Structural and Multidisciplinary Optimization (2022) 65:125
https://doi.org/10.1007/s00158-022-03214-z

RESEARCH PAPER

A marker-and-cell method for large-scale flow-based topology optimization on GPU

Jinyuan Liu[1] · Zangyueyang Xian[1] · Yuqing Zhou[2] · Tsuyoshi Nomura[3] · Ercan M. Dede[2] · Bo Zhu[1]

Received: 22 October 2021 / Revised: 23 February 2022 / Accepted: 25 February 2022 / Published online: 29 March 2022

Abstract

The focus of this paper is to propose a novel computational approach for the solution of large-scale flow-based topology optimization problems using a graphics processing unit (GPU). A marker-and-cell method is first used to discretize a fluid flow design domain. This is followed by a finite difference method to solve the Stokes equations for steady-state incompressible fluid flow. An adjoint method is then employed to conduct design sensitivity analysis for the optimization. We use a generalized minimal residual method as the base solver for the linear system and develop an efficient geometric multigrid preconditioner on GPU in a matrix-free form. We simplify the treatment of different boundary conditions with improved accuracy based on the theory of discrete exterior calculus. Numerical results utilizing different resolutions are presented and highlight a nearly linear computational time scalability. Consequently, intricate branching flow structures may be automatically and efficiently discovered at high resolutions. Our approach is capable of solving indefinite problems (i.e., one forward solution of the Stokes equations) with over 7 million elements in three dimensions (3D) and over 16 million elements in two dimensions (2D) within two minutes using a single desktop computer. Furthermore, all numerical experiments reported in this paper are performed on a single NVIDIA Quadro RTX 8000 graphics card. We subsequently compare the optimized flow structures obtained using the newly proposed method with those obtained by commercial finite element software in an established optimization loop and find the optimized structures from both methods to be in good agreement. To highlight the advantage of GPU acceleration, a quantitative run-time comparison study with the commercial finite element software is performed. Our implementation is shown to solve fluid flow problems with orders of magnitude higher resolution using only a fraction of the computational time.

Keywords Topology optimization · Stokes flow · Geometric multigrid on GPU · Finite difference method · Discrete exterior calculus

1 Introduction

Topology optimization has demonstrated its effectiveness in generating creative designs with superior properties in a variety of scientific and engineering applications, such as 3D printing and precision manufacturing, to name a few; see Sigmund and Maute (2013), Deaton and Grandhi (2014), Rozvany (2009) for surveys. Starting from a spatial domain uniformly filled with material, a standard topology optimization algorithm iteratively searches for the optimal material distribution for some design objectives, given the prescribed volume constraint and boundary conditions. Garnering benefits from the recent advances in computing power and efficient algorithms, the resolution of the design domain has been pushed forward to a level of billions of voxels (Aage et al. 2017; Liu et al. 2018).

Responsible Editor: Joe Alexandersen.

Topical Collection: Flow-driven Multiphysics.
Guest Editors: J Alexandersen, C S Andreasen, K Giannakoglou, K Maute, K Yaji.

✉ Ercan M. Dede
eric.dede@toyota.com

[1] Department of Computer Science, Dartmouth College, Hanover, NH 03755, USA

[2] Electronics Research Department, Toyota Research Institute of North America, Ann Arbor, MI 48105, USA

[3] Applied Mathematics Research Domain, Toyota Central R & D Labs., Inc., Tokyo 112-0004, Japan

The above concept can be naturally extended to fluid flow problems. A series of fluid-related metrics may be optimized by reasonably determining optimal flow channels surrounded by solid boundaries; refer to Alexandersen and Andreasen (2020) for a survey. Among all optimization approaches, the density-based methods benefit from the unified treatment of fluid and solid by using permeability, which is insensitive to the drastic change of topology and interface. A challenge with such an approach is that the velocity and pressure fields are present everywhere, which will use a large amount of memory. Thus, memory assignment and reuse need to be properly managed for a high-resolution design space.

In this paper, we present a system for large-scale topology optimization of Stokes flow on a desktop computer equipped with modern GPUs. We develop an efficient GPU-parallelized multigrid preconditioner to solve the indefinite Stokes equations. A performance comparison between our implementation and other commonly used numerical solvers is presented. Matrices are considered as linear mapping functions and represented in matrix-free forms to save memory. The Marker-and-Cell (MAC) scheme is used to discretize the system, which eases the descriptions of differential operators in a coupled way, as well as the sampling processes in the multigrid V-cycle. The validation study and numerical examples provided herein show our algorithm to be fast, stable and capable of automatically discovering complex flow structures.

2 Related work

Pioneered by the work of Borrvall and Petersson (2003), the optimization of fluid-based problems has drawn researchers' interests in a vast number of fields, such as Stokes flow (Guest and Prévost 2006; Aage et al. 2008; Challis and Guest 2009), steady state flow (Zhou and Li 2008; Olesen et al. 2006), unsteady flow (Kreissl 2011; Yaji et al. 2018), turbulence (Dilgen et al. 2018; Papoutsis-Kiachagias and Giannakoglou 2016), compressible flow (Evgrafov 2006; Sá et al. 2021), viscous flow (Kontoleontos et al. 2013), microfluidics (Andreasen et al. 2009), MEMS Maute and Frangopol (2003), functional device (Du et al. 2020) and fluid-structure interaction (Andreasen and Sigmund 2013; Yoon 2010; Vicente et al. 2015), to name a few. A multitude of design objectives have been proposed to optimize a device's fluid mechanical performance, including the power dissipation (Gersborg-Hansen et al. 2005), drag minimization (Kondoh et al. 2012), conjugate heat transfer (Dede 2009, 2012; Alexandersen et al. 2016) and flow rate distribution (Dede et al. 2020), etc. The design and optimization of large-scale fluid systems remains a challenging topic due to the system complexity and very large number of degrees of freedom (DoFs) involved.

Hardware acceleration has been employed to boost the performance of solvers in a number of high-resolution topology optimization implementations. Supercomputing techniques have power to optimize problems involving millions to over a billion elements (Aage et al. 2017, 2015). The main bottleneck for supercomputing is the network traffic across nodes. GPU-based approaches are also subject to such bottlenecks, but they are still popular solutions in speeding up topology optimization programs (Wadbro and Berggren 2009; Schmidt and Schulz 2011; Challis et al. 2014; Wu et al. 2015; Yadav and Suresh 2014; Liu et al. 2018) with better flexibility and accessibility. One shortage of a GPU is the relatively small amount of memory when compared to a CPU. Thus, we require a careful way of allocating and transferring data on a GPU. Solvers fitted with multi-GPU capability have been proposed in recent years to relax this limitation (Herrero-Pérez and Castejón 2021; Martínez-Frutos and Herrero-Pérez 2016). Besides supercomputer and GPU, heterogeneous (CPU/GPU) computing has also been explored (Liu et al. 2016). However, very few of these techniques have been adapted to tackle fluid-related optimization problems.

3 Topology optimization

A schematic of our density-based topology optimization method is shown in Fig. 1, together with some intermediate states that it generates for a representative problem. Our system starts from a rectangular domain discretized using a MAC grid, which is a standard voxel-based approach in computer graphics that allows for refined representation of arbitrary geometry with increased resolution. The allocations of the pressure, P, design variable, ρ, and velocity components, (u, v), for a single grid cell, (i, j), in 2D are shown in Fig. 2. Different kinds of boundary conditions can then be specified by the user, including a no slip wall condition, the magnitude of a pressure jump, as well as the flow rate across the boundary grid faces.

In this section, we discuss the discretized model and implementation in detail. The state variables are obtained by solving the modified Stokes equations subject to boundary conditions using the finite difference method (Chen 2016). The objective is to find an optimized shape such that the fluid flow resistance is minimized under the constraint of a prescribed upper-bound target volume fraction of fluid material. An adjoint-based method is then applied to conduct sensitivity analysis, which is used to update the material distribution.

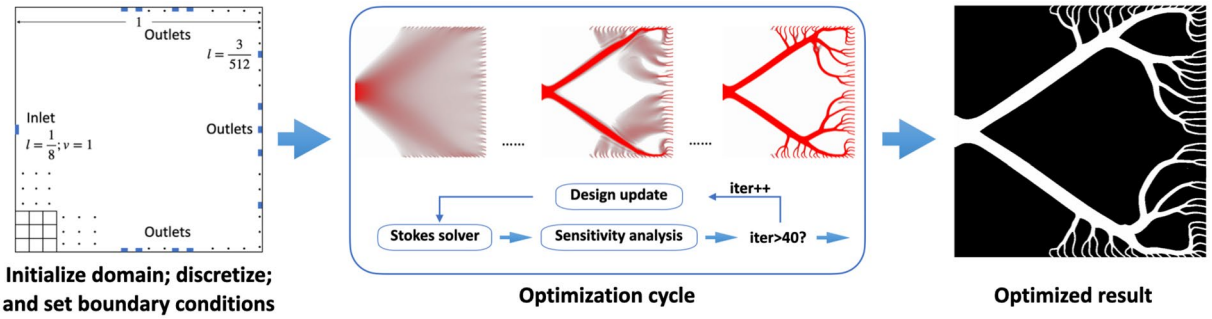

Fig. 1 A pipeline description of a representative system with optimization approach and intermediate topologies

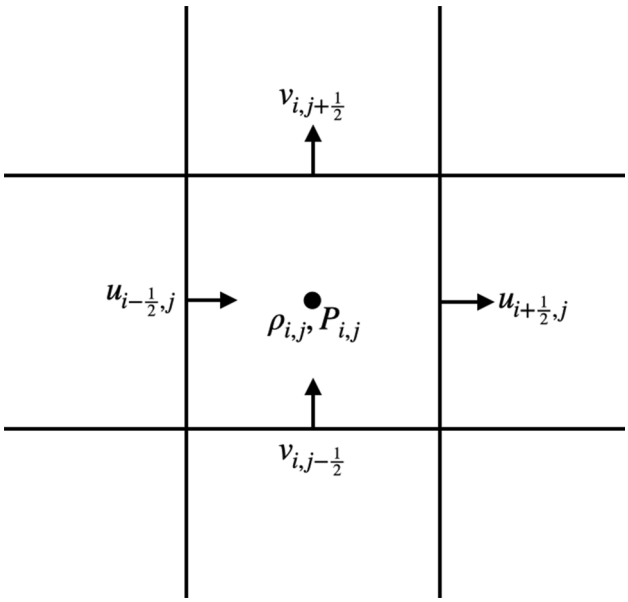

Fig. 2 Marker and cell discretization of variables

3.1 Flow model

The modified Stokes equations using the design variable, ρ, are

$$\begin{aligned}[-\mu\Delta + \alpha(\rho)]\mathbf{u} + \nabla P &= \mathbf{f} \text{ in } \Omega,\\ -\nabla\cdot\mathbf{u} &= 0 \text{ in } \Omega,\\ \mathbf{u} &= \mathbf{u}_b \text{ in } \partial\Omega.\end{aligned} \tag{1}$$

where $\mathbf{u}$, P, and $\mathbf{f}$ are the velocity field, pressure field, and external force, respectively. The specified velocity at the boundary is $\mathbf{u}_b$, while μ is the dynamic viscosity. Δ, ∇ and $\nabla\cdot$ are the Laplace, gradient and divergence operators, respectively. The design variable, ρ, represents the portion of fluid within each grid cell with 1 meaning complete fluid (i.e., void) and 0 meaning complete solid. The term α is a mapping function associated with ρ to map the design variable to inverse permeability:

$$\alpha(\rho) = \bar{\alpha} + (\underline{\alpha} - \bar{\alpha})\rho\frac{1+q}{\rho+q}. \tag{2}$$

in which $q > 0$ is a constant value that controls the shape of the inverse permeability on intermediate values of the fluid's portion. In (2), $\bar{\alpha}$ and $\underline{\alpha}$ are the upper and lower bound of α. This differentiable continuous mapping provides a soft no-penetration condition for solids. For all examples, we set μ to be 1, and $\bar{\alpha}$ and $\underline{\alpha}$ are set to be 10^4 and 10^{-4} respectively. Here, $\bar{\alpha}$ is far larger than the viscosity and thus can represent a non-permeable solid obstacle, but this value is not too large which can make the system of equations too singular and significantly attenuate the solver's convergence speed. Since $\alpha(\rho)$ is multiplied with the velocity which lies on the grid face, we compute the fluid design density variable, ρ, on a face by averaging the values carried by cells on both sides of the face.

The local index system for all state and design variables using the MAC scheme in 2D is shown in Fig. 3. The discretized version of (1) on a non-boundary cell (i, j) can be written as (3) based on the index system (with dynamic viscosity set to 1 for simplicity). Please be aware that the vector Laplacian, $\Delta\mathbf{u} = \nabla(\nabla\cdot\mathbf{u}) - \nabla\times(\nabla\times\mathbf{u})$, has been reduced to the simpler scalar Laplacian, $\Delta u = \nabla\cdot\nabla u$ and $\Delta v = \nabla\cdot\nabla v$, as we decompose the vector field, $\mathbf{u}$, to orthogonal components, u and v. For a boundary cell, the vector Laplacian is used to handle boundary conditions accurately. Here, h is the grid cell size. The first two equations correspond to the velocities in the x direction at the left and right faces, and the next two equations are for the velocities in the y direction. The last equation enforces the incompressibility condition for cell (i, j). Note that for a 2D domain composed of $m\times n$ grid cells, the pressure is discretized as $P(1:m, 1:n)$, and the velocity components are $u(1:m, 1:n+1)$ and $v(1:m+1, 1:n)$. This discrete view can be easily extended to 3D cases. Stacking all the

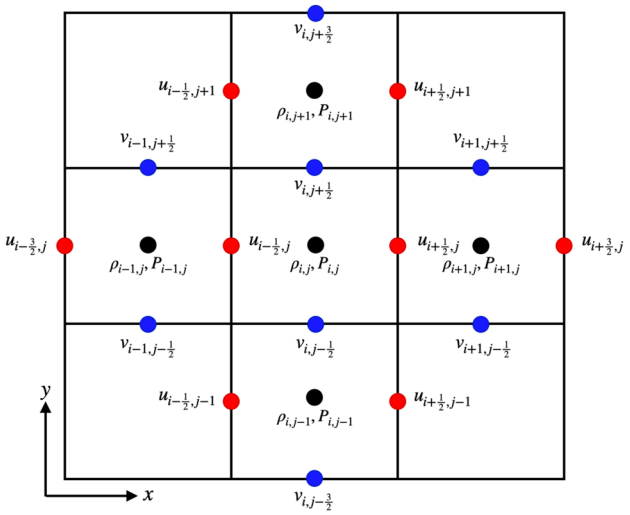

Fig. 3 Indices of variables involved in the local state equations for cell (i, j)

unknown state variables provides a symmetric yet indefinite linear system. Please note that m and n can be defined arbitrarily, but to obtain the best convergence performance, our solver requires that m is equal to n and further equal to some integer power of 2. Further details are explained in Sect. 6.5. The reason for this requirement is that our sampling technique in multigrid is relatively rigid, where we always cut DoFs along each dimension by 2. More flexible sampling techniques can be designed, but that is not a main aim of this paper.

$$
\begin{aligned}
&\frac{4u_{i-\frac{1}{2},j} - u_{i-\frac{3}{2},j} - u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j-1} - u_{i-\frac{1}{2},j+1}}{h^2} \\
&\quad + \frac{\alpha(\rho_{i-1,j}) + \alpha(\rho_{i,j})}{2} u_{i-\frac{1}{2},j} + \frac{P_{i,j} - P_{i-1,j}}{h} = f_{i-\frac{1}{2},j} \\
&\frac{4u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j} - u_{i+\frac{3}{2},j} - u_{i+\frac{1}{2},j-1} - u_{i+\frac{1}{2},j+1}}{h^2} \\
&\quad + \frac{\alpha(\rho_{i,j}) + \alpha(\rho_{i+1,j})}{2} u_{i+\frac{1}{2},j} + \frac{P_{i+1,j} - P_{i,j}}{h} = f_{i+\frac{1}{2},j} \\
&\frac{4v_{i,j-\frac{1}{2}} - v_{i,j-\frac{3}{2}} - v_{i,j+\frac{1}{2}} - v_{i-1,j-\frac{1}{2}} - v_{i+1,j-\frac{1}{2}}}{h^2} \\
&\quad + \frac{\alpha(\rho_{i,j-1}) + \alpha(\rho_{i,j})}{2} v_{i,j-\frac{1}{2}} + \frac{P_{i,j} - P_{i,j-1}}{h} = f_{i,j-\frac{1}{2}} \\
&\frac{4v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}} - v_{i,j+\frac{3}{2}} - v_{i-1,j+\frac{1}{2}} - v_{i+1,j+\frac{1}{2}}}{h^2} \\
&\quad + \frac{\alpha(\rho_{i,j}) + \alpha(\rho_{i,j+1})}{2} v_{i,j+\frac{1}{2}} + \frac{P_{i,j+1} - P_{i,j}}{h} = f_{i,j+\frac{1}{2}} \\
&\frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{h} + \frac{v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}}}{h} = 0
\end{aligned}
\tag{3}
$$

3.2 Optimization formulation

In this study, we use a multi-objective function similar to Gersborg-Hansen et al. (2005) and Challis and Guest (2009) to minimize the energy dissipation across the domain and penalize the leakage of fluid in solid regions. The optimization problem can be formulated as follows:

$$
\begin{aligned}
&\min_{\rho}: J = \frac{1}{2}\int_{\Omega} \alpha(\rho)\mathbf{u}\cdot\mathbf{u} + \frac{\mu}{2}\int_{\Omega} \nabla\mathbf{u} : \nabla\mathbf{u}, \\
&\text{s.t.:} \begin{bmatrix} -\mu\Delta + \alpha(\rho) & \nabla \\ -\nabla\cdot & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ P \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} \quad \text{in } \Omega, \\
&\qquad \mathbf{u} = \mathbf{u}_b \quad \text{on } \partial\Omega, \\
&\qquad \rho_{\mathbf{x}} \in [0,1] \quad \forall \mathbf{x} \in \Omega, \\
&\qquad \int_{\Omega} \frac{\rho}{V_0}\, \mathrm{d}\,\Omega \le \bar{V}.
\end{aligned}
\tag{4}
$$

where V_0 and $\bar{V}$ are the total volume of the design domain and the prescribed fluid material volume fraction target, respectively. For the objective function, J, we discretize the integration of the terms by summing up each individual value throughout the domain. For the first term of the objective function with respect to velocity lying on the grid face, we simply sum over the faces in all dimensions. The fluid design density variable, as before, is averaged by values from the two neighboring cells. The second term of the objective function regarding the velocity gradient tensor is not as trivial. We therefore compute $\nabla\mathbf{u}$ for each grid cell and sum over all cells. For the center cell shown in Fig. 3, the corresponding discretized local gradient is written in (5). This treatment can be naturally extended to 3D cases and handle special types of cells (e.g., edge cells, cells containing boundary faces, etc.).

For the sensitivity analysis, we refer the reader to Appendix 1 for details. We use a standard adjoint method to solve for the objective gradient with respect to the design variables. A general review of this approach is presented in Allaire (2015).

We use a first order method of moving asymptotes (MMA) Jiang et al. (1970) for optimization. Besides the objective and its gradient, we compute the volume constraint and its gradient at each optimization iteration, which are utilized by the optimizer to fit a different problem to solve. Thus, our volume constraint may not be tightly satisfied, but the difference is minimal. We also do not use any filtering technique for the design density field.

$$
\begin{aligned}
\nabla \mathbf{u} &= \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}, \\
\frac{\partial u}{\partial x} &= \frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{h}, \\
\frac{\partial u}{\partial y} &= \frac{u_{i-\frac{1}{2},j+1} + u_{i+\frac{1}{2},j+1} - u_{i-\frac{1}{2},j-1} - u_{i+\frac{1}{2},j-1}}{4h}, \\
\frac{\partial v}{\partial x} &= \frac{v_{i+1,j-\frac{1}{2}} + v_{i+1,j+\frac{1}{2}} - v_{i-1,j-\frac{1}{2}} - v_{i-1,j+\frac{1}{2}}}{4h}, \\
\frac{\partial v}{\partial y} &= \frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{h}.
\end{aligned} \tag{5}
$$

3.3 Post processing

Once the optimization algorithm reaches the maximal iteration number, the final topology represented by a solid/void density field is exported for numerical validation. For 2D examples, we directly draw a grayscale colormap using the density value. For 3D examples, the isosurface between solid and void cells is extracted and further smoothed by a box filter to reduce staircase artifacts resulting from the grid structural representation. Finally, the smoothed 3D model is exported as a binary STL format, which is used in commercial software for validation analysis and comparisons.

4 Multigrid solver

The performance bottleneck in topology optimization is solving the static Stokes equations, (1), to obtain the fluid velocity and pressure state variables. It involves assembling a large, sparse matrix, $\mathcal{A}$, and solving the linear system, $\mathcal{A}\mathbf{X} = \mathbf{b}$, which takes most of the processing time. Direct sparse solvers find the exact solution but are unscalable. Iterative solvers find approximate solutions at a much faster rate. However, effective preconditioners need to be designed and applied to maintain a decent convergence rate especially for high resolutions.

Coupled multigrid methods have been proven as efficient solvers for the incompressible Navier-Stokes equations (Benzi et al. 2005; Wittum 1989), which combine a hierarchy of coarser grids and relaxation schemes (smoothers) to recursively restrict a fine-grid residual to the next coarser grid where a correction term is computed and interpolated back. This leads to a basic grid traversal scheme referred to as the V-cycle outlined in Algorithm 1. In this work, we use a geometric multigrid solver as the preconditioner for a GMRES solver.

We test the performance of our solver by solving Stokes flow passing over an obstacle in both two and three dimensions. We place a circular cylinder at the center of a square 2D channel, as shown in Fig. 4, and a sphere in the center of a cube shaped duct for a similar 3D test case. The central obstacle is modeled as weakly permeable material with large α (set as 10^3 here) instead of a solid boundary condition. For comparison, we also use AMGCL (Demidov 2019) with a VexCL backend which provides CUDA parallelization to solve the same problem, and summarize each solver's performance in Table 1. For AMGCL, preconditioners like AMG fail when having zero diagonal parts in the system matrix, which happens with our pressure unknowns. We follow Demidov et al. (2021) by using a composite Schur Pressure Correction preconditioner together with GMRES as the outer iterative solver. For the velocity part, we use smoothed aggregation and ILU(0) as the coarsening and relaxation strategy. For the pressure part, we use the SPAI-0 smoother. We also use a mixed precision approach and set the smoothing iteration to be 4, which controls the number of GMRES iterations such that restarting is not needed. These setups are mostly borrowed from an AMGCL tutorial study for Stokes-like problems, and have been tested by us to perform best on our system when compared to other settings. We compare times on transferring data from CPU to GPU, solver initialization, and solving. Please note that the behaviours between our solver and AMGCL are different. For transferring data, AMGCL needs to build up the system matrix and transfer it to the GPU, but our solver is matrix free and only needs to transfer the description of the current state, which takes much less time and memory. The initialization between the solvers is also different due to the preconditioning strategy. Our initialization includes propagating the system description from top layer to bottom in the multigrid V-cycle and then updating the preconditioner

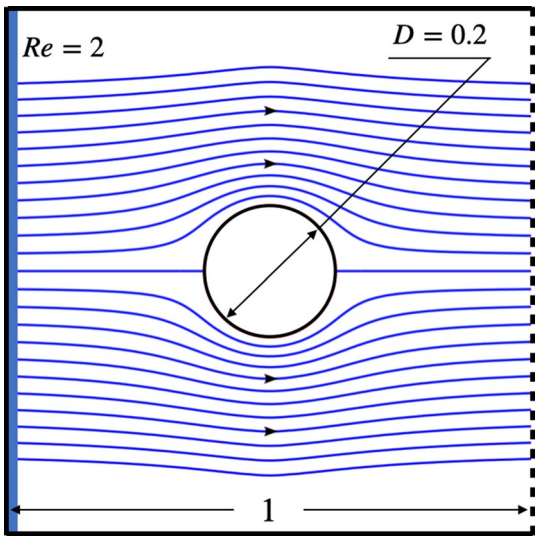

Fig. 4 2D benchmark problem for AMGCL and our solver: Stokes flow passing over a cylinder in a channel

Table 1 Solver performance comparisons (tol=1e−5)

Solver type	Scale	Iterations	Data transfer (ms)	Solver setup (ms)	Solving (s)
SchurPC+ GMRES (AMGCL)	256^2	16	46.3	126.0	5.8
	512^2	23	177.9	441.2	23.6
	1024^2	29	781.5	1627.3	99.4
	32^3	14	31.7	88.6	4.2
	64^3	29	272.2	568.8	24.6
	128^3	31	2345.3	3916.9	168.4
Multigrid+ GMRES (Our solver)	256^2	6	2.06	3.71	0.38
	512^2	6	11.2	6.98	1.23
	1024^2	7	41.0	17.68	6.67
	32^3	5	3.09	21.42	0.29
	64^3	7	44.4	66.3	1.45
	128^3	8	668.1	140.4	11.8

at each layer. Please also note that both solvers are dealing with a badly conditioned system due to the large α. For real Stokes equations, both solvers will be much faster. It turns out that our solver outperforms AMGCL's in both memory footprint and time efficiency.

Algorithm 1 Multigrid V-Cycle

Require: Initial guess x_0 and level K

for $k = 0, ..., K-1$ **do** ▷ Going downward
  $\mathbf{x}_k \leftarrow S_k(\mathcal{A}_k, \mathbf{b}_k, x_k)$ ▷ Pre-smoothing
  $\mathbf{r}_k \leftarrow \mathbf{b}_k - \mathcal{A}_k \mathbf{x}_k$ ▷ Residual
  $\mathbf{b}_{k+1} \leftarrow R_k^{k+1} \mathbf{r}^k$ ▷ Restriction
end for
$\mathbf{x}_K \leftarrow \mathcal{A}_K^{-1} \mathbf{b}_K$ ▷ Exact solution on level K
for $k = K-1, ..., 0$ **do** ▷ Going upward
  $\mathbf{x}_k \leftarrow \mathbf{x}_k + P_{k+1}^k \mathbf{x}_{k+1}$ ▷ Prolongation
  $\mathbf{x}_k \leftarrow S_k(\mathcal{A}_k, \mathbf{b}_k, x_k)$ ▷ Post-smoothing
end for

4.1 Smoothing

The numerical performance of different relaxation methods and multigrid methods for the parallel solution of the incompressible Navier–Stokes equations has been studied (John and Tobiska 2000). It has been proven that to obtain a good smoothing rate, the unknowns need to be updated in a locally coupled manner. We hereby use box-relaxation, a coupled smoother introduced by Vanka (1986), which smooths a group of related velocity and pressure unknowns at once. The box stencil is associated with each pressure unknown, P_i, and the action of the smoother is solving for the updates $\mathcal{A}_i \delta \mathbf{x}_i = \mathbf{r}_i$, $\delta \mathbf{x}_i = [\delta \mathbf{u}_i, \delta P_i]^T$ for each box in a sequential way for all grid cells. Here, $\mathbf{r}_i$ is the residual and $\mathcal{A}_i$ is the local operator restricted from the global matrix, $\mathcal{A}$, to the rows and columns corresponding to the elements in cell i. The size of $\mathcal{A}_i$ is small (5×5 for 2D and 7×7 for 3D). Therefore, direct methods can be used. We further parallelize the sequential smoothing process by coloring the domain as shown in Fig. 5. Due to the MAC scheme depicted before, a five-color scheme is sufficient to perform smoothing in parallel. Cells of the same color can then be updated simultaneously without affecting each other.

4.2 Restriction and prolongation

Other key ingredients of a multigrid method include the matrices R and P that change grids. A restriction matrix, R, transfers vectors from the fine grid to the coarse grid. The return step to the fine grid is done by interpolation using a prolongation matrix. In this section, we introduce restriction and prolongation operators defined on a 2D MAC grid for both scalar and vector variables, which naturally extend to 3D.

The prolongation and restriction of scalar variables that store on the cell centers resemble the operations of node-based variables on a Cartesian grid. In Fig. 6, we show how four variables on the finer red grid, h, (marked with red triangles) are restricted to one variable on the coarser grid, H, (marked as a black cube) and interpolate back. For pressure, P, we use 4-point center restriction

$$P_1^H = \frac{1}{4} P_1^h + \frac{1}{4} P_2^h + \frac{1}{4} P_3^h + \frac{1}{4} P_4^h, \tag{6}$$

and piecewise constant prolongation

$$P_1^h = P_2^h = P_3^h = P_4^h = P_1^H. \tag{7}$$

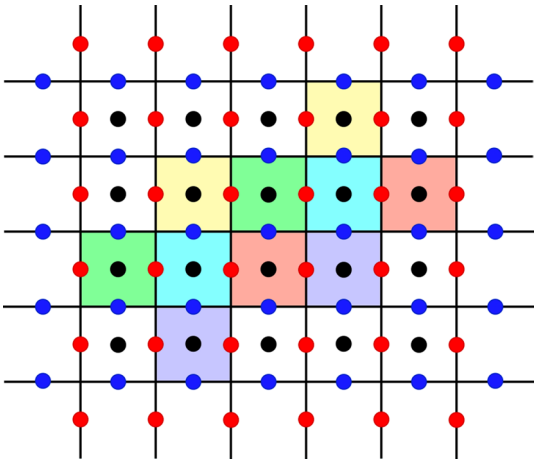

Fig. 5 Parallelization of box-relaxation method on MAC grid using coloring

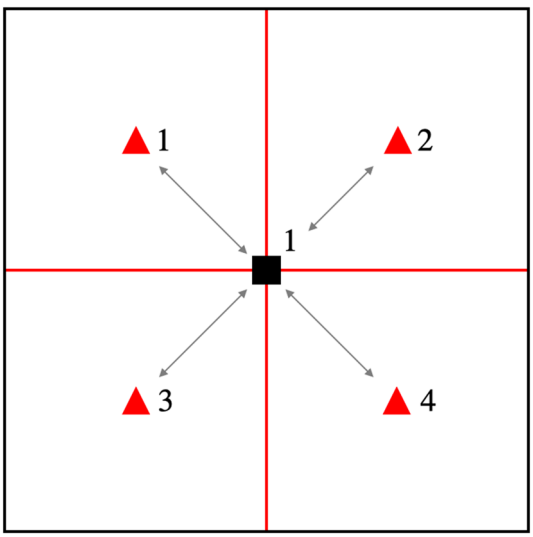

Fig. 6 Restriction and prologation of cell center DoFs

As for the vector variables stored on grid faces, a center restriction operation requires using 6 neighboring points. For the prolongation operators, we apply bilinear interpolation of neighboring coarse grid unknowns in the staggered grid. In Fig. 7, we show how the velocity's horizontal component, u, is restricted and interpolated. Operators for variables stored on other directional faces can be easily derived. Still, the red triangles represent variables at the finer level, h, and the black cubes represent variables at the coarser level, H. The corresponding discrete operators in linear equation form are written as follows:

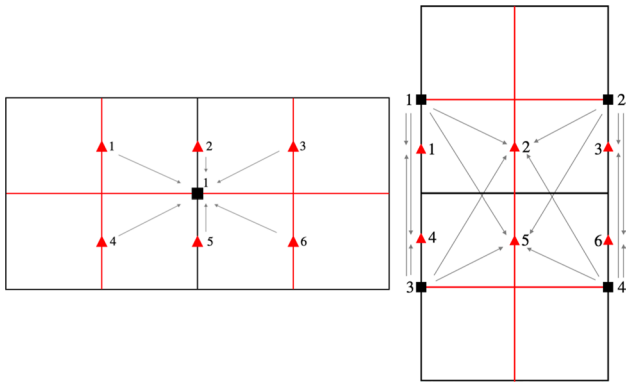

Fig. 7 Restriction (upper) and prolongation (lower) of vertical edge center DoFs

$$
\begin{aligned}
u_1^H &= \frac{1}{8}u_1^h + \frac{1}{4}u_2^h + \frac{1}{8}u_3^h + \frac{1}{8}u_4^h + \frac{1}{4}u_5^h + \frac{1}{8}u_6^h,\\
u_1^h &= \frac{3}{4}u_1^H + \frac{1}{4}u_3^H,\\
u_4^h &= \frac{1}{4}u_1^H + \frac{3}{4}u_3^H,\\
u_3^h &= \frac{3}{4}u_2^H + \frac{1}{4}u_4^H,\\
u_6^h &= \frac{1}{4}u_2^H + \frac{3}{4}u_4^H,\\
u_2^h &= \frac{3}{8}u_1^H + \frac{1}{8}u_3^H + \frac{3}{8}u_2^H + \frac{1}{8}u_4^H,\\
u_5^h &= \frac{1}{8}u_1^H + \frac{3}{8}u_3^H + \frac{1}{8}u_2^H + \frac{3}{8}u_4^H.
\end{aligned}
\tag{8}
$$

4.3 Storage-free matrix vector multiplication

A main benefit of iterative solvers is to avoid computing and storing the matrix $\mathcal{A}^{-1}$ explicitly. Here, we do not store the matrix $\mathcal{A}$ and use a function, $\mathcal{X} \rightarrow \mathcal{A}\mathbf{X}$, instead. This "matrix free" technique is implemented, as follows: first, matrix vector multiplication is reduced to vector vector dot product row by row; next, each row is reduced to a set of non-zero coefficients computed on the fly, and corresponding values are collected from $\mathcal{X}$. A tiny dot product is performed at the end.

We can go one step further given the nature of differential operators and the spatial layout of unknowns in our discretization. By observation, an element in $\mathcal{X}$ stored on either face or cell only has non-zero coefficients on neighbors, as shown in Fig. 8. If we cluster elements by their locations, the values we need to collect from elements in one cluster are highly overlapped, as shown in Fig. 9. This allows us to utilize memory more efficiently and reduce the time cost.

We store the velocity and pressure unknowns in a block memory layout. An example with a 3×3 block size is described in Fig. 9. The block size should be chosen

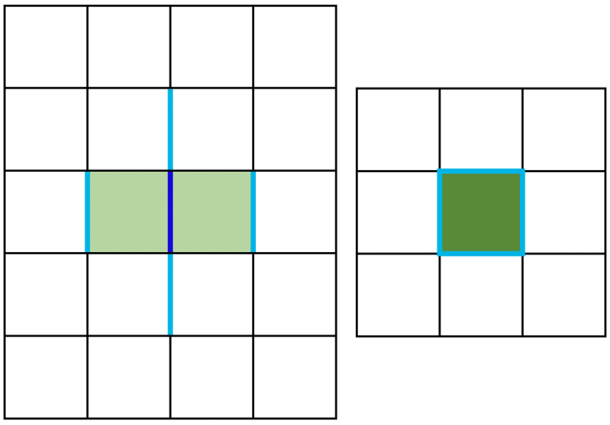

Fig. 8 An example of non-zero coefficients for face (left) and cell (right) element. Dark blue and green represent any face/cell in $\mathcal{X}$. The light blue faces and light green cells represent the corresponding neighbor face/cells with non-zero coefficients. (Color figure online)

carefully such that the memory bandwidth is fully utilized and a sufficient number of threads is assigned in a block to handle the process. In this work, we use 8×8 block in 2D and $4 \times 4 \times 4$ block in 3D. We also use cache/shared memory to store these values such that following arithmetic operations can be accelerated.

5 Implementation

In this section, we describe how the multigrid solver is deployed on a GPU. We first introduce data that need to be stored on the GPU and their layout. Then, the functions that operate on these data are explained along with specifics of the implementation.

5.1 Data layout

The key components of multigrid are matrix vector multiplication, restriction, prolongation, and smoothing. Matrix vector multiplication and smoothing are applied on the state variables of the modified Stokes problem (e.g., the pressure and velocity). Thus, they need to be stored on the GPU.

In our discretization, fluid velocity is stored on faces and pressure is stored on cells. As we discussed in Sect. 4.3, we store these variables in a block memory layout to improve its spatial locality. The block size should be sufficiently large (a multiple of 32) so that we can fully utilize the GPU memory bandwidth. It also should not be too big (not bigger than 1024) so that we can fit a block of data in one thread block. As a result, the block size we select is 8×8 for 2D and $4 \times 4 \times 4$ for 3D. We follow the row-major order for the storage of both data pieces in each block and blocks in a grid.

For implementation convenience, we assume the grid has a size that is a multiple of 8 in 2D (or 4 for 3D) for each dimension. This allows cell data to be stored in a block memory layout with perfect alignment. However, for face data, at least half of the boundary faces on a grid cannot fit into complete blocks. To tackle this problem, we still use blocks to store data, despite some unused entries, as shown in Fig. 10. Though the utilization ratio of these blocks is low, the number of these blocks is relatively small compared to the total number of blocks (the former is always one order of magnitude lower than the latter in terms of the grid size in each dimension). Thus, the usage of the proposed block memory layout causes only a marginal memory consumption increase which still maintains an efficient bandwidth utilization.

Besides the input, the output and part of the intermediate results are also stored on GPU, which are all discretized on grid cells, faces, and edges. We will discuss this storage in the next subsection. We want to highlight that data related to

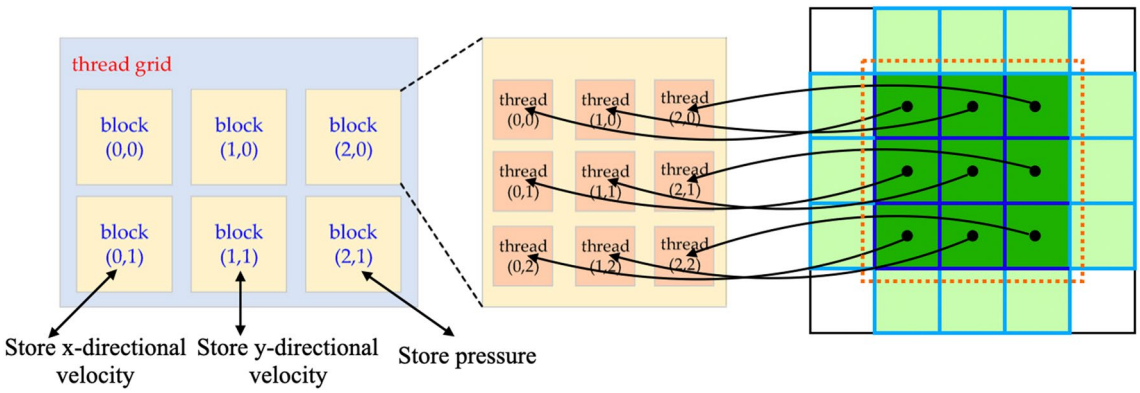

Fig. 9 Data storage of the pressure unknowns (black dots) of a 3×3 block grid on GPU. Dark blue edges and green cells are the unknowns associated with the block circled by the orange dotted box. Light blue edges and green cells are the unknowns stored from other blocks that need to be acquired in the shared memory during the computation of the central 3×3 grid. (Color figure online)

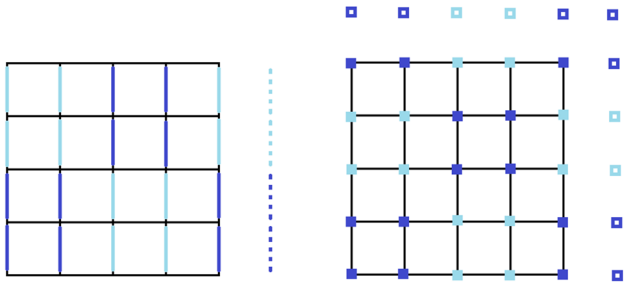

Fig. 10 An example of block memory layout. Left: faces on the x-axis. Right: vertices. The dashed line and hollow points indicate wasted entries

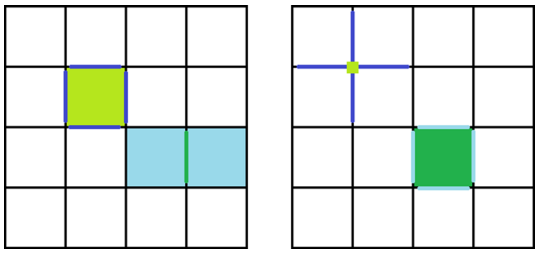

Fig. 11 An example of operators. Left: gradient and divergence. Right: curl. Blue colored features are inputs, while green colored features are outputs. (Color figure online)

edges are stored using the block memory layout, again with some entries unused, similar to data for faces.

For the multigrid we have the inputs, intermediate results, and outputs for each layer. The memory consumption of a single layer is linear to the number of cells of its (coarsened) grid, which results in an overall linear memory consumption with respect to the size of the original Stokes problem.

Restriction and prolongation are applied between different layers which do not require additional memory. The smoothing process requires computing the inverse of local matrix corresponding to each grid cell, and multiple small matrix-vector multiplication operations. These small matrices are stored and computed on the GPU. The memory consumption for these matrices is also linear, and the linear coefficient is exactly the DoFs of a box (i.e., 5 for 2D and 7 for 3D).

5.2 Discrete operators

In this section, we describe the functions needed for our GPU implementation to meet the requirements of a multigrid solver that includes matrix-vector multiplication, restriction, prolongation, and smoothing. We start from the matrix-vector multiplication operation with their three main parts of the pressure gradient, velocity divergence, and velocity Laplacian. On a staggered grid, the pressure gradient is stored on faces and a face value is the weighted sum of its neighbouring cells (with -1 or $+1$ as the coefficients). The velocity divergence is stored on cells. The value of a cell is the weighted sum of its faces.

All boundary conditions may be translated to fix values on a cell or a face. Another way to interpret this is: if the value on a cell or a face needs to be fixed, then the value itself should have no contribution to the gradient/divergence since it will be overwritten. We use this idea to enforce boundary conditions in our gradient and divergence operator; i.e., we simply set a value on a boundary is zero, and we compensate the desired fixed value on the right hand side of the Stokes equations which the boundary has contributions to.

It should be pointed out that instead of setting some coefficients to be zero, we use a filter to set values on a boundary to be zero before applying the gradient and divergence operators. This allows the implementation of these two operators to be independent from boundary conditions.

For the velocity divergence, we follow the equation $\Delta\mathbf{u} = \nabla(\nabla\cdot\mathbf{u}) - \nabla\times(\nabla\times\mathbf{u})$. We reuse the divergence and gradient operators above. For the curl operator, it is implemented in a similar way as the other two operators on a staggered grid. The output is stored on either nodes or cells for 2D, and the value is the weighted sum of its neighbouring faces (See Fig. 11 and (Crane 2018) for more details). Please note that for non-boundary cells we use the simple form of scalar Laplacian of decomposed velocity to replace the vector Laplacian of velocity vector field, as shown in (3).

We choose this implementation since it can easily support different boundary conditions for stress. Just like what we have done for the gradient and divergence operators, we apply a filter on the intermediate results. The divergence reflects the normal part, and the curl reflects the shear part. If we set the divergence on a cell to be zero, then the normal stress in that cell is ignored, which mimics an open boundary. If we set the curl on an edge to be zero, then the shear stress around that edge is ignored, which mimics a slip boundary condition. For our examples, we use open boundaries and non-slip boundaries, so we apply filters on cells only.

There are some intermediate results related to these operators that need to be stored on GPU. They are the velocity after filtering, divergence of the velocity, and curl of the velocity, which are stored on faces, cells and edges, respectively.

5.3 Other details

Matrix-vector multiplication, restriction and prolongation have high spatial locality, thus it is natural to use one thread

block to deal with one block of data so that memory bandwidth can be fully utilized.

There are two choices for handling the blocks including gathering and shattering. Gathering uses one thread block to compute the result of one block of data (e.g., the gradient from the pressure). In this case, we read all the needed input data, compute the result, then write them back. The same input data can be read multiple times, but the result will be written only once.

The shattering approach does the opposite, where it uses one block to deal with one block of input. After reading one block of input data, it computes its contribution to the result and accumulates them. Same input data can be only read once, but the result will be written multiple times.

We choose a gathering scheme. This is because writing is slower than reading on a GPU, and accumulating requires synchronization operations which are time consuming. Our implementation of gathering is straightforward. For one thread block, we conduct the following:

- read one block of data
- compute its contribution to the result
- repeat the prior two steps for all needed blocks
- write back the result

A check is performed prior to reading. If the position is outside of the grid, instead of actually reading from the memory, a zero value is assumed to mimic zero contribution. All threads in one thread block will not compute the contribution until the entire block of data is loaded into the shared memory. At one time, only one block of data is loaded instead of loading all required blocks, such that more threads can be executed at the same time.

Please note that there are no duplicates of velocity fields on boundary edges between blocks, and thus no communication is required. For an 8×8 grid block, the associated velocity field is 8×9 for u and 9×8 for v. However, only the first 8×8 of u (or v) are stored in the same block, while the last column of u, or row of v, are stored in other blocks. Considering a single grid cell, the associated velocity to store is u on the left face and v on the bottom face, and the right face is associated with the cell to the right, while the upper face is associated with the cell above. This pattern avoids duplicate storage and communication. When computing, for instance, divergence, cells at the rightmost and uppermost boundaries may have to gather velocities from different blocks, but that is a one-time read-only operation and the portion of the boundary cells are small. One last compromise is that we have to use additional blocks to store the last layer's velocity component of the entire grid, but still that additional memory cost is marginal.

The gradient, divergence, curl, prolongation, and restriction operators are all implemented as described above. For smoothing, or multiple small matrix vector multiplication operations, we use the provided functions in the cuBLAS library.

6 Numerical examples

In this section, we present a number of examples in both 2D and 3D, ranging from well-studied cases to the design of large-scale novel flow structures. For 2D cases, we start from standard examples like double pipes proposed in Borrvall and Petersson (2003) to verify the correctness of our approach. We then present branching problems with increasing resolution, number of outlets, and decreasing volume fractions to compare and show the scalability of our approach. Finally, we present a 4096×4096 asymmetric tree structure with extreme resolution and 162 fluid outlets to show the maximum data volume our implementation can support with 48 GB GPU memory.

Our 3D experiments mostly follow the philosophy of the 2D cases. The standard examples are mostly naturally extended from 2D. In addition, we use commercial software, COMSOL Multiphysics®, to optimize fluid flow structures at low-to-moderate resolutions to make quantitative comparisons of computational time and flow resistance performance of the optimized shapes. All numerical experiments reported in this paper are performed on a single desktop machine equipped with an Intel i9-9980XE CPU and a single NVIDIA Quadro RTX 8000 graphics card.

We directly enforce velocity boundary conditions at both inlets and outlets for all examples. Positions, sizes, and velocities for inlets are specified in each description subplot. In our experiments, the outlets always have the same width/area and velocity. To ensure mass conservation, outlet velocities are thus related to inlet velocities by $u_o = \frac{\sum_i^n u_i \cdot l_i}{n_o \cdot l_o}$, where n is the number of inlets and u_i and l_i are corresponding inlet velocity and width; n_o and l_o are the number of outlets and width of an outlet. Note that in 3D examples, we exchange the inlet/outlet width, l, with the inlet/outlet area, a.

During our numerical experiments, we usually find that the design volume moves very slowly toward our desired volume fraction or even stagnates at some point. To accelerate the convergence of the design density field, we also adaptively scale up the actual volume constraint and its gradient before using this information to generate the MMA subproblem, which beneficially amplifies the effect of the volume constraint. The scaling factor is tunable, but generally smaller for low resolution problems and larger for high resolution problems. Thus, we see minimal changes in the

topology and satisfaction of the volume constraint after 30 optimization iterations. We thus stop optimizing at the 40th iteration as any design changes afterward are negligible.

6.1 Rugby ball (2D)

The design domain and boundary conditions for the "rugby ball" design problem in two dimensions are shown in Fig. 12a. For comparison, we propose two choices for the maximum allowable fluid volume, $\bar{V} = [0.6, 0.7]$. Our boundary conditions differ from that of Borrvall and Petersson (2003) and Guest and Prévost (2006) at the upper and lower walls of the design domain, where no tangential velocities are specified. The results show trapezoidal shaped structures at both walls which smooth the flow transition at the leading and trailing edges. We initialize a homogeneous domain with a uniform design density that is the same as the desired volume fraction. The discretization size is 256×256, including 65.5×10^3 design variables and 197.1×10^3 state variables. The time consumed for the entire optimization workflow for each case is ~81 s with 40 optimization iterations, and the average run time for solving the Stokes equations and the adjoint equation at each iteration is about 0.7 s respectively. A time breakdown for this optimization problem is shown in Fig. 13. We find that the two solves already occupy most of the time, even though this problem is of the smallest scale among all of our experiments. For the experiments at a larger scale, our time bottleneck always lies with the two solves. Currently, we write all information to disk sequentially. The I/O time may be further reduced by only writing the density field. The presented optimized rugby ball structures are similar to those presented in the previous literature (Guest and Prévost 2006; Challis and Guest 2009). The angle at the front and back of each rugby ball matches the expected value of 90 degrees (Pironneau 1974).

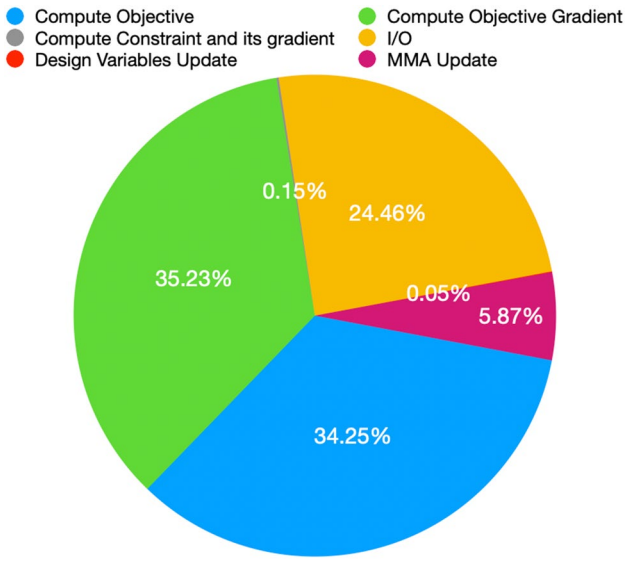

Fig. 13 Optimization time break down for Rugby ball (2D) design problem

6.2 Pipe design (2D)

The double pipe design has also been introduced in previous work (Borrvall and Petersson 2003; Guest and Prévost 2006), where domains of different aspect ratio were used to generate different channel patterns. We present here results that illustrate how different pipe patterns can also be generated by changing the gap size between the upper and lower inlets. The details of the boundary conditions, volume constraints, and optimized results are provided in Fig. 14. The results show a great degree of similarity with designs from the prior literature. We also show a new triple pipe design obtained by adding an additional inlet and outlet channel at the respective center of the left and right edges of the domain. This time we control the volume fraction while keeping the same boundary conditions, as shown in Fig. 15, and find that additional fluid channel bridge connections can be generated for larger fluid volumes. In this example,

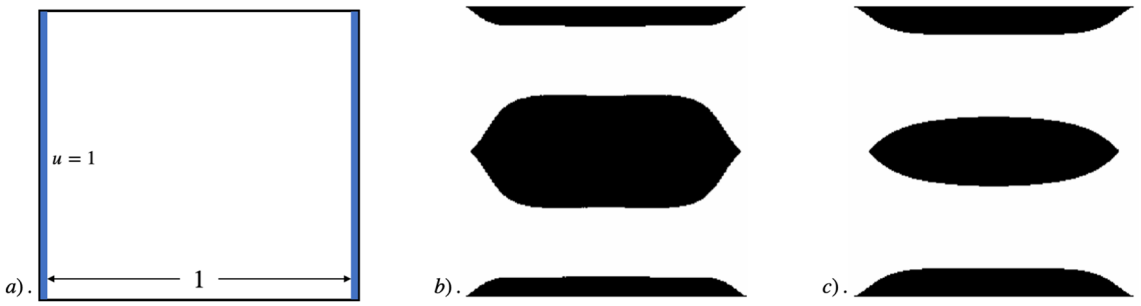

Fig. 12 Optimization results for the 2D "rugby ball" design problem with different volume fractions. **a** Design domain and boundary conditions; **b** volume fraction of fluid set as 0.6; **c** volume fraction of fluid set as 0.7

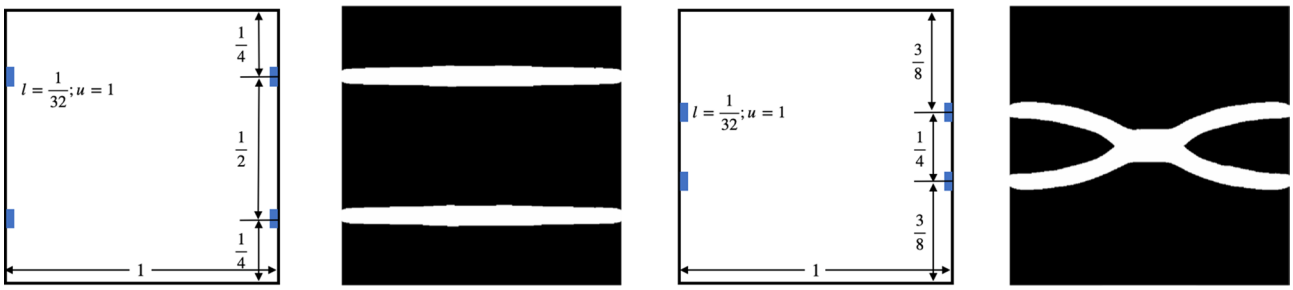

Fig. 14 Optimization results for the 2D double pipe problem with different inlet/outlet positions. The volume fraction of fluid for both designs is set to 0.12

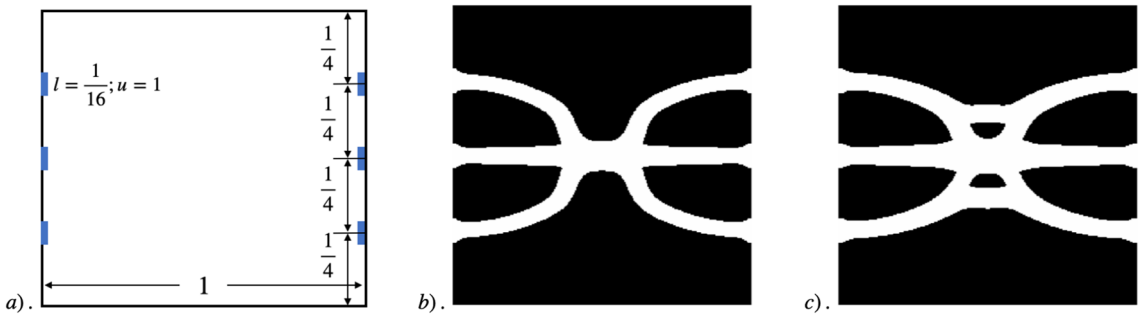

Fig. 15 Optimization results for the 2D triple pipe problem with different volume fractions. **a** Design domain and boundary conditions; **b** volume fraction of fluid set to 0.2; **c** volume fraction of fluid set to 0.25

the objective drops by 29% by increasing 5% of fluid volume. We believe this drastic decrease in energy dissipation is partially due to the newly generated branches. For these designs, we also use a discretized grid size of 256×256. The required time for the entire optimization workflow for each case is around 77 s with a total of 40 optimization iterations, and the average run time for solving the Stokes equations at each iteration is about 0.65 s. All figures show the original optimized design density field without any post processing.

6.3 Branching channels (2D)

Design of a branching fluid channel network is inspired by Dede et al. (2020), which showed the relation between the number of outlet channels and discretized elements. For a large number of outlets at high resolution, commercial optimization tools usually suffer from an unbearable time overhead due to the lack of an optimized, problem specific solver. Researchers may select alternative non-gradient-based methods for the dehomogenization and/or design synthesis of large-scale flow distribution channel networks (Dede et al. 2020). We show in Fig. 16 the scalability of our solver in 2D with up to 64 outlets at 1024×1024 resolution. The computational time for all three designs is summarized in Table 2, and the results exhibit a near linear relationship between the solver time and problem size. We also plot normalized fluid velocity color maps and normalized pressure contours in Fig. 16. For the velocity, we extract the magnitude from the vector field by interpolating the vector field at each grid cell center, and we then compute the norm. From the plots, we qualitatively observe for each design that the velocity distribution is the same through all outlets, as expected, since we explicitly specify the velocity BC. However, we do not specify any BC for pressure, so the pressure logically varies at each outlet.

Our solver can handle the design of an increased number of outlets with intricate channel topology. In Fig. 17, we show a design with 162 randomly distributed fluid outlets using a 4096×4096 grid with over 50 million DoF. An intricate, non-symmetric tree structure is automatically discovered to distribute the fluid flow. The width of the outlets may be further reduced by half thanks to the extremely high resolution. The outlet channel width was selected for illustration and visualization purposes. The run time for the Stokes solver at this resolution is about 58 s, and the total optimization time is about 117 minutes for 40 optimization iterations.

6.4 Pipe design (3D)

We next consider several design problems in three dimensions. We start with a quadruple inlet/outlet pipe design

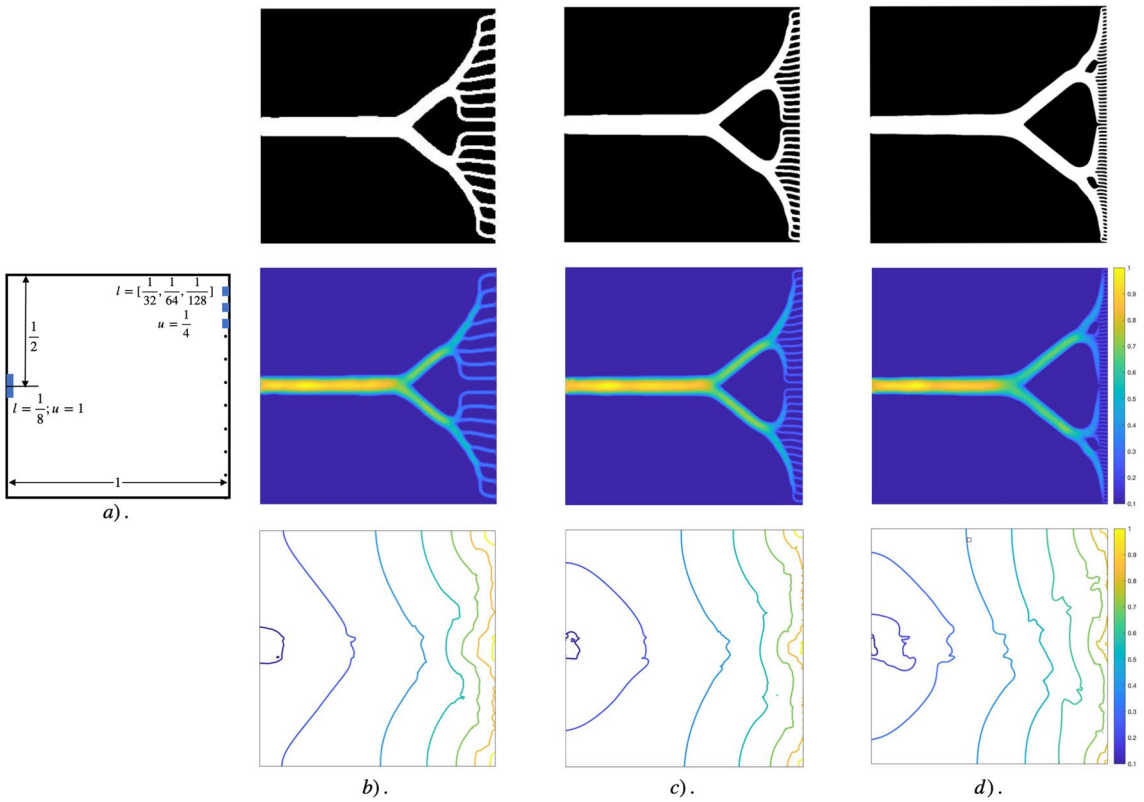

Fig. 16 Optimization results and normalized velocity/pressure distributions for the 2D branching channel problem with different resolutions, outlets, and volume fractions. **a** Design domain and boundary conditions; **b** grid size = 256 × 256, 16 outlets, fluid volume fraction = 0.17; **c** grid size = 512 × 512, 32 outlets, fluid volume fraction = 0.16; **d** grid size = 1024 × 1024, 64 outlets, volume fraction = 0.15. The first row shows each density distribution; the second row shows the normalized velocity magnitude color maps; the third row shows the normalized pressure contours

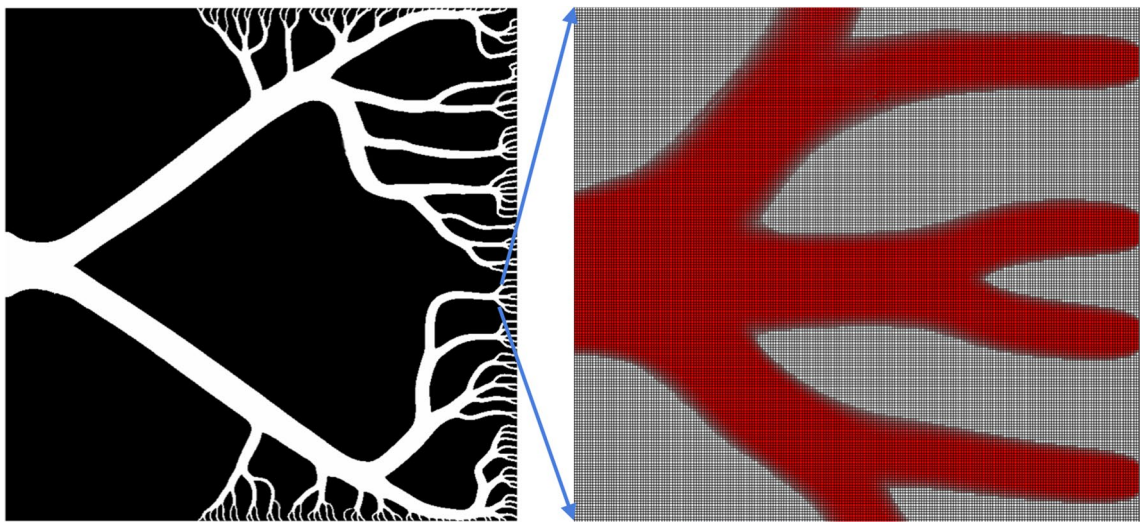

Fig. 17 Optimization result for a large scale 2D branching channel design problem (grid size = 4096 × 4096) with 162 fluid outlets

problem with volume constraints and boundary conditions, as shown in Fig. 18, as a natural extension of the 2D pipe design problems in Sect. 6.2. We present designs for two types of outlet configurations. The first design has outlets in alignment with the inlets, as shown in Fig. 18b. The second design has outlets forming a smaller square with a relative

Table 2 Computational time for the 2D branching channel designs in Fig. 16

Design (# of outlets)	*b* (16)	*c* (32)	*d* (64)
Resolution	256^2	512^2	1024^2
Stokes solver (s)	0.7	2.1	6.9
Total time (min)	1.3	3.8	13.8

rotation of 45°, as shown in Fig. 18c. Both designs show a channel merging-diverging pattern similar to the observed results in 2D. We also test the effectiveness of fluid volume control by increasing the volume fraction from 0.15 as in design (*c*) to 0.2. The optimized result shows the inlets no longer merge to a single main stream, but only merge with their neighbors forming a hollow center, as shown in Fig. 18d, e. We use a 128^3 grid to discretize the domain, which has over 2 million design variables and 6 million state variables. The computational time for the entire optimization workflow for each case is about 26.8 min for 40 optimization iterations, and the averaged run time for solving the Stokes equations at each iteration is about 17.4 s. All figures show the smoothed isosurface extracted from the optimized density field.

6.5 Branch channels (3D)

Following the idea of branching channel network design in 2D, we studied the design of a branching channel fluid flow manifold in 3D. Similar tasks have been conducted by Kreissl (2011) at a much lower resolution. Like the 2D cases, we have four different designs with increasing grid resolutions and number of outlets. The difference is that we now have a constant fluid volume fraction constraint for all cases, and the outlets are positioned randomly on the five faces of the cubic design space not including the inlet face. We also use COMSOL with Stokes flow physics to perform similar fluid flow channel design optimization with identical problem settings using its embedded direct solver. The numerical results are compared from multiple aspects including channel topology similarity, fluid flow resistance performance, and computational time. Details regarding the problem settings in COMSOL are summarized in Table 3. All settings other than the discretization and solver are fixed to be identical in our optimization approach. The optimized structures are shown in Fig. 19, with the first row being the results from COMSOL, and the last two rows representing the optimized designs using our approach and visualized from both inlet and outlet perspectives. For the optimized COMSOL results, we directly run a Stokes flow simulation after extracting the fluid domain within the software and measure the fluid

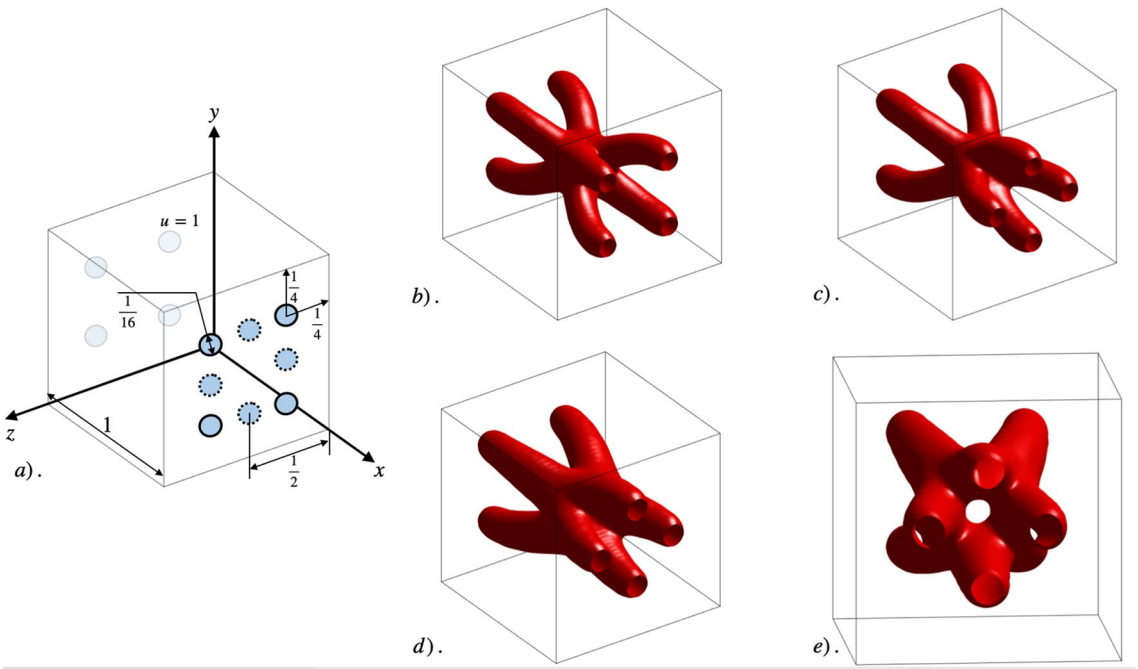

Fig. 18 Optimization results for the 3D quadruple pipe problem with different volume fractions and outlets. **a** Design domain and boundary conditions; **b** outlets are aligned with inlets with fluid volume fraction equal to 0.15; **c** outlets are aligned with the centers of the edges of the inlets with fluid volume fraction equal to 0.15; **d** outlets are aligned with centers of the edges of the inlets with fluid volume fraction equal to 0.2; **e** a side view of the design in (*d*) to show the hollow center

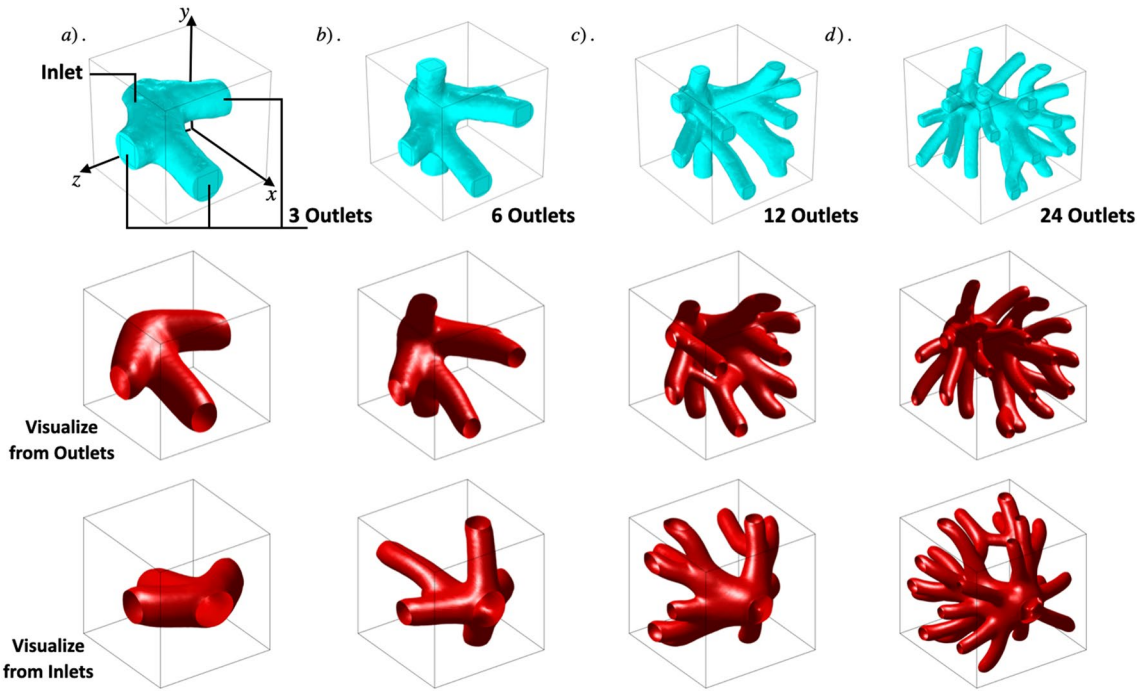

Fig. 19 Optimization results for a 3D branching channel problem with different resolutions and outlets. First row: benchmark results from COMSOL with identical settings. **a** grid size = $32 \times 32 \times 32$, 3 outlets; **b** grid size = $48 \times 48 \times 48$, 6 outlets; **c** grid size = $64 \times 64 \times 64$, 12 outlets; **d** grid size = $96 \times 96 \times 96$, 24 outlets

flow resistance across the channel structure. For the results from our implementation, we first extract a smoothed isosurface from the optimized density field for each design, and export to a binary STL file, which is then read by COMSOL for the same simulation. Observe in Fig. 19 that our results closely resemble those obtained using the commercial solver, with some intricate inner branching channel structures that are slightly different from the COMSOL results. All results together with the computational time are summarized in Table 4. From the data, we find that the flow performance of our designs matches well with the designs obtained using the commercial solver, especially at higher resolutions. Our solver speed far surpasses the commercial solver's built-in performance, as expected. It is worth noting that our solver's performance may decay slightly when the resolution is not an integer power of two, like 48^3, 96^3, or 160^3, or the resolution in each dimension is different, like $128 \times 128 \times 192$. This is because such grids can not be coarsened all the way down, as our coarsening strategy reduces the dimension by half at each multigrid

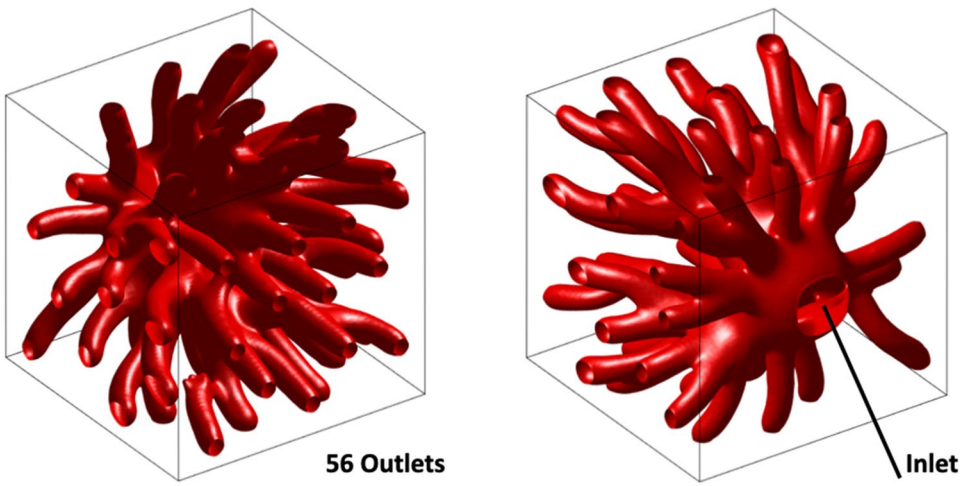

Fig. 20 Optimization result for a large-scale 3D branching channel problem (grid size = $192 \times 192 \times 192$) with 56 outlets

Table 3 Problem setup in COMSOL for optimization of a 3D branching channel network, Sect. 6.5

Physics	Stokes flow
Domain	$0.1 \times 0.1 \times 0.1$ (m^3)
Fluid density	1.204 (kg/m^3)
Dynamic viscosity	1.813e−5 (Pa s)
Inlet pressure	1e3 (Pa)
Outlet velocity	1 (m/s)
Objective	Flow resistance
Fluid volume fraction	0.15
Solver type	Direct
Optimization iterations	40
Mesh type	Tetrahedral
CPU	Intel i9-9980XE

Table 4 Computational time and flow performance comparisons for the 3D branching channel designs from Fig. 19

Design	*a*	*b*	*c*	*d*
(# of outlets)	(3)	(6)	(12)	(24)
Elements[a]	32.9e3	94.7e3	268.2e3	767.6e3
Volume[a](m^3)	1.51e−4	1.52e−4	1.50e−4	1.50e−4
Total time[a] (min)	5.7	15.4	62.9	445.4
Flow resistance[a]	2.79	2.39	4.45	5.76
Elements[b]	32^3	48^3	64^3	96^3
Volume[b] (m^3)	1.46e−4	1.40e−4	1.54e−4	1.57e−4
Total time[b] (min)	0.67	2.62	3.08	25.34
Flow resistance[b]	3.21	3.23	5.01	5.88

[a]Data from COMSOL

[b]Data from our approach

level. The direct solver we used at the coarsest level is sensitive to the change of problem size. We also conduct a high-resolution branching design in 3D with 56 outlets, as shown in Fig. 20. Here, we use a 192^3 grid with over 28 million DoF. Even though the resolution is not an integer power of two, our solver is still capable of solving the Stokes equations in 54 s and finishing the optimization run in 98 minutes over a total of 40 iterations.

7 Conclusions

In this paper, an efficient GPU-based computational approach was proposed for the flow-based topology optimization. A matrix-free GMRES solver preconditioned by geometric multigrid was developed on top of a staggered grid discretization and MAC scheme, which circumvented the bottleneck of solving the state equations. Our implementation was designed and optimized for desktop computer architectures enabled by a GPU. The problem size our implementation can handle was restricted only by the GPU memory. Through multiple 2D and 3D numerical examples, we demonstrated the solver performance and practical results, which for select cases were compared with solutions obtained using existing commercial software and open-source libraries. Our approach can handle similar tasks dozens of times faster than current tools, and seamlessly tackles larger size design cases.

An immediate next step is to add nonlinearity to the system to handle flow at a higher Reynolds number. In the future, we plan to further improve the memory allocation and explore multi-GPU parallelization. Another promising direction is to replace the indefinite Stokes equations with a couple of positive definite systems, similar to Aage et al. (2008). By doing so, we can boost the system performance using fast solvers (e.g., conjugate gradient) which require much less memory overhead. We are also interested in introducing other physics (e.g., thermal and structural consideration) to broaden the design space and application scope.

Appendix 1: Sensitivity

In this section, we discuss the derivative of the objective, J, with respect to the fluid design density variable, ρ. The adjoint method is used to compute the sensitivity. First, we simplify the notation by defining $\mathbf{X} = [\mathbf{u}, P]^T$ as a stacked vector comprising all state variables. The linear system in (4) can be written as $\mathcal{A}\mathbf{X} = \mathbf{b}$. Since the system matrix, $\mathcal{A}$, depends on ρ, so does the solution vector, $\mathbf{X}$. The objective is a function of the state and design variables. Let us define the Lagrangian,

$$\mathcal{L}(\mathbf{X}, \rho, \lambda) = J(\mathbf{X}, \rho) + \lambda^T(\mathcal{A}\mathbf{X} - \mathbf{b}), \tag{9}$$

where λ is the vector of Lagrange multipliers. As $(\mathcal{A}\mathbf{X} - \mathbf{b})$ is zero everywhere by construction, we may choose λ freely such that $J = \mathcal{L}$, and derive the gradients as follows,

$$\begin{aligned} \frac{dJ}{d\rho} &= \frac{d\mathcal{L}}{d\rho} \\ &= \frac{\partial J}{\partial \rho} + \frac{\partial J}{\partial \mathbf{X}}\frac{\partial \mathbf{X}}{\partial \rho} + \lambda^T\left(\frac{\partial \mathcal{A}}{\partial \rho}\mathbf{X} + \mathcal{A}\frac{\partial \mathbf{X}}{\partial \rho}\right) \\ &= \frac{\partial J}{\partial \rho} + \lambda^T\frac{\partial \mathcal{A}}{\partial \rho}\mathbf{X} + \left(\frac{\partial J}{\partial \mathbf{X}} + \lambda^T\mathcal{A}\right)\frac{\partial \mathbf{X}}{\partial \rho}. \end{aligned} \tag{10}$$

If we choose λ so that $\frac{\partial J}{\partial \mathbf{X}} + \lambda^T\mathcal{A} = 0$, the last term becomes zero, and we can avoid calculating $\frac{\partial \mathbf{X}}{\partial \rho}$. This condition is the adjoint equation:

$$\mathcal{A}^T\lambda = -\left(\frac{\partial J}{\partial \mathbf{X}}\right)^T. \tag{11}$$

Here, $\mathcal{A}^T = \mathcal{A}$ based on symmetry, and the state variable, $\mathbf{X}$, is pre-computed at each iteration. Equation (11) is solved the same way as the Stokes equations using our multigrid solver, after which λ is substituted to (10) to get the sensitivity. The objective, J, and matrix, $\mathcal{A}$, are linear functions of $\alpha(\rho)$, and their partial derivatives with respect to ρ can be obtained using the chain rule by differentiating (2). Optionally, the adjoint can also be obtained by using automatic differentiation (Griewank and Walther 2008) for the state equations. Overall, this adjoint formulation is an efficient way to evaluate the sensitivity, especially when the number of design variables is large.

Once the sensitivity is computed, it is used to update the fluid design density variable, ρ, of each grid cell. The updated density value is restricted to a range between 0 and 1.

Acknowledgements This project is supported in part by Toyota Research Institute of North America (TRINA), NSF MRI-1919647, and NSF HCC-2106733.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Replication of results The source code is available on Github with open access at https://github.com/jyl-pages/Stokes_TO

References

Aage N, Poulsen TH, Gersborg-Hansen A, Sigmund O (2008) Topology optimization of large scale stokes flow problems. Struct Multidisc Optim 35(2):175–180

Aage N, Andreassen E, Lazarov BS (2015) Topology optimization using petsc: an easy-to-use, fully parallel, open source topology optimization framework. Struct Multidisc Optim 51(3):565–572

Aage N, Andreassen E, Lazarov BS, Sigmund O (2017) Giga-voxel computational morphogenesis for structural design. Nature 550(7674):84–86

Alexandersen J, Andreasen CS (2020) A review of topology optimisation for fluid-based problems. Fluids 5(1):29

Alexandersen J, Sigmund O, Aage N (2016) Large scale three-dimensional topology optimisation of heat sinks cooled by natural convection. Int J Heat Mass Transf 100:876–891

Allaire G (2015) A review of adjoint methods for sensitivity analysis, uncertainty quantification and optimization in numerical codes. Ingénieurs de l'Automobile 836:33–36

Andreasen CS, Sigmund O (2013) Topology optimization of fluid-structure-interaction problems in poroelasticity. Comput Methods Appl Mech Eng 258:55–62

Andreasen CS, Gersborg AR, Sigmund O (2009) Topology optimization of microfluidic mixers. Int J Numer Methods Fluids 61(5):498–513

Benzi M, Golub GH, Liesen J (2005) Numerical solution of saddle point problems. Acta Numerica 14:1–137

Borrvall T, Petersson J (2003) Topology optimization of fluids in stokes flow. Int J Numer Methods Fluids 41(1):77–107

Challis VJ, Guest JK (2009) Level set topology optimization of fluids in stokes flow. Int J Numer Methods Eng 79(10):1284–1308

Challis VJ, Roberts AP, Grotowski JF (2014) High resolution topology optimization using graphics processing units (GPUS). Struct Multidisc Optim 49(2):315–325

Chen L (2016) Finite difference method for stokes equations: Mac scheme

Crane K (2018) Discrete differential geometry: an applied introduction. Notices of the AMS, Communication, pp 1153–1159

Deaton JD, Grandhi RV (2014) A survey of structural and multidisciplinary continuum topology optimization: post 2000. Struct Multidisc Optim 49(1):1–38

Dede EM (2009) Multiphysics topology optimization of heat transfer and fluid flow systems. In: Proceedings of the COMSOL users conference

Dede EM (2012) Optimization and design of a multipass branching microchannel heat sink for electronics cooling. J Electron Packag 134(4)

Dede EM, Zhou Y, Nomura T (2020) Inverse design of microchannel fluid flow networks using turing pattern dehomogenization. Struct Multidisc Optim 62:2203–2210

Demidov D (2019) Amgcl: an efficient, flexible, and extensible algebraic multigrid implementation. Lobachevskii J Math 40(5):535–546

Demidov D, Mu L, Wang B (2021) Accelerating linear solvers for stokes problems with c++ metaprogramming. J Comput Sci 49:101285

Dilgen CB, Dilgen SB, Fuhrman DR, Sigmund O, Lazarov BS (2018) Topology optimization of turbulent flows. Comput Methods Appl Mech Eng 331:363–393

Du T, Wu K, Spielberg A, Matusik W, Zhu B, Sifakis E (2020) Functional optimization of fluidic devices with differentiable stokes flow. ACM Trans Graphics 39(6):1–15

Evgrafov A (2006) Topology optimization of slightly compressible fluids. J Appl Math Mech 86(1):46–62

Gersborg-Hansen A, Sigmund O, Haber RB (2005) Topology optimization of channel flow problems. Struct Multidisc Optim 30(3):181–192

Griewank A, Walther A (2008) Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM, Philadelphia

Guest JK, Prévost JH (2006) Topology optimization of creeping fluid flows using a Darcy-Stokes finite element. Int J Numer Methods Eng 66(3):461–484

Herrero-Pérez D, Castejón PJM (2021) Multi-GPU acceleration of large-scale density-based topology optimization. Adv Eng Softw 157:103006

Jiang T (1970) A first order method of moving asymptotes for structural optimization. WIT Trans Built Environ 14

John V, Tobiska L (2000) Numerical performance of smoothers in coupled multigrid methods for the parallel solution of the incompressible navier-stokes equations. Int J Numer Methods Fluids 33(4):453–473

Kondoh T, Matsumori T, Kawamoto A (2012) Drag minimization and lift maximization in laminar flows via topology optimization employing simple objective function expressions based on body force integration. Struct Multidisc Optim 45(5):693–701

Kontoleontos E, Papoutsis-Kiachagias E, Zymaris A, Papadimitriou D, Giannakoglou K (2013) Adjoint-based constrained topology optimization for viscous flows, including heat transfer. Eng Optim 45(8):941–961

Kreissl S (2011) Topology optimization of flow problems modeled by the incompressible Navier-Stokes equations. PhD thesis, University of Colorado at Boulder

Liu H, Mitchell N, Aanjaneya M, Sifakis E (2016) A scalable Schur-complement fluids solver for heterogeneous compute platforms. ACM Trans Graphics 35(6):1–12

Liu H, Hu Y, Zhu B, Matusik W, Sifakis E (2018) Narrow-band topology optimization on a sparsely populated grid. ACM Trans Graphics 37(6):1–14

Martínez-Frutos J, Herrero-Pérez D (2016) Large-scale robust topology optimization using multi-GPU systems. Comput Methods Appl Mech Eng 311:393–414

Maute K, Frangopol DM (2003) Reliability-based design of mems mechanisms by topology optimization. Comput Struct 81(8–11):813–824

Olesen LH, Okkels F, Bruus H (2006) A high-level programming-language implementation of topology optimization applied to steady-state Navier-Stokes flow. Int J Numer Methods Eng 65(7):975–1001

Papoutsis-Kiachagias EM, Giannakoglou KC (2016) Continuous adjoint methods for turbulent flows, applied to shape and topology optimization: industrial applications. Arch Comput Methods Eng 23(2):255–299

Pironneau O (1974) On optimum design in fluid mechanics. J Fluid Mech 64(1):97–110

Rozvany GI (2009) A critical review of established methods of structural topology optimization. Struct Multidisc Optim 37(3):217–237

Sá LF, Okubo CM, Silva EC (2021) Topology optimization of subsonic compressible flows. Struct Multidisc Optim 64:1–22

Schmidt S, Schulz V (2011) A 2589 line topology optimization code written for the graphics card. Comput Vis Sci 14(6):249–256

Sigmund O, Maute K (2013) Topology optimization approaches. Struct Multidisc Optim 48(6):1031–1055

Vanka SP (1986) Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. J Comput Phys 65(1):138–158

Vicente W, Picelli R, Pavanello R, Xie Y (2015) Topology optimization of frequency responses of fluid-structure interaction systems. Finite Elem Anal Des 98:1–13

Wadbro E, Berggren M (2009) Megapixel topology optimization on a graphics processing unit. SIAM Rev 51(4):707–721

Wittum G (1989) Multi-grid methods for Stokes and Navier-Stokes equations. Numerische Mathematik 54(5):543–563

Wu J, Dick C, Westermann R (2015) A system for high-resolution topology optimization. IEEE Trans Vis Comput Graphics 22(3):1195–1208

Yadav P, Suresh K (2014) Large scale finite element analysis via assembly-free deflated conjugate gradient. J Comput Inf Sci Eng 14(4)

Yaji K, Ogino M, Chen C, Fujita K (2018) Large-scale topology optimization incorporating local-in-time adjoint-based method for unsteady thermal-fluid problem. Struct Multidisc Optim 58(2):817–822

Yoon GH (2010) Topology optimization for stationary fluid-structure interaction problems using a new monolithic formulation. Int J Numer Methods Eng 82(5):591–616

Zhou S, Li Q (2008) A variational level set method for the topology optimization of steady-state Navier-Stokes flow. J Comput Phys 227(24):10178–10195