Analysis of Block Stokes-Algebraic Multigrid Preconditioners on GPU Implementations

N. M. Evstigneev(✉)

Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia
evstigneevnm@yandex.ru

Abstract. The Stokes and Oseen problems are saddle-point problems common to many methods aimed at the efficient solution of incompressible flows. There are basically two methods for solving saddle problems – coupled (solution of the whole system) and segregated (fractional step method). In this paper, we consider the first approach as the most efficient for steady state problems, in the form of block-triangular preconditioning and its variants. The most difficult part is the design of an efficient preconditioner for the saddle-point problem. Different variants of preconditioning and formulations of pressure Schur complement approximate matrices are considered. The main question is: Given that computations on Graphics Processing Units (GPUs) are cheaper and less energy demanding than computations only on Central Processing Units (CPUs), can an efficient preconditioner be implemented in GPU-only calculation mode? We apply these preconditioners with the most advanced Algebraic Multigrid methods (AMG) based on the AMGCL framework developed by D. Demidov. The AMGCL framework is extensively modified for the purpose of testing in GPU-only calculation modes. To formulate the Stokes problem, we use the classical MAC method on a staggered grid and consider different types of 3D problems. It is concluded that GPU-only computations can be approximately 3–4 times more efficient than CPU+GPU implementations and about 20 times more efficient than CPU-only implementations of the original AMGCL framework.

Keywords: Saddle-point problems · Stokes equations · Algebraic Multigrid methods · General Purpose GPU computations · Iterative methods · Pressure Schur complement

1 Introduction

The solution of Stokes-type linear systems is a problem that is considered complicated [3]. The original Stokes system is derived from the fluid dynamics of an incompressible viscous fluid when flow dynamics is such that the nonlinear

The study was funded by the Russian Foundation for Basic Research, project No. 20-07-00066.

L. Sokolinsky and M. Zymbler (Eds.): PCT 2022, CCIS 1618, pp. 116–130, 2022.
https://doi.org/10.1007/978-3-031-11623-0_9

effects of the advective terms can be neglected. In this case, the temporal scale becomes proportional to the scale of length squared, and the solution of the stationary problem is mainly of interest. The discretization of the problem can be formulated in two ways: without stabilization and with stabilization. In both cases, it is necessary to solve a saddle-point linear system in the form:

$$\mathcal{A}x = b \Leftrightarrow \begin{pmatrix} A & B^T \\ B & C \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{1}$$

In the first case, one arrives at the symmetric indefinite linear system that approximates the saddle-point problem where the approximation satisfies the Ladyzhenskaya–Babushka–Bretzi (LBB) or *inf-sup* condition, and $C = 0, g = 0$ in (1). In the case of finite differences (FDM) this is the well-known staggered grid [19], and in the case of finite elements (FEM) these are the unequal order finite element pairs [25] of different design. In the second case, one arrives at the indefinite problem with an arbitrary approximation and some way of stabilization, e.g. Rhie–Chow stabilization for FDM [24,28] or Brezzi–Pitkaranta stabilization for FEM [5], in this case $C \neq 0, g \neq 0$. There exist many different methods for solving the problem using different approaches. The resulting system can be factored to obtain the Schur complement matrix $S = C - BA^{-1}B^T$, which can be used to find the variable p. Depending on the way this system is solved, two approaches are possible:

- Segregated approach: 1. Solve for $p = \hat{S}^{-1}(g - BA^{-1}f)$, 2. solve for $u = A^{-1}(f - B^T p)$, each with the iterative method and an appropriate preconditioner, where $\hat{S}^{-1}$ is an approximation to the Schur complement inverse.
- Coupled approach: Solve the whole system $\mathcal{A}x = b$, usually with the Krylov-type method and an appropriate preconditioner for the saddle-point system.

The advantages and disadvantages of each approach are discussed in [3,13] in detail. In this research, we focus on the coupled system solution using different variants of preconditioners based on the Algebraic Multigrid Method (AMG). Such methods are efficient and, if correctly constructed, are close to optimal [12,18,21,27,29] in terms of the number of iterations and grid diameter independence. However, there is still no universal preconditioner that can solve a wide range of different problems, especially for parallel computational architectures. Such methods can be constructed in two different approaches: separate velocity and pressure preconditioning using the Schur complement matrix approximation [15,22] and Vanka-type preconditioning, which can be considered as a global symmetric block Gauss–Seidel iteration process over all discretization cells [26]. In this paper, we focus on the first approach, leaving the description of the Vanka smoother implementation elsewhere. In this case, the following system is solved:

$$\mathcal{P}^{-1}\mathcal{A}x = \mathcal{P}^{-1}b, \tag{2}$$

where $\mathcal{P}$ is the coupled left preconditioning operator. The formulation and inversion of this preconditioner can be done in different ways, including block pressure correction or bock triangular approaches. On each step, it is necessary to solve

linear systems for the approximate matrix $\hat{S}$ and matrix A, which can be applied as exact or approximate factorization. In this study, we apply the AMG method.

The computational platform for the AMG method in this research is the AMGCL library developed by D.Demidov [8,9,11]. Our initial attempt to use the AMGX library failed [16], and the AMGCL framework was used instead. It is a header-only template library written in C++, which is very programmer-friendly and has many implemented features and methods, including GPU support. The library is also oriented on solving challenging linear systems, including Stokes-type systems, see [10]. The results obtained in the cited paper look promising. However, the idea of the AMGCL implementation is aimed at using a host-assembled matrix in the CPU memory. All methods constructing AMG hierarchies, prolongation and restriction operators are based on built-in matrix arrays that operate on CPU OpenMP or MPI parallel architectures. Hence, the framework cannot be used in GPU-only mode, where the system matrix $\mathcal{A}$ is stored in the device memory, without intermediate host-device memory transforms.

The analysis of different preconditioning strategies and approximate Schur complement matrices is performed. The main question is: Given that computations on Graphics Processing Units (GPUs) are cheaper and less energy demanding than computations only on Central Processing Units (CPUs), can an efficient preconditioner be implemented in GPU-only calculation mode? To achieve this goal, the AMGCL library is modified in order to execute the GPU-only mode when the matrix is formed in the device memory. The paper is laid out as follows. First, the problem formulation and different types of approximate Schur complement matrices are considered. Next, the AMG algorithm modifications of the GPU-only approach are described. Then, test problems and numerical experiments are conducted for different preconditioners and compared with each other and with the original AMGCL implementation.

2 Problem Formulation

2.1 Discretization and Formulation of Preconditioners

The nondimentionalized Stokes problem in the domain $\Omega \subset \mathbb{R}^3$ with the piecewise smooth Lipschiz boundary $\delta\Omega$ is described by the following system:

$$\begin{aligned} -\triangle \mathbf{v} + \nabla p &= \mathbf{f}, \\ \nabla \cdot \mathbf{v} &= 0, \\ \mathbf{v}|_{\delta\Omega_D} &= \mathbf{v_d}, \\ (\nabla \mathbf{v} - pI) \cdot \mathbf{n}|_{\delta\Omega_N} &= \mathbf{g}. \end{aligned} \tag{3}$$

The vector function $\mathbf{v}$ is commonly referred to as the velocity, the scalar function p as the pressure, the known vector function in the right-hand side $\mathbf{f}$ as the source term, the subscripts D and N represent the Dirichlet and Neumann boundary conditions, respectively, and the vector $\mathbf{n}$ represents the outward normal vector

on the boundary. This problem, despite being linear, is relatively computationally difficult. The solution pair $(\mathbf{v}, p)$ of (3) is not the minimizer of a quadratic functional, as it would be in the case of an elliptic-type Partial Differential Equation (PDE), but of a saddle point. The pressure function can be considered as the Lagrange multiplier for the original diffusive system being subject to the incompressibility constraint. Assuming that the functions under consideration are smooth, one can apply discretization to problem (3) to form discrete system (1). The main target of the research is the solution of linear system (2). The linear system is solved using the GMRES method, where the action of the preconditioner is supplied via the assembled hierarchies of AMG solvers. The first preconditioner tested is a block triangular preconditioner, which is applied as the following composition:

$$\mathcal{P}^{-1} = \begin{pmatrix} A & B^T \\ 0 & \hat{S} \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & -A^{-1}B^T\hat{S}^{-1} \\ 0 & \hat{S}^{-1} \end{pmatrix} = \begin{pmatrix} A^{-1} & 0 \\ 0 & E \end{pmatrix} \begin{pmatrix} E & -B^T \\ 0 & E \end{pmatrix} \begin{pmatrix} E & 0 \\ 0 & \hat{S}^{-1} \end{pmatrix}. \tag{4}$$

Each element of the matrix composition is applied using an AMG solver sweep (single V or W-cycle). The second preconditioner is a variant of the Braess–Sarazin preconditioner [4], which is applied as the following application:

$$\mathcal{P}^{-1} = \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix}^{-1} = \begin{pmatrix} \left(A^{-1} - A^{-1}B^T\hat{S}^{-1}BA^{-1}\right) & \left(A^{-1}B^T\hat{S}^{-1}\right) \\ \left(\hat{S}^{-1}BA^{-1}\right) & \left(-\hat{S}^{-1}\right) \end{pmatrix}. \tag{5}$$

Having the residual vector of the coupled problem $r = (r_u, r_p)^T = \mathcal{A}x - b$, one usually applies the pressure solver first and then reuses the result in the velocity solver to minimize the number of solver applications in (5) as:

$$\begin{aligned} \hat{S}p &= BA^{-1}r_u - r_p \\ Au &= r_u - B^T p. \end{aligned} \tag{6}$$

In any case, both preconditioners allow one to apply pressure and velocity solvers separately. These approaches are implemented in the AMGCL Stokes preconditioner [10]. For each preconditioner it is necessary to define different forms of approximate A, $\hat{S}$ matrices. In this study, we use the following matrix approximations. The velocity matrix A is inverted via AMG sweeps using either V or W cycles.

The Schur complement pressure matrix in its exact form is prohibitively expensive both in terms of the computation time and device memory occupancy (due to the usage of the inverted velocity matrix). Instead, we apply two different strategies to construct the matrix. First, the approximation of Carriere and Jeandel [7] is used:

$$\hat{S} = B(\mathrm{diag}(A))^{-1}B^T, \tag{7}$$

where $\mathrm{diag}(A)$ is the diagonal of the velocity matrix A. The preconditioning with this matrix is applied through a single AMG sweep (V or W cycle) using the Sparse approximate inverse (SPAI0) [6] or Damped Jacobi smoother, which

we call SIMPLE_amg. Another option is only a single application of the SPAI0 preconditioner formed from this matrix, which we call SIMPLE_spai0.

Another variant is the application of a weighted BFBt preconditioner of Elman [14] based on the concept of approximate commutators. In this case, the approximation to the inverse Schur complement is formulated as:

$$\hat{S}^{-1} = (\widetilde{B}\widetilde{B}^T + \widetilde{C})^{-1}\widetilde{B}\widetilde{A}\widetilde{B}^T(\widetilde{B}\widetilde{B}^T + \widetilde{C})^{-1}, \tag{8}$$

where the matrices with tilde are designated as weighted matrices, i.e. $\widetilde{C} = CM_d^{-1/2}$, $\widetilde{B} = BM_d^{-1/2}$, $\widetilde{A} = M_d^{-1/2}AM_d^{-1/2}$, and M_d is the diagonal of the velocity mass matrix for finite element discretization. In the case of finite difference discretization, non-scaled matrices are used [14], however, we apply block diagonal scaling as in [20]. According to our observations, such scaling slightly increased the convergence speed. The application of this approximate matrix can be performed in two different strategies. The first one is the application of a single sweep (V or W cycle) of the AMG preconditioner to each solution of the $\widetilde{B}\widetilde{B}^T$ equation, which we call BFBt_amg. The other option is the exact solution of this equation using the GMRES or CG method with the AMG preconditioner, which uses a single sweep with the SPAI0 smoother. We call this strategy BFBt_exact.

2.2 Discretization and Considered Problems

A simple staggered finite difference scheme discretization [19] is used in this study. Grid functions with indexes (j, k, l) in the (x, y, z) directions, respectively, are introduced and labeled identically to the continuous functions. Grid functions will be explicitly identified, if needed, to escape ambiguity. A single block of variables is formulated as $ux_{j-1/2,k,l}, uy_{j,k-1/2,l}, uz_{j,k,l-1/2}, p_{j,k,l}$. In this case, the formed matrix A is a 7-banded scalar matrix, and the matrices B and B^T each have 4 bands. In the case of boundary conditions, where interior walls present, the diffusion operator is discretized in such a way that the zero value is prescribed on the boundary of the wall. Inactive cells, where no flow occurs, are removed from the process of the matrix assembly.

This simple discretization is selected because it allows testing large matrices with a minimal number of nonzero elements. It is computationally convenient in a parallel processing and multigrid context since they hold an (almost) identical degree distribution for both the velocity and pressure and represent the lowest possible order approximation, which is LBB-stable. Several problems are formulated to verify the convergence of discrete system (3). Besides, the usage of some Finite Element discretization would require the application of block matrix operations, which are not yet implemented in our GPU AMG variant.

Problem 1. Stokes Lid Driven Cavity. The problem is formulated as a classical 3D lid driven cavity [1]. The aim of the test is to verify the convergence properties of the method for the problem with singularities in the corners, as

well as a nontrivial kernel for the Schur complement matrix: $\ker(S) = \text{span}(\mathbf{1})$. The domain is set to a unit cube $[0,1]^3$, zero Dirichlet boundary conditions are prescribed for the velocity on all plains, except one, where the unit tangential value is set.

Problem 2. Flow in a Channel with Obstacles. This problem poses some convergence complications. The part of the internal field (that corresponds to the internal walls) is excluded from the simulation. We also modify the aspect ratio of the domain in this problem. The boundary conditions are set as a unit streamwise pressure gradient and Neumann boundary conditions for the velocity vector. In other directions, the no-slip condition is set.

Problem 3. Flow in a Porous Medium. The last problem is taken from https://www.digitalrocksportal.org/projects/374/origin_data/1785/ as an example of a real world application. The problem is a unit cube $[0,1]^3$, the discretization of the original problem is 256 points in each direction, the porosity is 35%. This test verifies the performance and convergence of the method for a complex real world application. The boundary conditions are the same as for Problem 2.

3 AMGCL Framework Modifications

The process of the general preconditioned solution of the coupled Stokes system can be described by provided Algorithms 1, 2, which are implemented in AMGCL. The velocity solver is related to the preconditioner used to apply the system solve on A, and the pressure solver is related to the preconditioner used to apply the system solve on $\hat{S}$. In this implementation, the coupled system is solved using the GMRES method with a Krylov subspace size of 20 (parameter R in the SOLVE_SYSTEM call). In each iteration of the GMRES method, a preconditioner is called; it is applied by either process (4) or (6).

Algorithm 1. Setup phase

```
1: function SETUP_PRECONDITIONER(A, p_ind)
2:     {A, B, B^T, C} ← CUT_MATRICES(A, p_ind);
3:     divn ← FORM_VECTOR_DIVISION(p_ind);
4:     U ← FORM_VELOCITY_SOLVER(A);
5:     P ← FORM_PRESSURE_SOLVER(A, B, B^T, C);
6:     P ← FORM_PRECONDITIONER(U, P, divn);
7:     return (P);
```

The implementation of these Algorithms is performed in original AMGCL. Methods based on (4), (6) and (7) are readily available in AMGCL. We implemented method (8) separately. However, setup phase Algorithm 1 uses CPU

Algorithm 2. Solve phase

```
1: function SOLVE_SYSTEM(A,P,b, R)
2:     x ← GMRES(R)(A, P, b);
3:     return x;
```

matrices and performs a matrix copy to the host memory if the original matrix is located in the GPU device memory. All operations with the formulation of Schur complement approximates (7) are also performed using CPU matrices. The stages for building AMG hierarchies (if needed) in FORM_VELOCITY_SOLVER and FORM_PRESSURE_SOLVER require the utilization of significantly serial algorithms for AMGCL::COARSENING::PLAIN_AGGREGATES, which are used to construct operators on each level. In addition, some smoothers on the intermediate levels of AMG are also formed on the CPU only. As a result, according to Amdahl's law, the whole speedup is limited to these steps, and can only be speeded up partially using the OpenMP CPU implementation.

To circumvent this problem, it is necessary to reorganize the whole process of constructing AMG hierarchies, to specialize additional methods for CUDA_MATRIX classes and implement CUT_MATRICES and FORM_VECTOR_DIVISION methods on the GPU. Thus, CPU↔GPU memory copies are minimized, and the most time-consuming operations are executed on the GPU only.

To achieve the goal, we used CUDA C++ and templates extensively, since the original AMGCL framework heavily relies on template metaprogramming. The most time-consuming operations during the setup are the construction of additional matrices, the Schur complement approximation $\hat{S}$ and the formulation of pressure and velocity solvers. In turn, each solver can invoke the construction of AMG hierarchies, smoothers and prolongation/restriction operators on each level. The construction of additional matrices is replaced by direct CUDA kernel invokes, as well as by the process of constructing separate and joint residuals $r \leftrightarrow (r_u, r_p)^T$ using DIVN in Algorithm 1. The formulation of the preconditioning approximate matrix $\hat{S}$ is performed using the CUDA_MATRIX specialization for a sparse matrix product.

The construction of aggregates cannot be handled so easily. The serial version available in AMGCL::COARSENING::PLAIN_AGGREGATES cannot be implemented on the parallel architecture. Instead, an updated version of the Parallel Maximal Independent Set K (MIS(K)) is implemented, see [2,17]. An independent set is a set of nodes, in which no two of them are adjacent, and it is a maximal independent set (MIS) if it is not a subset of any other independent set. The generalization of MIS is MIS(K), in which the distance between any two independent nodes is greater than K, and for every other node there is at least one independent node that is within the distance less than or equal to K. Hence, the construction of interpolation operators and smoothing aggregation is performed on the GPU only using the obtained aggregates from the MIS(K) process. The whole redesign of the code resulted in a substantial modification of the original framework. The constructors had to be specified for the GPU-only operation

through all classes. To obtain the maximum speedup on both the original and modified variants of the code, one needs to use the latest CUDA Developer Toolkit, version 11.5, where many functions from the CUSPARSE and CUSOLVER libraries are optimized. In addition, we used our own implementation of matrices and arrays, taking into account the optimization in terms of memory layout depending on the device, and the GPU-optimized SPECK library [23] to perform GEMM (sparse matrix – matrix product) operations. These modifications also boosted the original AMGCL implementation. However, the used SPECK library is not a header only, and it violates the philosophy of original AMGCL. Nevertheless, in this research, we are more interested in performance than in design features.

4 Numerical Results

In this section, we present the numerical results obtained by solving the problems listed above. The following hardware configuration is used: CPU – 2×Intel Xeon Gold 6248R, totally having 48 cores (96 threads) with 512 GB of ECC host memory, GPU – Nvidia Tesla V100 with 32 GB of ECC device memory. Double precision is used in all calculations, no mixed precision was used (which should boost GPU performance even further). All results presented in the research are obtained on the hardware listed above. Ten runs of each test configuration were performed, and mean values were used in all measured tests to obtain statistically justified results. For some problems, we estimate the mean residual reduction rate. The residual reduction rate on the n-th iteration is $\rho^n = \|r^{n+1}\|/\|r^n\|$, where $n \geq 1$, therefore, the initial residual is excluded. The mean residual reduction rate is the averaged value $\rho = \langle \rho^n \rangle$.

4.1 Unit Cube Problem

We used a set of matrices from [10] where the testing of the method with an analytical solution on a unit cube was performed. The target relative residual is set to $1.0 \cdot 10^{-8}$. The problem has a total dimension of $5.5E5$, $2.94E6$ and $4.38E6$ with $1.43E7$, $7.84E7$ and $1.17E8$ nonzero entries in the matrices, respectively. This test was chosen since it has a scalar structure, and block matrices are not yet implemented in our GPU variant. First, an analysis of the optimal number of OpenMP threads for the original implementation for the used hardware is carried out. The results are provided in Fig. 1 for the setup and solve phases of the original AMGCL implementations. These results are obtained for the Block Triangular preconditioner and the SIMPLE_spai0 Schur matrix, the other variants have the same time ratio distributions. One can observe that the speedup reverses from 32 to 64 threads, depending on the matrix size. For further tests, we shall use the best number of OpenMP threads without explicitly demonstrating this value.

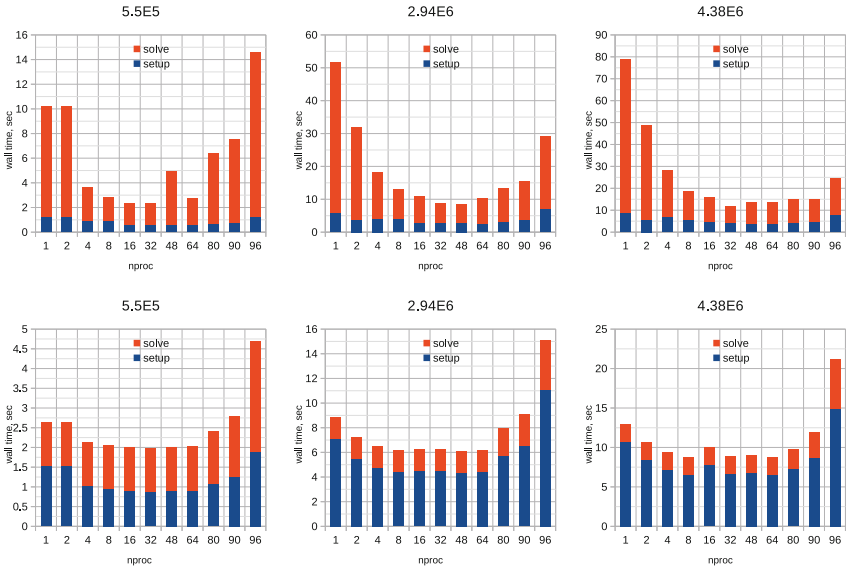

Fig. 1. Setup and solve wall times for the Unit cube problem for different vector sizes on the original AMGCL implementation depending on the number of OpenMP threads. Upper row - CPU, lower row - GPU.

One can also observe in Fig. 1, second row, an obvious bottleneck in the setup/solve phase ratio for the GPU implementation. This situation only confirms the need to implement a GPU-only variant with a full GPU setup phase.

In Fig. 2, upper row, we demonstrate the speedup obtained using the GPU-only variant of AMGCL compared to the original AMGCL GPU variant. It is shown that the setup phase was substantially speeded up. For a single core, we obtained a speedup of 8.29 on the biggest matrix and maximum speedup of 5.1 compared to the 64-core execution with the GPU. The solve phase either did not change or slightly decelerated with the worst value of 0.92 for the problem sized $2.94E6$ compared to the 64-core original implementation. This is because we used a smoothed aggregation based on the MIS(K) algorithm instead of the original aggregation algorithm. As a result, the convergence of the problem to the preset relative tolerance required 1–2 more iterations. In the lower row in Fig. 2, we demonstrate the wall time of different preconditioners. It should be noted that BFBt preconditioners (8) are worse in this problem since the scaling was empirical as no mass matrices are available for this problem. In total, we obtained a speedup of about 2.6–2.9 for the largest matrix compared to the best (OpenMP GPU) original AMGCL implementation. For the pure single-threaded CPU implementation vs. the pure GPU implementation, we achieved a speedup of about 22.2 times on this problem against 7.08 times for the original AMGCL implementations.

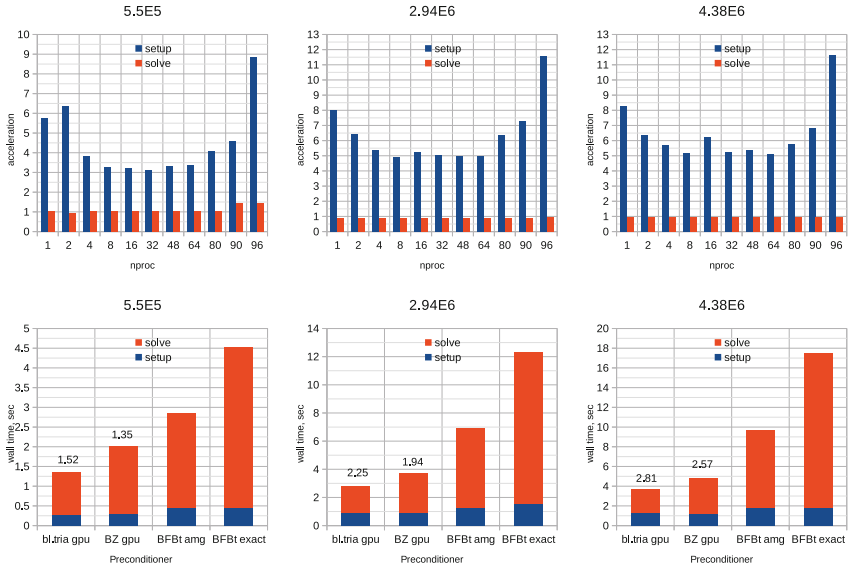

Fig. 2. Upper row: Setup and solve speedup using the new AMG GPU-only implementation vs. the original AMGCL implementation depending on the number of OpenMP threads. Lower row: Wall time for the new GPU-only speedup for different preconditioners. The numbers indicate the speedup vs. the best original implementation.

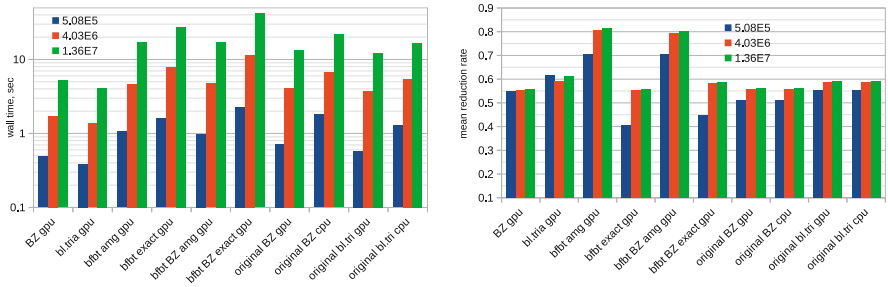

Fig. 3. Wall time in log scale (left) and mean residual reduction rate (right) for Problem 1: bl.tria is block triangular preconditioner (4) and BZ is Braess Sarazin preconditioner (5).

4.2 Stokes Lid Driven Cavity

We constructed three matrices corresponding to the problem domain sized 50^3, 100^3 and 150^3. The total problem sizes are $5.1E5$, $4.03E6$ and $1.35E7$, respectively. The numbers of nonzero elements in the matrices are $6.94E6$, $3.27E7$ and $1.11E8$, respectively. The target relative residual is set to $1.0 \cdot 10^{-8}$. The wall time execution and the mean residual reduction rate are presented in Fig. 3. In all variants, where applicable, we used the SIMPLE_spai0 approximate Schur matrix formulation as the fastest among those tested.

One can observe that the new GPU variant for both the block triangular and Braess Sarazin preconditioners is faster than the best variant of the original AMGCL implementation (GPU + 64 OpenMP threads) for these preconditioners. We obtained a speedup of 1.47, 1.49 for the smallest matrix and 2.92, 3.52 for the largest matrix, respectively. It can be seen that the performance of the preconditioners in terms of the residual reduction rate is different. The new variants of the BZ and bl.tria preconditioners based on the MIS(K) aggregates are slightly worse. The BFBt preconditioner is worse in any variant. This is due to the nontrivial kernel for the Schur complement matrix. Possible remedies can be found in the literature and are beyond the scope of this paper, see [14]. The reduction rate in Fig. 3 on the right indicates that the grid independence was achieved for all preconditioners. For the GPU variant vs. the single-threaded CPU variant, we achieved a speedup of about 25.4 times against 8.1 times for the original AMGCL implementations.

4.3 Flow in a Channel with Obstacles

The problem of a channel with obstacles in different directions is presented with the domain sized 50^3, 100^3, 150^3 and 200^3. The obstacles are rotating planes in different directions that force the flow like a 3D heater, see Fig. 5, left, for the streamline visualization. The total problem sizes are $1.87E5$,$1.84E6$,$6.66E6$ and $1.62E7$, respectively. The numbers of nonzero elements in the matrices are $1.28E6$, $1.41E7$, $5.25E7$ and $1.29E8$, respectively. The target relative residual is set to $1.0 \cdot 10^{-8}$. The resulting wall time execution and the mean residual reduction rate are presented in Fig. 4.

We observe that only the BFBt-type approximation to the inverse of the Schur complement matrix is capable of correctly preconditioning the problem. Only 3 iterations were required for the smallest matrix in the case of the exact BFBt variant. The grid independence was achieved with a mean reduction rate of about 0.55–0.58 for the BFBt_amg variants. The other preconditioners were unable to converge for 300 iterations to the desired tolerance and were terminated. In this case, we can compare the original AMGCL implementation and the new one in terms of speedup for the same number of iterations. In this case, the GPU-only variant is about 1.3–2.0 times faster than the best AMGCL variant (GPU + 64 OpenMP threads) for the largest matrix. This is because more iterations were used where the difference in the implementations was not large. For the GPU variant vs. the single-threaded CPU variant, we achieved a speedup of about 32.18 times against 18.9 times for the original AMGCL implementations.

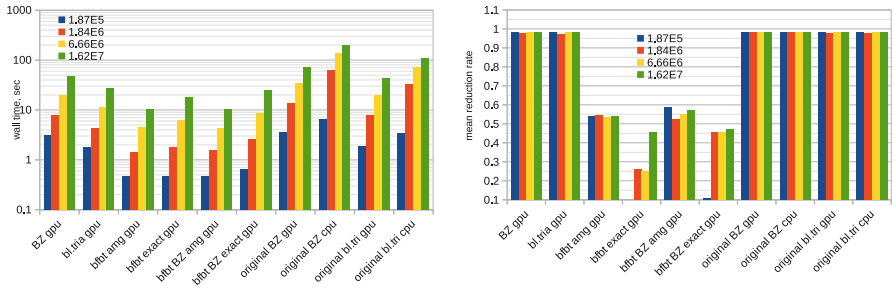

Fig. 4. Wall time in log scale (left) and mean residual reduction rate (right) for Problem 2: bl.tria is block triangular preconditioner (4) and BZ is Braess Sarazin preconditioner (5).

4.4 Flow in a Porous Medium

The total size of the problem is $1.96E7$, and the matrix has $1.54E8$ nonzero entries. The flow is visualized through the streamlines in Fig. 5, right. The target relative residual is set to $1.0 \cdot 10^{-8}$. The execution results are brought together in Table 1. The setup step was speeded up by 5.22 and 4.28 times for the GPU vs. CPU variants of the original implementation, respectively. The total speedup is about 1.6–1.8 times since more time was spent on iterations. Again, we can observe that for this type of problems and scalar matrices, the best variant is the BFBt_amg preconditioner. It can also be seen that since the original aggregation method is essentially serial, the speedup of the setup step in the original implementation is about 2.5 for 64 threads.

Table 1. Solution data for the problem of flow in a porous medium.

name	time_setup	time_solve	total_time	iterations	residual	$< \rho >$
BZ gpu	2.68	54.77	57.45	300	6.0E-7	0.97
bl.tria gpu	2.65	30.29	32.94	300	1.7E-6	0.97
bfbt amg gpu	3.46	10.08	13.54	30	7.7E-9	0.58
bfbt exact gpu	3.25	31.06	34.31	35	9.6E-9	0.71
bfbt BZ amg gpu	3.53	11.62	15.15	35	8.7E-9	0.60
bfbt BZ exact gpu	3.21	51.80	55.02	46	7.8E-9	0.76
original BZ gpu 64	13.97	69.46	83.43	300	5.4E-7	0.97
original BZ cpu 64	11.45	297.06	308.51	300	5.4E-7	0.97
original BZ cpu 1	28.22	1,586.05	1,614.27	300	5.4E-7	0.97
original bl.tri gpu 64	12.74	37.73	50.47	300	1.1E-6	0.97
original bl.tri cpu 64	11.34	160.72	172.06	300	1.1E-6	0.97
original bl.tri cpu 1	28.07	1,045.27	1,073.34	300	1.1E-6	0.97

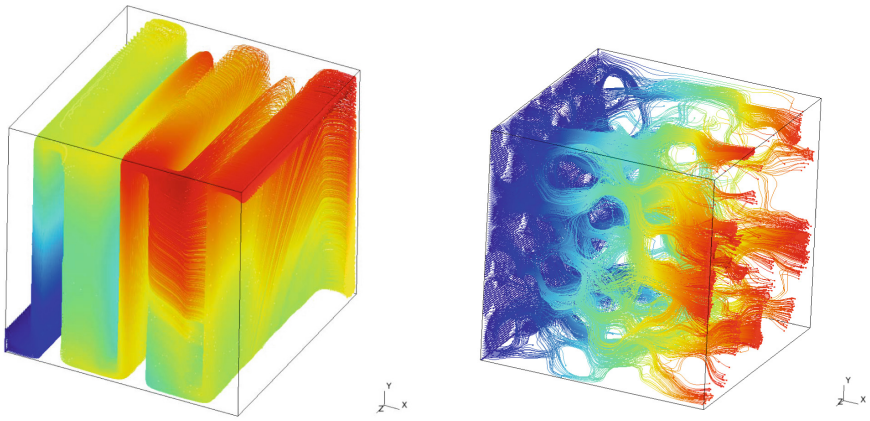

Fig. 5. Streamlines for Problem 2 (left) and Problem 3 (right).

5 Conclusion

We implemented the GPU-only variant of the aggregation AMG method based on the AMGCL implementation. To do it, we needed to change the aggregation algorithm on the parallel version (used MIS(K)) and redesign the whole library in such a way that no intermediate CPU calculations were required. We used CUDA C++ together with template programming to achieve this. We also suggested a scaled variant of the BFBt preconditioner implementation for the approximate Schur complement matrix inverse. The tests show that the suggested GPU-only variant can be approximately 3–4 times more efficient than CPU+GPU implementations and about 20–35 times more efficient than the CPU-only implementations of the original AMGCL framework. This speedup is obtained mostly due to the setup step, which is speeded up by about 5–7 times for large matrices compared to the best variant of the AMGCL configuration. The BFBt preconditioner shows efficient properties for the problems with obstacles. Support for block matrices and multiple GPUs is next to be implemented.

References

1. Albensoeder, S., Kuhlmann, H.: Accurate three-dimensional lid-driven cavity flow. J. Comput. Phys. **206**(2), 536–558 (2005). https://doi.org/10.1016/j.jcp.2004.12.024
2. Bell, N., Dalton, S., Olson, L.N.: Exposing fine-grained parallelism in algebraic multigrid methods. SIAM J. Sci. Comput. **34**(4), C123–C152 (2012). https://doi.org/10.1137/110838844
3. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. Acta Numerica **14**, 1–137 (2005). https://doi.org/10.1017/s0962492904000212
4. Braess, D., Sarazin, R.: An efficient smoother for the stokes problem. Appl. Numer. Math. **23**(1), 3–19 (1997). https://doi.org/10.1016/s0168-9274(96)00059-1

5. Brezzi, F., Pitkäranta, J.: On the stabilization of finite element approximations of the stokes equations. In: Efficient Solutions of Elliptic Systems, pp. 11–19. Vieweg Teubner Verlag (1984). https://doi.org/10.1007/978-3-663-14169-3_2
6. Bröker, O.: Sparse approximate inverse smoothers for geometric and algebraic multigrid. Appl. Numer. Math. **41**(1), 61–80 (2002). https://doi.org/10.1016/s0168-9274(01)00110-6
7. Carriere, P., Jeandel, D.: A 3D finite element method for the simulation of thermo-convective flows and its performances on a vector-parallel computer. Int. J. Numer. Methods Fluids **12**(10), 929–946 (1991). https://doi.org/10.1002/fld.1650121003
8. Demidov, D.: AMGCL: an efficient, flexible, and extensible algebraic multigrid implementation. Lobachevskii J. Math. **40**(5), 535–546 (2019). https://doi.org/10.1134/S1995080219050056
9. Demidov, D.: AMGCL - A C++ library for efficient solution of large sparse linear systems. Softw. Impacts **6**, 100037 (2020). https://doi.org/10.1016/j.simpa.2020.100037
10. Demidov, D., Mu, L., Wang, B.: Accelerating linear solvers for stokes problems with C++ metaprogramming. J. Comput. Sci. **49**, 101285 (2021). https://doi.org/10.1016/j.jocs.2020.101285
11. Demidov, D., Shevchenko, D.: Modification of algebraic multigrid for effective GPGPU-based solution of nonstationary hydrodynamics problems. J. Comput. Sci. **3**(6), 460–462 (2012). https://doi.org/10.1016/j.jocs.2012.08.008
12. Elman, H.: Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics. Oxford University Press, Oxford (2014)
13. Elman, H., Howle, V., Shadid, J., Shuttleworth, R., Tuminaro, R.: A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible navier-stokes equations. J. Comput. Phys. **227**(3), 1790–1808 (2008). https://doi.org/10.1016/j.jcp.2007.09.026
14. Elman, H., Howle, V.E., Shadid, J., Shuttleworth, R., Tuminaro, R.: Block preconditioners based on approximate commutators. SIAM J. Sci. Comput. **27**(5), 1651–1668 (2006). https://doi.org/10.1137/040608817
15. Elman, H.C., Howle, V.E., Shadid, J.N., Tuminaro, R.S.: A parallel block multi-level preconditioner for the 3D incompressible navier-stokes equations. J. Comput. Phys. **187**(2), 504–523 (2003). https://doi.org/10.1016/s0021-9991(03)00121-9
16. Evstigneev, N.M., Ryabkov, O.I.: Application of the AmgX library to the discontinuous Galerkin methods for elliptic problems. In: Sokolinsky, L., Zymbler, M. (eds.) PCT 2021. CCIS, vol. 1437, pp. 178–193. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81691-9_13
17. Gandham, R., Esler, K., Zhang, Y.: A GPU accelerated aggregation algebraic multigrid method. Comput. Math. Appl. **68**(10), 1151–1160 (2014). https://doi.org/10.1016/j.camwa.2014.08.022
18. Gee, M.W., Küttler, U., Wall, W.A.: Truly monolithic algebraic multigrid for fluid-structure interaction. Int. J. Numer. Methods Eng. **85**(8), 987–1016 (2010). https://doi.org/10.1002/nme.3001
19. Harlow, F.H., Welch, J.E.: Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. Phys. Fluids **8**(12), 2182 (1965). https://doi.org/10.1063/1.1761178
20. May, D.A., Moresi, L.: Preconditioned iterative methods for stokes flow problems arising in computational geodynamics. Phys. Earth Planet. Inter. **171**(1–4), 33–47 (2008). https://doi.org/10.1016/j.pepi.2008.07.036
21. Notay, Y.: Algebraic multigrid for stokes equations. SIAM J. Sci. Comput. **39**(5), S88–S111 (2017). https://doi.org/10.1137/16m1071419

22. Olshanskii, M.A., Vassilevski, Y.V.: Pressure Schur complement preconditioners for the discrete Oseen problem. SIAM J. Sci. Comput. **29**(6), 2686–2704 (2007). https://doi.org/10.1137/070679776
23. Parger, M., Winter, M., Mlakar, D., Steinberger, M.: Speck: accelerating GPU sparse matrix-matrix multiplication through lightweight analysis. In: Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM (2020). https://doi.org/10.1145/3332466.3374521
24. Rhie, C.M., Chow, W.L.: Numerical study of the turbulent flow past an airfoil with trailing edge separation. AIAA J. **21**(11), 1525–1532 (1983). https://doi.org/10.2514/3.8284
25. Temam, R.: Navier-Stokes Equations: Theory and Numerical Analysis. Elsevier Science, The Netherlands (1984)
26. Vanka, S.: Block-implicit multigrid solution of navier-stokes equations in primitive variables. J. Comput. Phys. **65**(1), 138–158 (1986). https://doi.org/10.1016/0021-9991(86)90008-2
27. Wabro, M.: Coupled algebraic multigrid methods for the Oseen problem. Comput. Vis. Sci. **7**(3–4), 141–151 (2004). https://doi.org/10.1007/s00791-004-0138-z
28. Zhang, S., Zhao, X., Bayyuk, S.: Generalized formulations for the Rhie-Chow interpolation. J. Comput. Phys. **258**, 880–914 (2014). https://doi.org/10.1016/j.jcp.2013.11.006
29. Zulehner, W.: A class of smoothers for saddle point problems. Computing **65**(3), 227–246 (2000). https://doi.org/10.1007/s006070070008