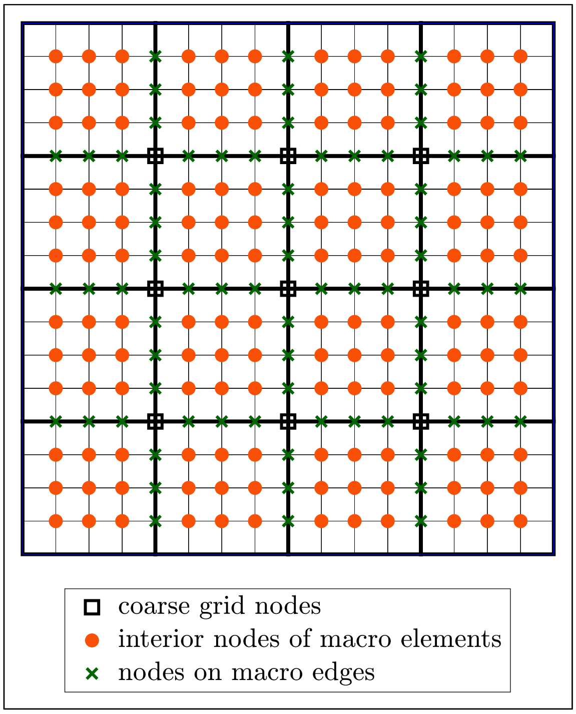

Figure 9. Illustration of the three types of nodes on a uniformly refined coarse grid (bold lines) for the unit square with h_0 = 1/4 and h = 1/16 and Q1 finite elements.

and apply further prehandling in accordance with $L^{-1}\tilde{A}_h L^{-\mathrm{T}}$ and $L^{-1}\tilde{b}_h$. The back-transformation of the solution vector is realized via multiplication with $SL^{-\mathrm{T}}$.

For both variants, with and without a partial Cholesky decomposition, the remarkable property is that the condition number of the transformed stiffness matrix is asymptotically characterized by $O((\log 1/h)^2)$ in 2D, as shown by Yserentant (1986), in contrast to $O((1/h)^2)$ in the case of standard finite elements without prehandling. However, in the 3D case the condition number of the matrix with respect to a hierarchical Basis is $O(1/h)$ (Ong, 1997). Due to the sparse structure of S, the transformation is not expensive and the resulting transformed stiffness matrix is still sparse, even if we apply a partial Cholesky decomposition since the number of nodes in the coarse grid is very small in comparison the final number of nodes. The advantage of this additional prehandling is a further significant decrease of the condition number without an excessive loss of sparsity. For detailed numerical results, see (Ruda, 2020; Ruda et al., 2021).

We conducted numerical tests that show that solutions obtained in lower (single or half) precision are considerably more accurate for fine meshes if we use the method of prehandling. An example can be seen in Figure 8. It clarifies that prehandling allows for (in this case) four to five further steps of uniform refinement in half precision without an exceeding loss of accuracy compared to standard finite element methods in double precision. Thus, the use of low precision becomes feasible, at least when solving Poisson's equation in 2D on hierarchically refined meshes and if a relative accuracy of approximately 1% is acceptable, whereby the latter is a realistic demand in complex technical simulations.

4. Direct solvers based on the HFEM approach

4.1. Derivation of the methods

Let us first take a closer look at the particular structure of hierarchically refined meshes and the consequent structure of stiffness matrices that arise from prehandling via the HFEM approach. Then we derive new solvers tailored for GPUs in low precision. As before, we investigate the case of Poisson's equation in 2D, discretized with P1 or Q1 finite elements. From now on, for simplicity, we omit the subscript h and assume that $A \in \mathbb{R}^{N\times N}$ and $b \in \mathbb{R}^{N}$ denote the stiffness matrix and the right-hand side that result from prehandling with the partial Cholesky decomposition. So according to the previous section, we have $A = L^{-1}S^T A_h S L^{-T}, b = L^{-1}S^T b_h$ and get the solution to the initial linear system (i.e., with respect to a nodal basis) by computing $x_h = SL^{-T}x$ if x satisfies $Ax = b$.

A subdivision of the nodes of the hierarchical mesh and numbering the nodes accordingly yields a special structure of the stiffness matrix that can be exploited to solve the linear system efficiently. We only consider the interior nodes of the discrete domain since nodes on the (Dirichlet) boundary are treated separately. A distinction is made between the following three sets of node indices:

- $\mathcal{C}$ stores the indices of nodes that belong to the coarse grid,
- $\mathcal{E}$ denotes the set of indices of fine-grid nodes lying on the edges of the coarse grid (excluding the coarse grid nodes), and
- $\mathcal{I}$ contains the indices of the remaining nodes located in the interior of the cells of the coarse grid (also referred to as macro elements).

Consequently, $\mathcal{C}\cup\mathcal{E}\cup\mathcal{I}$ is a disjoint union including all indices of the interior nodes of the discrete domain. An illustrative example of this subdivision of the nodes on the unit square can be seen in Figure 9. For our approach, it is further essential to number the nodes in $\mathcal{I}$ in a certain manner, namely macro element by macro element and in the same order in a geometrical sense for all groups of similar macro elements. This approach results in the matrix $A_{\mathcal{I}\mathcal{I}}$ (the part of A with rows and columns restricted to the indices of $\mathcal{I}$) consisting of separate blocks (as many as there are macro elements). The blocks corresponding to similar macro elements are identical. It follows from the prehandling via the complete

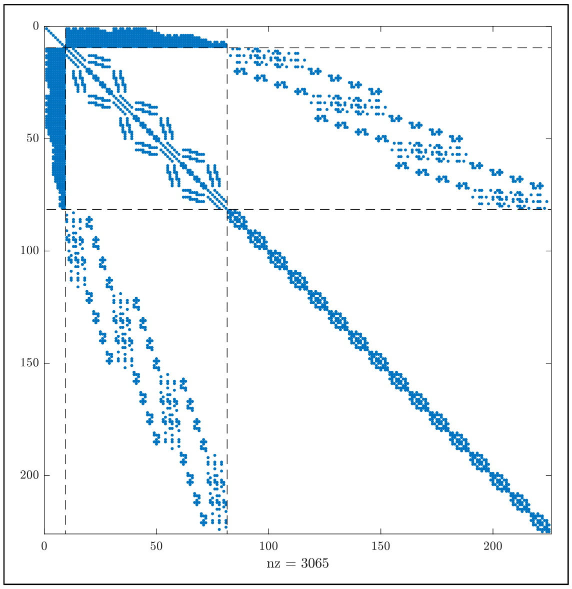

Figure 10. Sparsity pattern of the prehandled stiffness matrix corresponding to the mesh in Figure 9 for the special node numbering.

Cholesky decomposition on the coarse grid that $A_{\mathcal{CC}} = I$ where I denotes the identity matrix of corresponding size. Furthermore, in the case of arbitrary triangular (P1) and orthogonal quadrilateral (Q1) initial meshes, there is no coupling between the coarse grid nodes and the nodes in the interior of the macro cells because the corresponding entries in the stiffness matrix cancel out when the hierarchical basis representation is employed. Hence, $A_{\mathcal{CI}} = A_{\mathcal{IC}}^T = 0$ under the stated conditions. The remaining blocks $A_{\mathcal{EE}}$ and $A_{\mathcal{EI}}$ (and its transpose $A_{\mathcal{IE}}$) are sparse matrices, whereas $A_{\mathcal{CE}}$ and $A_{\mathcal{EC}} = A_{\mathcal{CE}}^T$ are full with a density of up to 50% because these are the only blocks that are affected by the multiplication with the dense, non-diagonal part of L^{-1} and L^{-T} in terms of fill-in. See Figure 10 for an exemplary presentation of the sparsity pattern that results from the above node numbering in the order $\mathcal{C}$, $\mathcal{E}$, $\mathcal{I}$.

It is worth noting that, when a given coarse grid with width h_0 is uniformly refined, the number of interior nodes $|\mathcal{I}|$ grows as h^{-2}. The total number of unknowns N grows quadratically as well. However, the increase in $|\mathcal{E}|$ is proportional to h^{-1} (or $\sqrt{N}$), while $|\mathcal{C}|$ is constant during refinement. In the simple test case of the unit square discretized with Q1 elements on a uniform mesh, we have

$$|\mathcal{C}| = \left(\frac{1}{h_0} - 1\right)^2 \tag{6}$$

$$|\mathcal{E}| = 2\left(\frac{1}{h_0} - 1\right)\left(\frac{1}{h} - \frac{1}{h_0}\right) \tag{7}$$

$$|\mathcal{I}| = \left(\frac{1}{h} - \frac{1}{h_0}\right)^2 \tag{8}$$

and $A_{\mathcal{II}}$ decomposes into h_0^{-2} indistinguishable, independent blocks with $(h_0/h - 1)^2$ rows/columns each.

Using the notation $A_{\mathcal{CE}} = B$, $A_{\mathcal{EE}} = E$, $A_{\mathcal{EI}} = D$ and $A_{\mathcal{II}} = C$ for better readability, the linear system of equations can be written as

$$\begin{pmatrix} I & B & 0 \\ B^{\mathrm{T}} & E & D \\ 0 & D^{\mathrm{T}} & C \end{pmatrix}\begin{pmatrix} x_{\mathcal{C}} \\ x_{\mathcal{E}} \\ x_{\mathcal{I}} \end{pmatrix} = \begin{pmatrix} b_{\mathcal{C}} \\ b_{\mathcal{E}} \\ b_{\mathcal{I}} \end{pmatrix} \tag{9}$$

or in the equivalent form

$$x_{\mathcal{C}} + Bx_{\mathcal{E}} = b_{\mathcal{C}} \tag{10}$$

$$B^{\mathrm{T}}x_{\mathcal{C}} + Ex_{\mathcal{E}} + Dx_{\mathcal{I}} = b_{\varepsilon} \tag{11}$$

$$D^{\mathrm{T}}x_{\mathcal{E}} + Cx_{\mathcal{I}} = b_{\mathcal{I}} \tag{12}$$

These three subsystems can be rearranged and substituted into each other in several ways, similarly to a Schur complement method for 3 × 3 block matrices. The three resulting different (semi-)direct methods for solving the above system have the potential to exploit the computing power of GPUs in lower precision since dense and well-conditioned matrices and corresponding matrix-vector and matrix-matrix multiplications are involved. As a collective term, we also refer to the methods as PSC methods (Prehandling Schur Complement). They are derived as follows.

4.1.1. (M1) Direct Method 1. For the derivation of the first method, we substitute the suitably rearranged equations (10) and (12) into equation (11). For a better overview, Schur complements are denoted by Greek capital letters. Let Λ be the Schur complement of C in $\begin{pmatrix} E & D \\ D^T & C \end{pmatrix}$, that is

$$\Lambda = E - DC^{-1}D^{\mathrm{T}} \tag{13}$$

and let Π be the Schur complement of I in the matrix $\begin{pmatrix} I & B \\ B^T & \Lambda \end{pmatrix}$, that is

$$\Pi = \Lambda - B^{\mathrm{T}}B \tag{14}$$

Using block elimination and the above definitions, we obtain the solution

$$x_{\mathcal{E}} = \Pi^{-1}\left(b_{\mathcal{E}} - B^{\mathrm{T}}b_{\mathcal{C}} - DC^{-1}b_{\mathcal{I}}\right) \tag{15}$$

$$x_{\mathcal{C}} = b_{\mathcal{C}} - Bx_{\mathcal{E}} \tag{16}$$

$$x_{\mathcal{I}} = C^{-1}\left(b_{\mathcal{I}} - D^{\mathrm{T}}x_{\mathcal{E}}\right) \tag{17}$$

Table 1. Cardinality of index sets for nodes of different types, spectral condition numbers of the matrices C_i and Π and number of nonzero entries in their inverses relative to NNZ_{FEM} (nonzeros in standard FEM stiffness matrix) for different fine (h) and coarse mesh sizes (h_0) on the unit square discretized with Q1 finite elements. The total number of interior nodes N is listed in parentheses.

h (N)	h_0	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{I}\rvert$	$\kappa(C_i)$	$\kappa(\Pi)$	$\text{NNZ}(C_i^{-1})/\text{NNZ}_{\text{FEM}}$	$NNZ(\Pi^{-1})/NNZ_{FEM}$
1/256 (65,025)	1/4	9	1,512	63,504	23.9	24.1	27.06	3.93
	1/8	49	3,472	61,504	16.9	19.5	1.59	20.71
	1/16	225	7,200	57,600	11.1	14.5	0.09	89.05
	1/32	961	13,888	50,176	6.6	10.1	0.004	331.31
1/512 (261,121)	1/4	9	3,048	258,064	32.2	30.7	110.99	3.96
	1/8	49	7,056	254,016	23.9	25.5	6.72	21.24
	1/16	225	14,880	246,016	16.9	19.8	0.39	94.46
	1/32	961	29,760	230,400	11.1	14.6	0.02	377.85
1/1024 (1,046,529)	1/4	9	6120	1,040,400	41.8	37.9	449.50	3.98
	1/8	49	14,224	1,032,256	32.2	32.3	27.66	21.51
	1/16	225	30,240	1,016,064	23.9	25.9	1.67	97.22
	1/32	961	61,504	984,064	16.9	19.9	0.10	402.14

4.1.2. (M2) Direct Method 2.

Another approach is substituting equations (11) and (12) into equation (10) of the block system. In this case one needs the Schur complement of Λ, instead of I, in $\begin{pmatrix} I & B \\ B^T & \Lambda \end{pmatrix}$ denoted by Θ and given as

$$\Theta = I - B\Lambda^{-1}B^{\mathrm{T}} \tag{18}$$

After rearranging, the procedure of solving the linear system of equations reads as

$$x_{\mathcal{C}} = \Theta^{-1}\left[b_{\mathcal{C}} + B\Lambda^{-1}\left(DC^{-1}b_{\mathcal{I}} - b_{\mathcal{E}}\right)\right] \tag{19}$$

$$x_{\mathcal{E}} = \Lambda^{-1}\left(b_{\mathcal{E}} - B^{\mathrm{T}}x_{\mathcal{C}} - DC^{-1}b_{\mathcal{I}}\right) \tag{20}$$

$$x_{\mathcal{I}} = C^{-1}\left(b_{\mathcal{I}} - D^{\mathrm{T}}x_{\mathcal{E}}\right) \tag{21}$$

4.1.3. (M3) Semi-direct Method.

The third method is semi-direct. The sets of indices $\mathcal{C}$ and $\mathcal{E}$ are united and we consider the Schur complement Σ of C in the entire matrix A

$$\Sigma = \begin{pmatrix} I & B \\ B^{\mathrm{T}} & E \end{pmatrix} - \begin{pmatrix} 0 \\ D \end{pmatrix} C^{-1} \begin{pmatrix} 0 & D^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} I & B \\ B^{\mathrm{T}} & \Lambda \end{pmatrix} \tag{22}$$

and solve for the components

$$\begin{pmatrix} I & B \\ B^{\mathrm{T}} & \Lambda \end{pmatrix} \begin{pmatrix} x_{\mathcal{C}} \\ x_{\mathcal{E}} \end{pmatrix} = \begin{pmatrix} b_{\mathcal{C}} \\ b_{\mathcal{E}} - DC^{-1}b_{\mathcal{I}} \end{pmatrix} \tag{23}$$

$$x_{\mathcal{I}} = C^{-1}\left(b_I - D^{\mathrm{T}}x_{\mathcal{E}}\right) \tag{24}$$

The first step (23) can be implemented using an iterative algorithm as the conjugate gradient method (Saad 2003). Each occurring product of the matrix Σ with a vector can be subdivided into the almost dense components, that is, multiplication with B and B^{T}, that can be profitably computed on a GPU, and the sparse matrix-vector multiplication with Λ.

Note that within all three methods due to the block structure of the matrix C, its inverse C^{-1} only needs to be computed and saved once for each group of similar rectangles or triangles. In other words, if the coarse grid consists of M groups of similar elements and group i consists of m_i similar elements for $i \in \{1, \ldots, M\}$, we have

$$C = \operatorname{diag}\left(\underbrace{C_1,\ldots,C_1}_{m_1}, \underbrace{C_2,\ldots,C_2}_{m_2}, \ldots, \underbrace{C_M,\ldots,C_M}_{m_M}\right) \tag{25}$$

and explicitly compute the inverses C_i^{-1} for $i = 1, \ldots, M$. Each multiplication of C^{-1} by a vector $v \in \mathbb{R}^{|\mathcal{I}|}$ can be reduced to dense matrix-matrix products by splitting it up into M parts $v = (v_1,\ldots,v_M)^T$ that correspond to the groups of similar coarse elements. For $i = 1, \ldots, M$ the parts v_i are again subdivided into m_i equally sized parts $v_i^1,\ldots,v_i^{m_i}$ that are reshaped into a matrix $V_i = (v_i^1,\ldots,v_i^{m_i})$ with m_i columns. The product $C^{-1}v$ is thus given by the partial products $C_i^{-1}V_i$ for $i = 1, \ldots, M$ converted into a vector.

The same principle can be applied if a matrix is multiplied by C^{-1}, which is important if the initial linear system is solved for multiple, $N_{\text{rhs}} > 1$, right-hand sides simultaneously leading to $AX = B$ where $X, B \in \mathbb{R}^{N \times N_{rhs}}$ are matrices of the solution vectors and right-hand sides, respectively. The involved linear systems with respect to C, Π, Θ, and Λ can be solved with methods other than multiplication by inverses, but if $N_{\text{rhs}} \gg 1$, it is worthwhile to compute the inverses of these moderately large matrices and exploit the additional potential of Tensor Cores. An example of

Table 2. Total NNZ in multiples of N of the matrices Π^{-1} and C_i^{-1} for Q1 and P1 finite elements on the unit square depending on the coarse grid width $h_0 = 2^l\sqrt{h}$ for h = 1/1024 (N = 1, 046, 529) and h = 1/256 (N = 65, 025).

	Q1		P1	
ℓ	$h = 1/256$	$h = 1/1024$	$h = 1/256$	$h = 1/1024$
−1	4,080	16,368	9,180	36,828
0	1,021	4,093	2,295	9,207
1	271	1,039	578	2,306
2	320	512	207	639
3	4,112	4,160	1,060	1,168

Table 3. Number of FLOP in multiples of $N^{\frac{3}{2}}$ for matrix-vector multiplication with Π^{-1}, C^{-1} (twice) and in total for Q1 and P1 finite elements on the unit square depending on the coarse grid width $h_0 = 2^l\sqrt{h}$.

	Q1			P1		
ℓ	Π^{-1}	C^{-1}	Total	Π^{-1}	C^{-1}	Total
−1	32.0	1.0	33.0	72.0	0.5	72.5
0	8.0	4.0	12.0	18.0	2.0	20.0
1	2.0	16.0	18.0	4.5	8.0	12.5
2	0.5	64.0	64.5	1.1	32.0	33.1
3	0.1	256.0	256.1	0.3	128.0	128.3

application of multiple right-hand sides is given at the end of Section 5.

When comparing the direct methods (M1) and (M2) it is evident, that (M2) is more computationally demanding because, in contrast to (M1), it does not just involve the assembly of the matrix Λ but also its inversion. The semi-direct method (M3) has the advantage that besides C^{-1} no other storage consuming dense $|\mathcal{E}| \times |\mathcal{E}|$ -matrices such as Π^{-1} in (M1) or Λ^{-1} in (M2) need to be computed and saved. In this work, we cover the direct method (M1) while (M3) is subject of forthcoming studies.

4.2. Properties of the matrices

Let us now further investigate the properties of the auxiliary matrices that are needed for (M1). Table 1 shows the cardinality of index sets for nodes of different types, condition numbers of the matrices C_i and Π and the number of nonzero entries (NNZ) in their inverses relative to the number of nonzeros $\mathrm{NNZ}_{\mathrm{FEM}}$ in a standard finite element stiffness matrix depending on fine and coarse grid widths for the simple test case of the uniformly refined 2D unit square. If finite elements (Q1) without prehandling are used, one obtains ill-conditioned stiffness matrices for Poisson's equation with a spectral condition number in the order of $2 \cdot 10^5$ for the mesh size $h = 1/1024$. In this respect, the condition numbers of C_i and Π are remarkably low. This is primarily achieved by prehandling via hierarchical bases and Cholesky decomposition and enhanced by extracting a partial block matrix or computing the Schur complement for C_i and Π, respectively. As an example, if we have $h = 1/256$ and $h_0 = 1/16$ on the unit square discretized with Q1 finite elements, the condition number of the stiffness matrix without any prehandling is 13,280, transforming it with respect to a hierarchical basis yields a condition number of 100, which decreases to 22 if also the partial Cholesky decomposition is applied for prehandling. Such low condition numbers enable the use of single and even half precision when solving the linear system while preserving sufficient accuracy. By using (M1), rather than basic prehandling without Schur complements, one can achieve even higher accuracy if the matrices Π and C_i^{-1} are assembled in double precision and then converted into half precision for the solution purposes.

Obviously, the (M1) algorithm requires more storage than many standard approaches, especially because the dense inverse matrices C_i^{-1} and Π^{-1} need to be saved. To get an idea of their sizes we indicate the number of nonzero entries (equaling the total number of entries since the matrices are dense) divided by the number of nonzeros of the sparse standard finite element stiffness matrix for the associated fine grid. This matrix has at most nine nonzero entries per row due to the nine-point stencil of Q1 finite elements. When the coarse grid is refined while keeping h constant, the size of the single blocks C_i of C and thus of C_i^{-1} decreases because there are more macro cells containing less interior nodes. At the same time the number of nodes on the edges of the coarse grid $|\mathcal{E}|$ increases and as a result the matrix Π and its inverse grow in size. The next subsection addresses the question of which coarse mesh width should be chosen to obtain a good compromise in terms of storage and computational costs.

4.3. Estimation of complexity and storage requirement

To determine an estimation of complexity and storage requirement, we first stick with the case of the unit square and Q1 finite elements and introduce the auxiliary variable ℓ that specifies by how many refinement levels the coarse grid width h_0 differs from $\sqrt{h}$, that is, $h_0 = 2^\ell\sqrt{h}$ ($\ell = \ldots, -1, 0, 1, 2, \ldots$) if h is an even power of two. We only consider the relevant components of (M1), storing and applying the matrices Π^{-1} and C_i^{-1}, whereas applying B^T, that is only of size $|\mathcal{E}| \times |\mathcal{C}|$, and D, that is sparse, as well as vector additions are negligible.

We first estimate the storage cost for the dense matrix $\Pi^{-1} \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{E}|}$ with respect to ℓ and the total number of

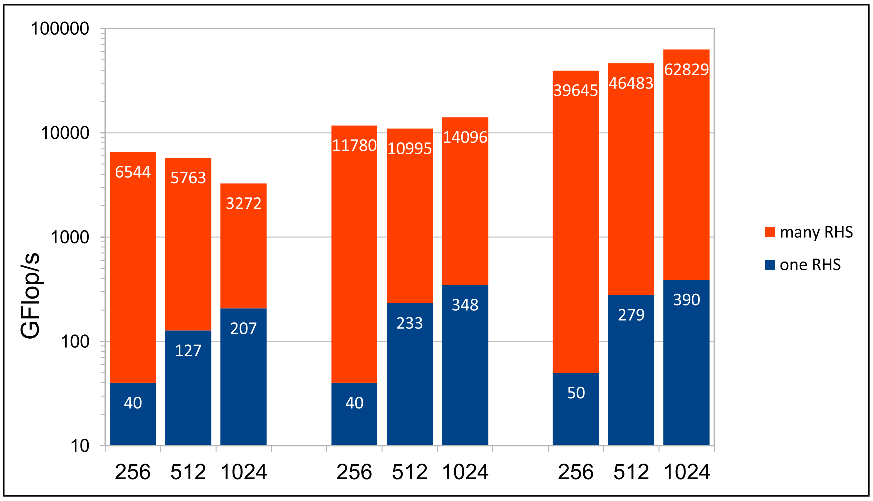

Figure 11. GFLOP/s for (M1) with one and many RHS depending on h^{-1} on the V100 GPU in DP, SP, and HP with Tensor Cores (left, middle, and right three columns, respectively).

unknowns N. From the expression (7) for the number of nodes in the set $\mathcal{E}$ follows

$$|\mathcal{E}| \approx 2\frac{1}{2^\ell\sqrt{h}}\frac{1}{h} = \frac{2}{2^\ell}h^{-\frac{3}{2}} \approx \frac{2}{2^\ell}N^{\frac{3}{4}} \tag{26}$$

where we use the relation $N = (h^{-1} - 1)^2 \approx h^{-2}$ in the last step. Since it is a square matrix, the total number of (nonzero) entries is given by

$$\mathrm{NNZ}\left(\Pi^{-1}\right) = |\mathcal{E}|^2 \approx \frac{4}{4^\ell}N^{3/2} \tag{27}$$

Consequently, a matrix-vector multiplication with Π^{-1} costs approximately $8/4^\ell N^{3/2}$ arithmetic operations (FLOP).

To obtain analogous results for C_i^{-1} we again estimate the number of rows/columns. In the case at hand all macro cells are congruent squares so that all blocks of C and C^{-1} are identical and there is only one partial matrix C_i^{-1} to be computed and stored. The nodes in the set $\mathcal{I}$, whose cardinality is given in equation (8), are divided into h_0^{-2} parts so that the number of rows of C_i^{-1} is

$$h_0^2|\mathcal{I}| \approx 4^\ell h\frac{1}{h^2} = 4^\ell h^{-1} \approx 4^\ell N^{1/2} \tag{28}$$

and thus

$$\mathrm{NNZ}\left(C_i^{-1}\right) \approx 16^\ell N \tag{29}$$

As a result, a matrix-vector multiplication with C_i^{-1} costs $2 \cdot 16^\ell N$ FLOP. To compute a product with the entire matrix C^{-1} we need h_0^{-2} (that can also be written as $1/4^\ell h^{-1}$ or approx. $1/4^\ell N^{1/2}$) of these partial products which leads to a FLOP number of

$$\frac{1}{4^\ell}N^{1/2} \cdot 2 \cdot 16^\ell N = 2 \cdot 4^\ell N^{3/2} \tag{30}$$

Note that the solution process of (M1) includes two applications of C^{-1}.

To summarize, the total storage requirement of the relevant matrices of (M1) is

$$\mathrm{NNZ}\left(\Pi^{-1}\right) + \mathrm{NNZ}\left(C_i^{-1}\right) \approx \frac{4}{4^\ell}N^{3/2} + 16^\ell N \tag{31}$$

and the computational costs for the relevant steps are $\left(\frac{8}{4^\ell} + 4 \cdot 4^\ell\right)N^{3/2}$ FLOP.

If the unit square is instead discretized with isosceles right triangles (with legs of length h) and linear (P1) finite elements are used, similar results can be derived. The basic difference to the Q1 case is that for the same values of h and h_0 there are twice as many macro cells, fewer nodes in the interior and more nodes on the edges of the macro cells, namely

$$|\mathcal{E}| = \left(\frac{3}{h_0} - 2\right)\left(\frac{1}{h} - \frac{1}{h_0}\right) \tag{32}$$

$$|\mathcal{I}| = \left(\frac{1}{h} - \frac{2}{h_0}\right)\left(\frac{1}{h} - \frac{1}{h_0}\right) \tag{33}$$

while $|\mathcal{C}| = (1/h_0 - 1)^2$ remains. On that basis, we can conduct similar estimations as above and obtain

$$\mathrm{NNZ}\left(\Pi^{-1}\right) + \mathrm{NNZ}\left(C_i^{-1}\right) \approx \frac{9}{4^\ell}N^{3/2} + \frac{16^\ell}{4}N \tag{34}$$

as storage requirement. Moreover, the computational costs of applying Π^{-1} and C_i^{-1} to a vector are approximately $18/4^\ell N^{3/2}$ FLOP and $4^\ell N^{3/2}$ FLOP, respectively, so that in total $(18/4^\ell + 2 \cdot 4^\ell)N^{3/2}$ FLOP are required.

In order to get an intuitive understanding of the results, we show some important data concerning storage in

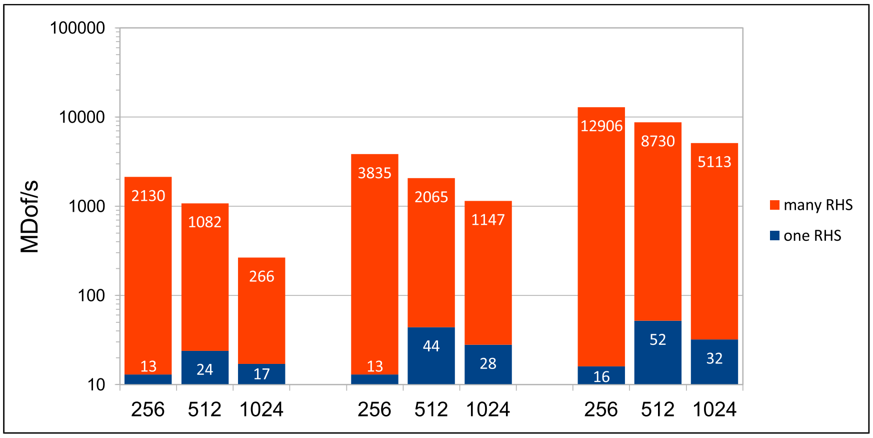

Figure 12. MDof/s (million degrees of freedom solved per second) for (M1) with one and many RHS depending on h^{-1} on the V100 GPU in DP, SP, and HP with Tensor Cores (left, middle, and right three columns, respectively).

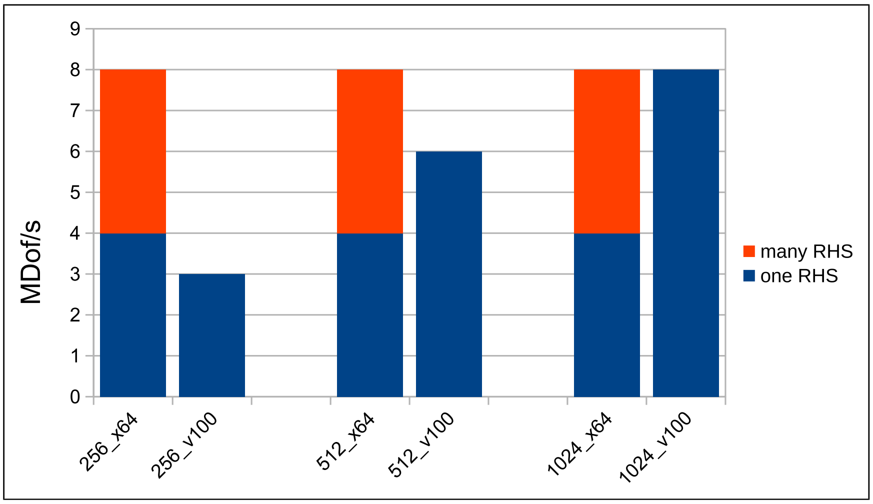

Figure 13. MDof/s with a standard method (standard finite elements and multigrid in DP) on the AMD CPU (x64) for one and many RHS (left columns) and on the V100 for one RHS (right columns) depending on h^{-1}.

Table 2 and computational cost in Table 3. We can see in Table 2 that the least storage-intensive case is $\ell = 2$. The least amount of total FLOP (approx. twelve multiples of $N^{3/2}$), however, is reached if $\ell = 0$ (Q1) or $\ell = 1$ (P1) is chosen as the results in Table 3 indicate. To conclude, it might be profitable to invest more storage space to minimize the complexity (by the choice $h_0 = \sqrt{h}$ for Q1) and thus accelerate the computation of the solution to the linear system.

5. Numerical results and comparison to standard FEM solvers on standard hardware

We now perform a numerical study to evaluate the new method (M1) in practice. In particular, we focus our attention to the question of whether we can exploit the computing power of the V100 and its Tensor Cores to achieve an advantage over a standard method, in this case a geometric finite element multigrid solver in double precision using the CSR format for matrices on the multicore AMD EPYC 7542 CPU. The geometric multigrid solver used in these benchmarks is a simple V-cycle multigrid which performs four smoothing steps of a Richardson iteration damped by the factor 0.3 as both a pre- and post-smoother on each multigrid level. The coarse mesh in the multigrid hierarchy consists only of one element, so all degrees of freedom on the coarse mesh are restrained by the Dirichlet boundary conditions and therefore no coarse grid solver is required. Because our benchmark problems do not contain anisotropies or other difficulties, this simple multigrid configuration turned out to be

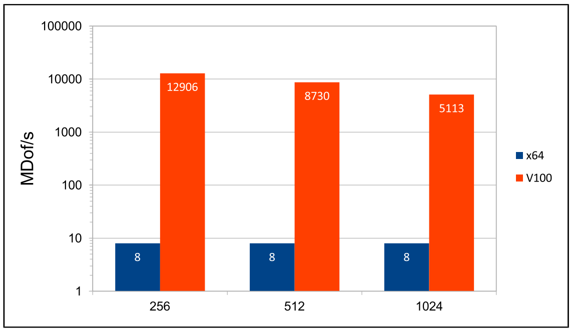

Figure 14. Comparison of MDof/s between standard method on the AMD CPU (left columns) and the direct method (M1) in HP with Tensor Cores on the V100 (right columns), both for many RHS, depending on h^{-1}.

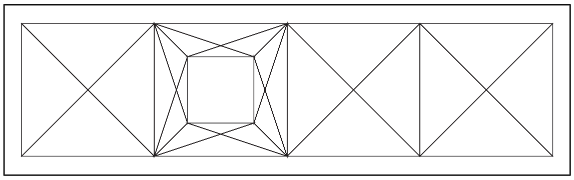

Figure 15. Coarsest mesh (level L = 0) of a flow around a square configuration on the domain $\Omega = (0,4) \times (0,1) \backslash [5/4,7/4] \times [1/4,3/4] \subset \mathbb{R}^2$.

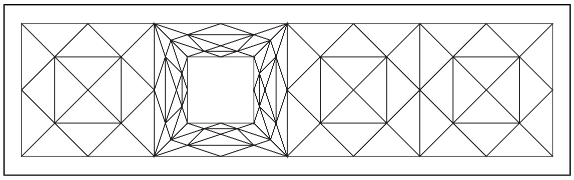

Figure 16. Mesh shown in Figure 15 after one step of uniform refinement (level L = 1).

the most performant one in our tests as compared to configurations with more sophisticated smoothers such as ILU, SPAI or SOR, whose results we omit here because they are beyond the scope of this work. The associated software is the finite element software package FEAT3[1] (Ruelmann et al., 2021). Both hardware configurations are specified in the beginning of Section 2. We present numerical results for the unit square discretized using a uniform mesh of Q1 finite elements for the pairs of fine and coarse mesh sizes $(h,h_0) \in \{(1/256, 1/16),(1/512,1/16),(1/1024,1/32)\}$, the latter leading to $N \approx 1$ million unknowns.

Figures 11 and 12 show the results of the new approach (M1) regarding GFLOP/s rates in Figure 11 and another measure, that is necessary to make the different methods comparable, that is, averaged number of million degrees of freedom solved per second (MDofs/s), in Figure 12, in double, single, and half precision and for one ($N_{\text{rhs}} = 1$) and many ($N_{\text{rhs}} \gg 1$) right-hand sides, respectively. As expected, the case of solving Poisson's equation on the same mesh for many right-hand sides simultaneously yields the highest performance. More precisely, we get an actual performance of 62,829 GFLOP/s or approximately 60 TFLOP/s in our study of (M1) with $h = 1/1024$, $N_{\text{rhs}} \gg 1$ and in half precision on the V100 while (almost) no accuracy is lost due to the well-conditioned matrices. Furthermore, the outstanding performance depicted in Figure 11, especially in half precision, clearly indicates that the V100 is effectively used. According to the specification of the V100, the speed-up from single to half precision without Tensor Cores is a factor of 2 (with Tensor Cores 8). However, in this example it is 4.5 times faster in half precision due to Tensor Cores.

The observed GFLOP/s rates do not provide information about the efficiency of (M1) and do not enable a comparison to the standard method because we also need to take the complexity into account. As described in the previous section, we expect approximately $12N^{3/2}$ arithmetic operations if (M1) is applied to solve a problem with N unknowns. This is not optimal considering the $O(N)$ complexity of multigrid methods. As a rule of thumb, one needs approximately $1{,}000N$ FLOP to obtain a sufficiently accurate result with a multigrid algorithm on a structured mesh like the ones that we use in this study. Hence, if we have $N \approx 1$ million ($h = 1/1024$ representing a large-scale system in 2D) we compare the factors 1,000 (multigrid) and $12N^{3/2} \approx 12{,}000$ (M1), that is, twelve times more arithmetic operations for the direct method (M1). Yet, the very effective use of the hardware performance of the V100 by (M1) not only compensates for this difference in complexity but also leads to a significantly higher overall efficiency due to less computing time (in MDof/s) as the results in Figures 13 and 14 show. If we consider the case of one right-hand side and compare the results of (M1) on the V100 in half precision (right set of blue columns in Figure 12) with the standard approach in double precision in Figure 13, we see that (M1) is up to four times faster if the V100 is used for the standard method and up to eight times if the AMD CPU is used. In the case of many right-hand sides, the multigrid algorithm on the AMD CPU solves for an average of 8 MDof/s for the range of mesh sizes considered in this study (see columns for many RHS in Figure 13). In Figure 13 we do not present results on the multigrid method with many right-hand sides on the V100 because the development of an optimized GPU multigrid code for this case (realized in FEAT3, see above) is still part of ongoing work. With (M1) on the V100 in half precision with Tensor Cores (right set of columns for many RHS in Figure 12) we observe a further gain in efficiency. To be precise, we get the following values of MDof/s: 12,906 ($h = 1/256$), 8,730

Table 4. Number of three types of nodes, spectral condition numbers of the matrices C_i (here $\max_{i=1,2,3}\kappa(C_i)$) and Π and number of nonzero entries in their inverses (added up for the three different C_i^{-1}) relative to $\mathrm{NNZ}_{\mathrm{FEM}}$ (nonzeros in standard FEM stiffness matrix) for different fine (L) and coarse grid refinement levels (L_0) on the flow around a square configuration (see Figures 15 and 16) with P1 finite elements. The total number of interior nodes N is listed in parentheses.

L (N)	L_0	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{I}\rvert$	$\kappa(C_i)$	$\kappa(\Pi)$	$\mathrm{NNZ}(C_i^{-1})/\mathrm{NNZ}_{\mathrm{FEM}}$	$\mathrm{NNZ}(\Pi^{-1})/\mathrm{NNZ}_{\mathrm{FEM}}$
7 (228,480)	0	7	4,445	224,028	121.9	61.0	137.14	14.11
	1	42	9,702	218,736	82.3	82.7	8.17	67.21
	2	196	19,964	208,320	45.9	69.0	0.46	284.60
	3	840	39,480	188,160	23.2	48.9	0.02	1,113.00
8 (915,712)	0	7	8,925	906,780	168.2	80.8	559.96	14.18
	1	42	19,558	896,112	121.9	111.2	34.18	68.08
	2	196	40,572	874,944	82.3	97.3	2.04	292.95
	3	840	81,592	833,280	45.9	74.1	0.12	1,184.79

(h = 1/512) and 5,113 (h = 1/1024). That is to say, by using (M1) on the V100 we are 600 times faster than we are with a multigrid method on the AMD CPU if h = 1/1024 and $N_{\mathrm{rhs}} \gg 1$ without losing accuracy. Finally, Figure 14 shows the MDof/s values for a direct comparison between (M1) in half precision on the V100 with Tensor Cores and a multigrid method in double precision on the AMD CPU for many right-hand sides.

Regarding the scalability of the presented method, it can be said that for slightly higher problem sizes the performance might even further improve as the Tensor Cores can be better utilized. However, there is a limitation due to the high memory requirement of $O(N^{3/2})$, which is why we are currently working on the more memory-efficient semi-direct method (M3).

5.1. Other meshes

So far, we have only covered the simple case of the equidistantly refined unit square although the finite element multigrid method we use for comparative purposes is designed for arbitrary coarse grids, in particular partially unstructured coarse grids that are refined hierarchically. Nevertheless, the comparison of both methods is valid because the new approach (M1) can also be applied to partially unstructured and uniformly refined triangular coarse grids. Figures 15 and 16 show a (typical) flow around a square configuration as an example of a mesh we could also analyze with regard to (M1) with linear finite elements (P1) in the framework of this study.

Whereas in the previous case of the unit square all coarse grid cells are similar, we have three different groups of similar triangles (isosceles right triangles on the three unit squares and isosceles obtuse and scalene triangles on the unit square with the central hole) in this case (see Figures 15 and 16). Consequently, the matrix C and its inverse consist of three different blocks and the matrices C_i^{-1} need to be computed and stored for i = 1, 2, 3. If L_0 denotes the refinement level of the coarse grid, we have $28 \cdot 4^{L_0}$ macro cells (and thus blocks of C) in total and, using the notation from the end of Section 4.1, $m_1 = 12 \cdot 4^{L_0}$ blocks corresponding to the isosceles right and $m_2 = m_3 = 8 \cdot 4^{L_0}$ blocks corresponding to the isosceles obtuse and scalene macro cells, respectively. It is described at the above-mentioned place how multiplications with C^{-1} are realized efficiently. The resulting amounts of nodes of the different types and condition numbers as well as nonzero entries (relative to the number of nonzeros in the corresponding standard finite element stiffness matrix) of the matrices occurring in (M1) are listed in Table 4. Since there are three different matrices C_i, we consider the maximum condition number and the summed up number of the nonzero entries of the C_i^{-1}. The results indicate that the new approach (M1) is also practicable for this type of mesh.

Another application of the method (M1) are meshes with high aspect ratios, such as anisotropic or skewed meshes. The convergence behavior of iterative multigrid methods gets significantly worse in this case, whereas the behavior of the direct method is hardly affected if such meshes are used.

5.2. Multiple right-hand sides

The presented direct method can also be used to accelerate time-dependent flow simulations modeled by the incompressible Navier–Stokes equations. To give an application example of this, we refer to operator-splitting methods, in particular discrete projection methods, that require the solution of a pressure Poisson problem in each time step. Only the right-hand sides of the linear systems change if the spatial mesh does not vary. A conjunction with currently examined time-simultaneous approaches (Dünnebacke et al., 2021) yields variants of the above-mentioned projection methods that allow for solving the pressure Poisson

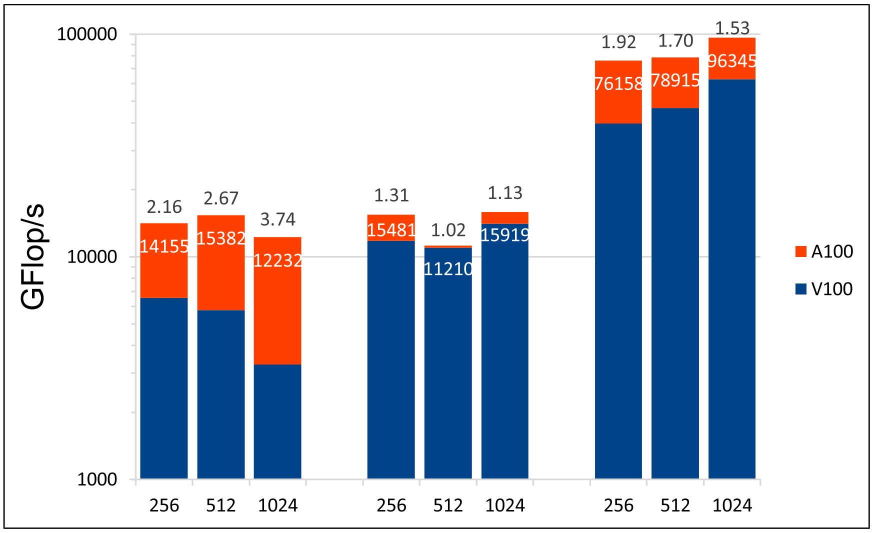

Figure 17. GFLOP/s for (M1) with many RHS depending on h^{-1} on the V100 and the A100 GPU in DP, SP, and HP with Tensor Cores (left, middle, and right three columns, respectively) with speed-up indicated above the columns.

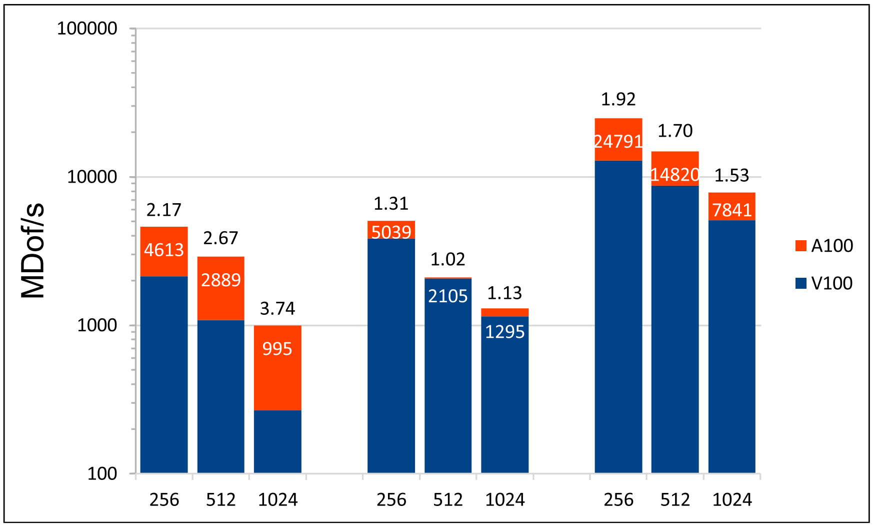

Figure 18. MDof/s for (M1) with many RHS depending on h^{-1} on the V100 and the A100 GPU in DP, SP, and HP with Tensor Cores (left, middle, and right three columns, respectively) with speed-up indicated above the columns.

problems for all time steps simultaneously. That exactly corresponds to the case of a system $AX = B$ with a constant stiffness matrix A and a matrix of solutions X and right-hand-sides B with $N_{\mathrm{rhs}} \gg 1$ columns, in which the highest performance is reached in our tests.

6. Summary and outlook

The purpose of this work was the development and numerical study of a new hardware-oriented algorithm based on knowledge of modern special hardware. As a result, we are able to combine high numerical efficiency with the computational performance of the V100 GPU and its Tensor Cores.

Clearly, our studies are preliminary and only cover the case of Poisson's equation so far. However, this problem represents a central component of many numerical methods for flow simulations using the time-dependent incompressible Navier–Stokes equations. The additional use of parallelism in such basic components of advanced simulation software for applications in numerical continuum mechanics can reduce the computing time by orders of magnitude

Of course, further research needs to be carried out on prehandling in 3D, other differential operators and finite element spaces, as well as on semi-direct variants of the Schur complement approach, such as the method (M3) we also derived. The list of interesting open problems also includes analysis of the new method on other current GPUs. The presented studies, showing the possible gain in efficiency in mathematical simulation tools (especially in numerical continuum mechanics), prove to be even more important considering that the supercomputer JUWELS at Forschungszentrum Jülich, Germany ranked eighth in the current TOP500 list (June 2021) as the fastest European computer system (TOP500, 2021). Its booster module is based on the NVIDIA Ampere A100 Tensor Core GPU, the successor model of the V100. The A100 GPU promises more than 300 TFLOP/s with Tensor Cores in half precision (NVIDIA 2021) which becomes (nearly fully) exploitable by means of prehandling techniques as the results of preliminary tests presented in Figures 17 and 18 show. The results shown inside the columns were obtained with the A100 and can be compared with the values shown in Figures 11 and 12. Indeed, we obtain a further acceleration by approximately 50%.

Acknowledgements

All comparative results presented in Section 5 have been created using the FEM software package FEAT3[2], whereby calculations have been carried out on the LiDO3[3] cluster at TU Dortmund University. We gratefully acknowledge the extensive programming work of Heiko Poelstra during the preparation of this study.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Dustin Ruda https://orcid.org/0000-0003-4252-3099

Notes

1. see https://www.mathematik.tu-dortmund.de/~featflow/en/software/feat3.html
2. see http://www.mathematik.tu-dortmund.de/~featflow/en/software/feat3.html.
3. see www.lido.tu-dortmund.de/cms/en/home/index.html.

References

Abdelfattah A, Anzt H, Boman EG, et al. (2021) A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The International Journal of High Performance Computing Applications* 35(4): 344–369. DOI: 10.1177/10943420211003313.

AMD (2021) AMD EPYC 7542 datasheet. Available at: www.amd.com/de/products/cpu/amd-epyc-7542 (accessed on 14 July 2021)

Deuflhard P, Leinen P and Yserentant H (1989) Concepts of an adaptive hierarchical finite element code. *IMPACT of Computing in Science and Engineering* 1(1): 3–35. DOI: 10.1016/0899-8248(89)90018-9.

Dünnebacke J, Turek S, Lohmann C, et al. (2021) Increased space-parallelism via time-simultaneous Newton-multigrid methods for nonstationary nonlinear PDE problems. *The International Journal of High Performance Computing Applications* 35(3): 211–225. DOI: 10.1177/10943420211001940.

Haidar A, Bayraktar H, Tomov S, et al. (2020) Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 476: 20200110. DOI: 10.1098/rspa.2020.0110.

Kronbichler M and Ljungkvist K (2019) Multigrid for matrix-free high-order finite element computations on graphics processors. *ACM Transactions on Parallel Computing* 6(1): 1–32. DOI: 10.1145/3322813.

NVIDIA (2020) NVIDIA V100 Tensor Core GPU (SXM2) datasheet. Available at: https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf (accessed 14 July 2021)

NVIDIA (2021) NVIDIA A100 tensor Core GPU datasheet. Available at: www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/a100-80gb-datasheet-update-nvidia-us-1521051-r2-web.pdf (accessed on 14 July 2021)

Ong MEG (1997) Hierarchical Basis Preconditioners in Three Dimensions. *SIAM Journal on Scientific Computing* 18(2): 479–498. DOI: 10.1137/S1064827594276539.

Oo KL and Vogel A (2020) Accelerating geometric multigrid preconditioning with half-precision arithmetic on GPUs. arXiv:2007.07539 DOI 10.48550/arXiv.2007.07539.

Poelstra H (2019) *Hardwareorientierte Mehrgitterlöser für Poisson-artige Differentialgleichungen auf verallgemeinerten Tensorprodukt-Gittern*. Master Thesis, TU Dortmund University, Germany.

Ruda D (2020) *Numerische Studien zur "Vorbehandlung" (prehandling) von Poisson-artigen Problemen durch die hierarchische Finite-Elemente-Methode*. Master Thesis, TU Dortmund University, Germany.

Ruda D, Turek S, Zajac P, et al. (2021) The concept of prehandling as direct preconditioning for poisson-like problems, *Lecture Notes in Computational Science and Engineering*. In: Numerical mathematics and advanced applications ENUMATH 2019. Lecture notes in computational science and engineering (eds Vermolen FJ and Vuik C), vol 139, Egmond aan Zee, The Netherlands, September 30 - October 4, 2019, pp. 1011–1019. Cham: Springer International Publishing. DOI: 10.1007/978-3-030-55874-1_100.

Ruelmann H, Geveler M, Ribbrock D, et al. (2021) Basic machine learning approaches for the acceleration of PDE Simulations and Realization in the FEAT3 Software, *Lecture Notes in Computational Science and Engineering*. In: Numerical mathematics and advanced applications ENUMATH 2019. Lecture notes in computational science and engineering (eds Vermolen FJ and Vuik C), vol 139, Egmond aan Zee, The Netherlands, September 30 - October 4, 2019, pp. 449–457. Cham: Springer International Publishing. DOI: 10.1007/978-3-030-55874-1_44.

Saad Y (2003, Iterative methods for sparse linear systems. In: *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA: SIAM. DOI: 10.1137/1.9780898718003.

TOP500 List June 2021 (2021). Available at: www.top500.org/lists/top500/2021/06/(accessed on 14 July 2021)

Yserentant H (1986) On the Multi-Level Splitting of Finite Element Spaces. *Numerische Mathematik* 49: 379–412. DOI: 10.1007/BF01389538.

Author biographies

Dustin Ruda completed his master's degree in Mathematics at TU Dortmund University in 2020, where he has since been working as a PhD student at the Chair of Applied Mathematics and Numerics. His research interests include the adaptation of finite element approaches for elliptic problems and construction and analysis of solution methods that allow the use of fast low precision hardware.

Stefan Turek obtained his PhD in Mathematics from the University of Heidelberg and has been regular professor for Applied Mathematics and Numerics at TU Dortmund University since 1999. He has published more than 200 papers/conference contributions in the field of numerics for PDEs and finite element methods, multigrid solvers, computational fluid dynamics, and high performance computing. Besides, he is strongly involved in developing basic FEM software (FEAT) and the open-source CFD package FEATFLOW (www.featflow.de). Recently, the main research topics of his group of PhD students, postdocs, and collaborators is computational fluid dynamics and the development and realization of simulation techniques and massively parallel HPC software including hardware-oriented numerics and machine learning on special hardware (GPU, TPU) for technical simulations in continuum mechanics and life science.

Dirk Ribbrock studied Computer Science at TU Dortmund University and finished his diploma thesis in 2010. Since then, he has been working as a research assistant at the Chair of Applied Mathematics and Numerics. His research interests are hardware-oriented numerics and parallelization of numerical applications, especially via GPU.

Peter Zajac finished his mathematics diploma thesis in 2011 at TU Dortmund University and is employed at the faculty of mathematics of TU Dortmund since then, where he is one of the main educational staff responsible for the teaching of programming languages. He is the lead developer of the Finite Element Analysis Toolbox 3 (FEAT3) and is responsible for its continuous kernel development as well as for the introduction and support of new and existing programmers and users to this software. His main research fields are the development of geometric multigrid methods for large parallel clusters as well as the construction and analysis of finite element spaces.