An efficient hybrid multigrid solver for high-order discontinuous Galerkin methods

Master's thesis

Author:
Peter Münch

Matriculation number:
03614777

Supervisors:
Dr. Martin Kronbichler and Niklas Fehn, M.Sc.

Date of issue: 01.05.2018
Submission date: 31.10.2018

Statement of Authorship

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt habe, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

Garching b. München, October 31, 2018 _____

Contents

List of figures

List of tables

Nomenclature

Geometry, domain, and boundaries

Γ	Boundary of domain	
Γ_D	Boundary of domain with Dirichlet boundary conditions	
Γ_N	Boundary of domain with Neumann boundary conditions	
Ω	Domain	
$\partial\Omega$	Boundary of domain	
$\boldsymbol{n}$	Outward pointing unit normal vector	
$\boldsymbol{x}$	Cartesian coordinates	
A	Area	[m^2]
d	Number of spatial dimensions	
L	Length scale	[m]
V	Volume	[m^3]

Fluid mechanical quantities

κ, ν	Dynamic diffusivity	[m^2/s]
$\boldsymbol{a}$	Velocity	[m/s]
$\boldsymbol{f}$	Body forces per unit mass	[N/kg]
$\boldsymbol{v}$	Velocity	[m/s]
$\boldsymbol{F}(\boldsymbol{u})$	Flux tensor	[m^2/s^2]
$\boldsymbol{F}_c(\boldsymbol{u})$	Convective flux tensor	[m^2/s^2]
$\boldsymbol{F}_v(\boldsymbol{u})$	Viscous flux tensor	[m^2/s^2]
f	Source term per unit mass	[1/s]
g_D, g_N	Values and normal gradient at the Dirichlet and Neumann boundary	[-]
p	Kinematic pressure	[m^2/s^2]
u	Generic scalar quantity	[-]

Differential operators

$\frac{\partial}{\partial t}$	Partial derivative with respect to time
∇	Gradient operator
$\nabla\cdot$	Divergence operator
$\nabla^2 = \Delta$	Laplace operator

Average and jump operators

$\{\{\cdot\}\}$	Average operator
$[\![\cdot]\!]$	Jump operator

Spatial discretization

$\hat{\nabla}$	Gradient on the unit cell	
$(\cdot)^*$	Numerical flux	
ϕ	Shape functions/unit cell basis functions	
τ	Penalty parameter of SIP method	
Ω_e	Region of element e	
$\partial\Omega_e$	Boundary of element e	
Ω_h	Computational domain	
Γ_e	Boundary of element e	
$\mathcal{J}$	Variational problem	
$\mathcal{T}_h$	Tessellation	
$\mathcal{Q}_k$	Space spanned by the tensor product of degree-k-polynomials	
V_h, $\mathcal{V}_h^{\boldsymbol{u}}$, $\mathcal{V}_h^p$, S	Spaces of test and trial functions	
$\boldsymbol{\sigma}_h^*$	Gradient flux term	
$\boldsymbol{J} = \partial\boldsymbol{x}/\partial\hat{\boldsymbol{x}}$	Jacobian matrix	
$\hat{\boldsymbol{x}}$	Position on the unit cell	
$\hat{\boldsymbol{x}}_q$	Position of quadrature point on the unit cell	
d_h^e	Discretized velocity divergence term	
H^1	Space of square integrable functions with square integrable derivatives	
k, k_u, k_p	Polynomial order of shape function	
l	Refinement level, multigrid level	
L_2	Space of square integrable functions	
l_h^e	Discretized (negative) Laplace operator	
N_{el}	Number of elements	
$p = k + 1$	Number of point in 1D	
u_h^*	Value flux term	
w_q	Quadrature weight	

Temporal discretization

Δt	Time step size	[s]
t	Time	[s]
t^n	Discrete instant of time (time step number n)	[s]
α_i, γ_0	Constants related to BDF time integration	
β_i	Constant related to extrapolation scheme	
J	Order of time integration scheme	

Dimensionless numbers

Cr	Courant–Friedrichs–Lewy number
Pe	Péclet number
Pe_e	Element Péclet number

Matrices, operators, and vectors

Π	Permutation matrix
$\mathcal{A}$, $\boldsymbol{A}$	System matrix
$\mathcal{A}^b$, $\boldsymbol{A}^b$	Boundary stiffness matrix
$\mathcal{A}^e$, $\boldsymbol{A}^e$	Element stiffness matrix
$\mathcal{A}^f$, $\boldsymbol{A}^f$	Face stiffness matrices
$\mathcal{P}$, $\boldsymbol{P}$	Prolongator
$\mathcal{R}$, $\boldsymbol{R}$	Restrictor
$\mathcal{S}$, $\boldsymbol{S}$	Smoother
α, β, δ, ρ, ω λ	Eigenvalue of system matrix $\boldsymbol{A}$
$\boldsymbol{b}$	Load vector/right-hand side vector
$\boldsymbol{b}_e$	Element load vector
$\boldsymbol{d}$	Defect vector
$\boldsymbol{p}$, $\boldsymbol{v}$, $\boldsymbol{t}$	Arbitrary vectors
$\boldsymbol{r}$	Residuum vector
$\boldsymbol{x}$	Solution vector
$\boldsymbol{I}_i$	Identity matrix of size $i \times i$
M	Mass matrix
M^{-1}	Preconditioner
N_{DoFs}	Number of unknowns

Computer science

$\mathcal{M}$	Normalized memory consumption	[-]
BW	Bandwidth	[GB/s]
$E_n = S_n/n$	Parallel efficiency (n = number of nodes)	[-]
$\mathrm{P}_{\max}$	Peak performance of processor	[FLOP/s]
$S_n = T_1/T_n$	Speedup (n = number of nodes)	[-]

Solver characteristics

n	Number of cycles	
n_{10}	Number of cycles to reduce order by ten orders	
ρ	Convergence rate	[-]
r	Throughput	[DoFs/s]

Strong-scaling limit model

α	Cost of one matrix-free multigrid level	[s]
β	Cost of one embedded cycle	[s]
N_{iter}	Number of cycles	
$N_{iter,i}$, $\overline{N}_{iter,i}$	(Average) number of inner cycles	
$N_{iter,o}$	Number of outer cycles	
r	Throughput	[DoFs/s]

Acronyms

AMG	Algebraic multigrid method
BDF	Backward differentiation formula
CFL	Courant–Friedrichs–Lewy number
CG	Continuous Galerkin method
CPU	Central processing unit
DG	Discontinuous Galerkin method
DoF	Degrees of freedom
FDA	U.S. Food and Drug Administration
FEM	Finite element method
FGMRES	Flexible generalized minimal residual method
FVM	Finite volume method
GMRES	Generalized minimal residual method
h-MG	h-multigrid method
hp-MG	hp-multigrid method
ILP	Instruction-level parallelism
ILU	Incomplete LU factorization
INDEXA	A high-order discontinuous Galerkin solver for turbulent incompressible flow towards the EXA scale
KLU	Clark Kent LU factorization
LRZ	Leibniz Supercomputing Centre
MG	Multigrid
MPI	Message Passing Interface
NNZ	Number of nonzero matrix elements
PCG	Preconditioned conjugate gradient method
PDE	Partial differential equation
p-MG	p-multigrid method
RFO	Read for ownership
SIMD	Single instruction stream, multiple data streams
SIP	Symmetric interior penalty Galerkin method
SpMV	Sparse matrix-vector multiplication
SUPG	Streamline-upwind Petrov–Galerkin method

AN EFFICIENT HYBRID MULTIGRID SOLVER FOR HIGH-ORDER DISCONTINUOUS GALERKIN METHODS

MASTER'S THESIS

Peter Münch
Institute for Computational Mechanics
Technical University Munich
Boltzmannstr. 15, 85748 Garching b. München, Germany
petermuench@mytum.de

October 31, 2018

ABSTRACT

A hybrid multigrid solver for high-order discontinuous Galerkin methods combining coarsening in the mesh size h and the polynomial degree p has been developed and is presented in this Master's thesis. It can efficiently solve second-order partial differential equations for complex geometries on modern CPU hardware as well as on massively parallel and distributed systems. The given operator is rediscretized on every level, and the size of the coarse-grid problem is reduced as much as possible. Almost all multigrid components are evaluated in a highly efficient matrix-free way, based on sum factorization, and an algebraic multigrid solver is applied for solving the coarse-grid problem. A detailed investigation of a variety of multigrid design choices, including alternative p-coarsening strategies and the auxiliary space idea, is presented. The implementation efficiency of all multigrid components on their own is demonstrated using experimental measurements, conducted for standard finite element methods and discontinuous Galerkin methods for 2D and 3D, as well as for Cartesian and curved meshes. The results are compared to theoretical expectations. The overall efficiency of the developed hybrid multigrid solver is demonstrated by its application to the Poisson problem, to the convection–diffusion equation, and to the unsteady incompressible Navier–Stokes equations. Strong- and weak-scaling results, showing excellent parallel efficiency, as well as a novel strong-scaling model of the developed hybrid multigrid solver are presented.

***K*eywords** AMG · convection–diffusion equation · discontinuous Galerkin methods · h-multigrid · hp-multigrid · high-order methods · hybrid multigrid methods · incompressible Navier–Stokes equations · matrix-free methods · node-level performance analysis · p-multigrid · Poisson problem · strong scaling · weak scaling

1 Introduction

The discontinuous Galerkin (DG) method has attained increasing popularity over the last decade as a method for solving partial differential equations (PDE). Due to its stability for convection-dominated problems, high-order capacity even on unstructured mesh, geometrical flexibility on curved boundaries, as well as efficiency on massively parallel high-performance computers, DG is a feasible alternative to finite volume methods (FVM) and to finite element methods (FEM)[1].

In many typical applications of DG (*e.g.* Navier–Stokes equations with projection method [53], Vlasov–Poisson equations describing the evolution of a plasma in its self-consisting electromagnetic field [38]), second-order PDEs have to be solved as a subproblem, *e.g.* of the Poisson form:

$$-\Delta u = f, \tag{1}$$

or more general, of the convection–diffusive form:

$$\nabla \cdot (\boldsymbol{a}\, u) - \nabla \cdot (\kappa \nabla u) = f. \tag{2}$$

Multigrid methods are among the most competitive solvers for such problems [33] where an equation system of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ has to be solved ($\boldsymbol{A}$ is the system matrix, $\boldsymbol{b}$ is the right-hand side vector containing source term and boundary conditions, and $\boldsymbol{x}$ is the solution vector). A generic way to express the multigrid algorithm in the context of finite element methods [14, p. 229] for solving the equivalent variational problem $\mathcal{J}(x) \to \min\limits_{x \in S_f}$ in order to find a solution x from the 'fine' space S_f is:

1. pre-smoothing: remove the high-frequency error components in the initial guess with a smoother $\mathcal{S}$:

$$x \leftarrow \mathcal{S}(x), \tag{3a}$$

2. coarse-grid correction: solve the variational problem on a coarse grid:

$$\mathcal{J}\,(x+v) \to \min_{v \in S_c} \qquad \leftrightarrow \qquad \boldsymbol{A}_c \boldsymbol{v} = \boldsymbol{b}_c, \tag{3b}$$

   and add the coarse-grid correction v which has been interpolated with a prolongator $\mathcal{P}$ onto the fine grid:

$$x \leftarrow x + \mathcal{P}(v), \tag{3c}$$

3. post-smoothing: remove the high-frequency error components introduced during the interpolation:

$$x \leftarrow \mathcal{S}(x). \tag{3d}$$

Details have been hidden intentionally behind operators, which can be chosen arbitrarily by the user. Questions such as 'Which smoother $\mathcal{S}$ should be chosen?' or 'How should the coarse problem be solved?' might arise in regard to that. Whether the algorithm is called recursively to solve the coarse problem or not, makes the algorithm a multi-level (multigrid) or a two-level (fine and coarse-grid) algorithm. Probably the most important questions in designing a multigrid software are concerning how the coarse grid is created and what the variational problem looks like on the coarse grid. Or in other words, given a fine grid and a matrix $\boldsymbol{A}$, how is the coarse matrix $\boldsymbol{A}_c$ constructed and how is the prolongation matrix $\boldsymbol{P}$ constructed? Various multigrid approaches tackle the last two problems differently.

Which multigrid approach to choose depends closely on the mesh generation. Let us assume that a coarse (possibly unstructured) grid is given and the fine mesh is generated by globally refining each cell on the coarse grid – also known as macro cell – recursively. In that case, it is a natural choice to use the resulting mesh hierarchy also for multigrid. Since in this case the polynomial shape functions of the elements normally have the same order k on all levels and only differ in their mesh-size h, this method is referred to as h-multigrid (h-MG).

Alternatively, in the context of high-order finite elements it is possible to create additional levels by sequentially reducing the polynomial order k of the shape functions of the elements, while keeping the same mesh. This method is known as p-multigrid (p-MG). The combination of h-multigrid and p-multigrid is referred to as hp-multigrid (hp-MG).

In all three cases (h-MG, p-MG, hp-MG), the definition of the notion of a mesh is indispensable in explicitly creating the grid levels. The transfer between the levels is trivial because it is either an element-local operation (p-MG) or an operation between an element and its direct children (h-MG). For a very fine, unstructured mesh with low-order elements, it is not as trivial to explicitly construct enough multigrid levels. In that case, it is common to use algebraic

[1] The standard FEM is referred to as continuous Galerkin (CG) methods in this Master's thesis.

multigrid (AMG; see review by Stüben [93])[2], which sets up the system matrix $\boldsymbol{A}_f$ and exploits its properties to algebraically create coarse levels and associated transfer matrices $\boldsymbol{P}$. The coarse matrix $\boldsymbol{A}_c$ is obtained using the Galerkin multiplication:

$$\boldsymbol{A}_c = \boldsymbol{R}\boldsymbol{A}_f\boldsymbol{P}, \tag{4}$$

where $\boldsymbol{R}$ is the restrictor, which restricts the residuum $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ onto the coarse grid via $\boldsymbol{b}_c = \boldsymbol{R}\boldsymbol{r}$. For symmetric matrices, the restrictor is generally selected as $\boldsymbol{R} = \boldsymbol{P}^T$. In contrast to AMG, both h-MG and p-MG offer the possibility of rediscretizing the PDE for every level with a coarser mesh or with lower-order elements. This approach is highly efficient in terms of performance and computational resources in combination with matrix-free methods. On the other hand, matrix-free methods are restricted with regard to the choice of smoothers and often cannot use algebraic smoothers developed specifically for anisotropic problems[3].

This concludes the introduction of terminology. A brief review of the literature on multigrid methods is given below. Although the focus is on the p-multigrid for high-order discontinuous Galerkin methods, other well-known concepts that bear the potential to improve the overall efficiency are also outlined. Publications referenced later in this work are discussed in more detail.

For spectral elements, p-multigrid ('spectral element multigrid') was first proposed 1987 by Rønquist and Patera [86]. A theoretical justification was provided by Maday and Munoz [66] the following year. In the late 1990, Hu, Guo, and Katz [35, 36, 48, 49] investigated different aspects of p-multigrid ('multi-p methods') such as cycle forms, usage as a preconditioner of conjugate gradient methods, application of condensed finite elements, smoothers, and parallelization.

Helenbrook, Atkins, and co-authors explored p-multigrid in many papers [3, 39–42, 68, 69] over the past two decades. In [39], they demonstrated its mesh and order-independent properties for the solution of the Poisson equation discretized with continuous finite elements. In [3, 40, 69], p-multigrid was applied to DG discretization of the Poisson equation and to the diffusion equation for different DG formulations (Bassi and Rebay [8], Bassi et al. [11], Brezzi et al. [15], interior penalty by Douglas and Dupont [22], local DG by Cockburn and Shu [18]). In their investigations, the authors made the following observations:

- Constructing the coarse matrix algebraically is more stable than a rediscretization for each level. Except for Brezzi et al. [15], all DG formulations resulted in unstable iterations in the case of rediscretization due to inconsistency between matrices. A remedy for the instability of the interior penalty scheme is to adjust the penalty parameter with k.
- Coarsening to $k_{i-1} = k_i - 1$ instead of $k_{i-1} = k_i/2$ results in only a small improvement of the convergence rate. It is, however, computationally more expensive. The p-multigrid coarsening strategy $k_{i-1} = (k_i+1)/2-1$ is also functional.
- The coarse problem does not need to be solved precisely as long as the result is represented well on the coarse space.
- Long-wavelength modes are not represented well for piecewise constant space (DG with $k = 0$). Thus, p-multigrid with p-coarsening to $k_c = 0$ has a poor convergence rate. As a remedy, embedded cycles between $k = 0$ and $k = 1$ have been proposed [3], which restores the convergence rate observed for $k_c = 1$.
- Alternatively, p-multigrid with transfer from linear DG space to piecewise linear CG space shows a convergence rate that is independent of the grid size and is weakly sensitive to the polynomial order. The coarse matrix for CG is constructed from the DG coarse matrix in accordance with the Galerkin multiplication:

  $$\boldsymbol{A}_{CG} = \boldsymbol{P}_{CG}^T\boldsymbol{A}_{DG}\boldsymbol{P}_{CG}, \tag{5}$$

  using prolongation matrices $\boldsymbol{P}_{CG}$ described in [41]. Since an approximate solution is computed in an 'auxiliary' continuous space for the original discontinuous problem, this approach is also referred to as auxiliary space idea or two-level preconditioning [20, 21, 97].

In [69], the authors investigated p-multigrid for convection–diffusion equations and took an in-depth look at the transfer from DG to CG in this context. They demonstrated that a simple transfer from DG to CG is not applicable in this case.

[2] An alternative approach – as used by COMSOL [60] as well as by Krause and Zulian [54] – is to construct nonconforming meshes on multigrid levels. This approach requires general interpolation or projection matrices and cannot exploit the child-parent relationship between cells on the coarse level and the fine level, preventing the use of highly efficient matrix-free interpolation procedures (see Subsections 2.3.2 and 4.3). We, therefore, postpone the investigation of the application of this approach as a possible coarse-grid solver to a later time. It should be noted here though that to make this approach feasible in the context of high-order methods, the polynomial degree of each element has to be reduced to low order in a sequence of p-coarsening steps (see Subsection 2.3.4).

[3] The development of robust and efficient matrix-free smoothers is still a widespread field of research [30, 50, 62, 82].

A modified method was proposed, which uses an upwind-weighted restriction operator $\boldsymbol{R}_{SUPG}$ with the effect that the resulting coarse matrix is comparable to Streamline-Upwind Petrov–Galerkin (SUPG) discretization without the need to rediscretize the system, *i.e.*:

$$\boldsymbol{A}_{SUPG} \approx \boldsymbol{R}_{SUPG}\boldsymbol{A}_{DG}\boldsymbol{P}_{CG}. \tag{6}$$

In [39], the authors applied p-multigrid to SUPG discretization and to DG discretization of the convection equation. They found that isotropic coarsening, as used in p-multigrid, does not dampen well the long-wavelength modes along and the short-wavelength modes normal to the streamlines. They proposed to use anisotropic multigrid coarsening and relaxation schemes that exert strong damping along the streamline. In order to find the steady-state solution of DG discretizations of the compressible Euler equations, p-MG has been applied in [42, 68].

Sundar et al. [95] conducted a comparison of p-MG and h-MG for high-order continuous finite elements. They observed only a small difference in the number of iterations. Besides this, they also examined another approach[4] to constructing a coarse grid: on the basis of high-order discretization, a linear space was constructed using the same set of nodes. This approximation, however, is found to be less efficient.

Stiller [91, 92] investigated Schwarz methods for smoothing in the context of p-multigrid for DG. Multiplicative and weighted additive Schwarz methods with element-centered and face-centered approaches have been tested also for high aspect ratios in 2D and 3D. Another strategy was investigated by Huismann et al. [50] for p-multigrid in the context of spectral-element methods: a vertex-based smoother, which utilizes 2^d elements as a subdomain.

AMG is not widespread in the context of high-order DG methods because the number of non-zeros in the system matrix scales polynomially $\sim \mathcal{O}((k+1)^{2d})$ (with the exponent 4 for 2D and 6 for 3D), the matrix is not necessarily diagonal-dominant, and AMG graph algorithms tend to coarsen much too rapidly due to the wide stencil character of DG methods [90].

Bastian et al. [12] and Siefert et al. [90] extended the given (non-smoothed and smoothed aggregation) AMG implementations for high-order DG methods and used them in the context of heterogenous elliptic problems and the Darcy problem. Both authors employed the auxiliary space idea to compute an approximate coarse-space correction in the subspace spanned by continuous, piecewise linear basis functions on the original mesh, using the existing AMG implementations. Before and after the coarse-grid approximation, smoother steps were performed on the high-order DG space.

These high-order AMG implementations require the user to provide the matrix on the finest grid $\boldsymbol{A}_f$, a pre- and post-smoother $\boldsymbol{S}$, and a transfer matrix $\boldsymbol{P}$. The coarse matrix $\boldsymbol{A}_c$ is constructed algebraically, using the Galerkin approach $\boldsymbol{A}_c = \boldsymbol{P}^T\boldsymbol{A}_f\boldsymbol{P}$. The authors highlighted that due to the need to provide the transfer matrix, which explicitly uses mesh information, this method is not fully algebraic. Siefert et al. [90] also demonstrated that, if it were a bottleneck, the multiplication with the global transfer matrix could be replaced by reference-element based transfers, requiring only a single local transfer matrix. Overlapping and non-overlapping versions of the Gauss–Seidel relaxation method were used as a smoother. The results of both papers show a linear increase in the numbers of V-cycles with increasing polynomial order k.

In a recent publication, Bastian et al. [13] presented the integration of their library into a matrix-free environment outlined by Müthing et al. [73]. The most expensive operations (matrix-vector multiplication, pre- and post-smoothing) on the finest grid were performed in a matrix-free way, and convection-dominated flow problems were considered. Only the matrix on the coarse grid has been assembled explicitly and was solved using AMG. Since only matrix-based components were replaced by equivalent matrix-free implementations, no beneficial effect on the number of V-cycles could be observed.

O'Malley et al. [80] revisited the auxiliary space idea for linear DG methods. Using either continuous piecewise linear or discontinuous piecewise constant finite elements on a coarse grid, they employed different publicly available AMG libraries (`AGMG` [74, 76–78], `ML` [31], `GAMG` [5, 6], `BoomerAMG` [44]). Similar to Helenbrook and Atkins [41], they were able to demonstrate that the continuous piecewise linear auxiliary space outperforms the discontinuous piecewise constant one due to its higher convergence rate. In [79], the same authors extended their investigations to discontinuous elements with quadratic polynomial shape function, using a p-MG step to reduce the order. In [79, 80], it was shown that `AGMG` is the best performing AMG library for first- and second-order DG methods.

For large-scale simulations, it is common to create a hierarchy of meshes by mesh refinement of a given macro mesh. Highly efficient libraries have been developed for managing such meshes [7] based on space-filling curves [4]. Kronbichler and Wall [59] demonstrated the suitability of this approach for simple geometries (hypercube with 1 macro cell, hypershell with 6 macro cells) in the context of high-order discontinuous Galerkin methods, using the

[4]This approach was previously also discussed in [16, 81].

Table 1: Categorization of multigrid solvers from relevant publications. Symbol legend: ✓= fulfilled, (✓) = partly fulfilled, no symbol = not fulfilled. The present work extends the h-multigrid solver presented in [59] with new features (highlighted in **orange** color). If the authors only consider polynomial orders up to $k \leq 2$, then the category 'high-order DG' is only partly fulfilled [79, 80, 87]. We categorize publications partly fulfilling the category 'p-MG' if the authors of the given publications use a two-level p-multigrid algorithm, *i.e.* jump directly from high-order to first-order auxiliary space [12, 13, 90].

	[3, 39–42, 68, 69]	[12, 90]	[13]	[79, 80]	[63, 94]	[87]	Kronbichler and Wall [59]
high-order DG	✓	✓	✓	(✓)		(✓)	✓
matrix-free			✓		✓	✓	✓
h-MG	✓				✓	✓	✓
p-MG	✓	(✓)	(✓)	✓		✓	✓
AMG (coarse-grid solver)		✓	✓	✓	✓	✓	✓

FE library `deal.II` [1]. For more complex geometries and adaptively refined grids in the context of continuous linear finite elements, Sundar et al. [94] presented the concept of combining a highly efficient matrix-free h-MG with AMG as a coarse-grid solver, using the library `p4est` [7] for meshes consisting of quadrilateral and hexahedral elements. Simulations for more than a quarter million cores have been presented, *i.a.* using a coarse mesh with 45 thousand macro cells. Lu et al. [63] has developed a similar concept for triangular and tetrahedral meshes. The recipients of the 2015 ACM Gordon Bell Prize, Rudi et al. [87], extended the algorithm presented in [94] for higher order and used it to simulate a highly heterogenous flow in the earth's mantle. They employed a mixed continuous-velocity/discontinuous-pressure element pair $(k, k-1)$ and used a hybrid spectral-geometric-algebraic multigrid solver (involving the transfer from discontinuous modal space to continuous nodal space as well as p-coarsening, h-coarsening, and algebraic coarsening) for preconditioning. The results considered only orders up to $k \leq 2$.

For solving steady-state problems, p-MG methods are also frequently used. They have been employed for the Euler equations [9, 10, 19, 42, 47, 61, 64, 65, 68, 75], the Navier–Stokes equations [29, 32, 83, 89], and the RANS equations [51]. Non-linear solvers for high-order problems in general are discussed by Brown [16].

The multigrid solvers from relevant publications discussed so far are categorized in Table 1, which summarizes and concludes this literature review. The categories are • *high-order DG*, • *matrix-free* operator evaluation, • *h-MG*, • *p-MG*, and • *AMG* as a coarse-grid solver. They are chosen in this way because we believe that an efficient solution of linear equation systems originating from high-order DG ($k \geq 3$) discretizations for non-trivial geometries can be obtained on modern CPU hardware only

1. if operator applications (incl. transfer operations) are performed in a matrix-free way on all but the coarsest level,
2. if the size of the coarse problem is decreased as much as possible during a sequence of p- and h-coarsening and during a possible rediscretization step to continuous elements, and
3. if on the coarsest level, the most efficient algebraic solver (AMG) is used.

The reader can conclude from Table 1 that individually none of the present multigrid solvers completely fulfills all categories. They do, however, when taken together as a whole. It should be noted here that Rudi et al. [87] presented a multigrid algorithm, which in principle fulfills all categories of Table 1. The authors of that publication, however, did not consider high-order DG in their experiments and only used polynomial orders up to $k \leq 2$ for continuous velocity space and $k \leq 1$ for discontinuous pressure space (*i.e.* their solver only 'partly fulfills' the category 'high-order DG' in Table 1). Hence, this Master's thesis aims at extending the multigrid solver [59][5] by applying well-established features from [3, 12, 13, 39–42, 63, 68, 69, 79, 80, 87, 90, 94] to an efficient hybrid multigrid solver, which completely fulfills all categories of Table 1, and intends to investigate the benefits of such a solver for both high-order continuous and high-order discontinuous Galerkin methods, considering also polynomial orders $k > 2$. In doing so, we intend to focus

[5] This solver has been modularized recently, extended to vector quantities, and successfully applied in solving a scalar convection-diffusion equation and the Navier–Stokes equations [23–28, 53].

on efficient matrix-free implementations of all components. This also entails a rediscretization of the given PDE on every level.

Such a hybrid solver allows us to revisit questions discussed in the reviewed publications from a new perspective and to quantify the exact influence of individual components:

- We are able to compare different p-sequences from the literature [41, 72]. As an extreme p-sequence, we also consider the two-level p-multigrid algorithm, in which a direct jump from high-order to first-order auxiliary space is performed, in order to mimic the results obtained by Bastian et al. [12, 13] and Siefert et al. [90]. In this way, we can quantify the exact benefit of the intermediate step (p-multigrid) in the overall algorithm.
- We can perform a fair comparison of h-MG and p-MG. We consider h-MG for a simple Poisson problem with only a single coarse-grid cell as the lower limit of the achievable time to solution. In contrast to Sundar et al. [95], we can also compare the parallel efficiency and strong-scaling limits since `deal.II` [1] is written for efficient, large-scale and massively-parallel simulations [7].

This Master's thesis relies on publicly available libraries such as the finite element infrastructure in `deal.II` [1] (its matrix-free support is indispensable), the library `p4est` [17] used by `deal.II` for efficiently managing distributed hierarchical meshes, and the distributed sparse-matrix support from the library `Trilinos` [45]. In particular, we used its AMG implementation from the package `ML` [31].

The remainder of this work is organized as follows. In Section 2, the key methods of the developed hybrid multigrid solver are presented. Section 3 provides some implementation details. Sections 4, 5, 6, and 7 show performance results of the implemented components and of the application of the new multigrid solver for Poisson, convection–diffusion, and incompressible Navier–Stokes problems. The conclusions of this Master's thesis are presented in Section 8.

2 Methods

We consider the Poisson equation as a model problem to introduce the reader to the developed hybrid multigrid solver:

$$-\Delta u = f \quad \text{in} \quad \Omega, \tag{7}$$

where u is the solution variable and f is the source term. The physical domain Ω is bounded by $\partial\Omega = \Gamma$, which is partitioned into the Dirichlet part Γ_D, where $u = g_\mathrm{D}$, and the Neumann part Γ_N, where $-\boldsymbol{n} \cdot \nabla u = g_\mathrm{N}$ is prescribed. The vector $\boldsymbol{n}$ donates the unit outer normal vector on the boundary Γ.

For discretization, we assume a tessellation $\mathcal{T}_h$ of the computational domain Ω_h into N_{el} elements Ω_e and use a mesh consisting of quadrilateral (2D) or hexahedral (3D) elements, which allows us to express the shape functions as tensor product of 1D shape functions. We assume an element to be the image of the reference element $[0,1]^d$ under a polynomial mapping of degree k_m, based on Gauss–Lobatto support points, which are placed according to a manifold description of the computational domain.

The bilinear forms that are associated with volume and face integrals of one element are donated by

$$(v,u)_{\Omega_e} = \int_{\Omega_e} v \odot u \,\mathrm{d}\,\Omega \quad \text{and} \quad (v,u)_{\partial\Omega_e} = \int_{\partial\Omega_e} v \odot u \,\mathrm{d}\,\Gamma, \tag{8}$$

where the operator $\odot$ symbolizes inner products.

2.1 High-order CG discretization

For continuous Galerkin methods, we assume a polynomial approximation of the solution of elements from the space

$$V_{h,g_\mathrm{D}}^{CG} = \left\{ v_h \in H^1(\Omega) \,:\, v_h|_{\Omega_e} \in \mathcal{Q}_k(\Omega_e)\, \forall \Omega_e \in \mathcal{T}_h \,\wedge\, v_h|_{\Gamma_\mathrm{D}} = g_\mathrm{D} \right\}. \tag{9}$$

H^1 is the space of square integrable functions with square integrable derivatives. It has the inner product $(x,y)_{\Omega_h}$ and the norm $||x||_{\Omega_h}$:

$$(x,y)_{\Omega_h} = \int_{\Omega_h} (xy + \nabla x \cdot \nabla y) \,\mathrm{d}\,\Omega \quad \text{and} \quad ||x||_{\Omega_h}^2 = \int_{\Omega_h} \left(x^2 + |\nabla x|^2\right) \mathrm{d}\,\Omega. \tag{10}$$

$\mathcal{Q}_k(\Omega^e)$ donates the spaces spanned by the tensor product of degree-k-polynomials in element Ω_e. We consider a basis representation by Lagrange polynomials: the nodes coincide with the $(k+1)$-points of the Gauss–Lobatto–Legendre quadrature rule. The solution space satisfies the boundary conditions g_D on Γ_D by projection or interpolation.

The discrete finite element version of the Poisson equation (7) is found by multiplication with a test function, integration over Ω_h, integration by parts of the left-hand side and insertion of the Neumann boundary condition. The final weak form is to find a function $u_h \in V_{h,g_\mathrm{D}^{CG}}$ such that

$$(\nabla v_h, \nabla u_h)_{\Omega_h} = (v_h, f)_{\Omega_h} - (v_h, g_N)_{\Gamma_N} \qquad \forall v_h \in V_{h,0_\mathrm{D}}^{CG}. \tag{11}$$

On each element, the left-hand side gives rise to an element stiffness matrix $\boldsymbol{A}_e$ and the right-hand side to an element load vector $\boldsymbol{b}_e$:

$$A_{e,ij} = (\nabla\phi_i, \nabla\phi_j)_{\Omega_e} \quad \text{and} \quad b_{e,i} = (\phi_i, f)_{\Omega_e} - (\phi_i, g_N)_{\Gamma_e \cap \Gamma_N}. \tag{12}$$

The local quantities $\boldsymbol{A}_e$ and $\boldsymbol{b}_e$ are assembled into the global stiffness matrix $\boldsymbol{A}_{CG}$ and the global load vector $\boldsymbol{b}_{CG}$ in the usual finite element way[6].

2.2 High-order DG discretization

For discontinuous Galerkin methods, we assume a polynomial approximation of the solution of elements from the space

$$V_h^{DG} = \left\{ v_h \in L_2(\Omega) \,:\, v_h|_{\Omega^e} \in \mathcal{Q}_k(\Omega^e)\, \forall \Omega_e \in \mathcal{T}_h \right\}. \tag{13}$$

Dirichlet boundary conditions are enforced weakly. No continuity over element boundaries is enforced. Only L_2 regularity of the solution is required. L_2 is the space of square integrable functions with inner product $(u,v)_{\Omega_h}$ and norm $||u||_{\Omega_h}$:

$$(u,v)_{\Omega_h} = \sum_{e=1}^{N_{el}} (u,v)_{\Omega_e} \quad \text{with} \quad (u,v)_{\Omega_e} = \int_{\Omega_e} uv \,\mathrm{d}\,\Omega \quad \text{and} \quad ||u||_{\Omega_h}^2 = (u,u)_{\Omega_h}. \tag{14}$$

[6] In this Master's thesis, we do not eliminate the Dirichlet rows and columns via static condensation to make the transfer between CG space and DG space simpler.

As a discontinuous Galerkin representation, we use the symmetric interior penalty (SIP) discontinuous Galerkin method. Two alternative derivations of SIP can be found in [2] and [59]. The weak form for a single element is as follows:

$$(\nabla v_h,\, \nabla u_h)_{\Omega_e} - (\nabla v_h,\, (u_h - u_h^*)\,\boldsymbol{n})_{\Gamma_e} - (v_h,\, \boldsymbol{\sigma}_h^* \cdot \boldsymbol{n})_{\Gamma_e} = (v_h,\, f)_{\Omega_e}\,, \tag{15a}$$

with appropriate value flux term u_h^* and gradient flux term $\boldsymbol{\sigma}_h^*$:

$$u_h^* = \{\{u_h\}\} \quad \text{and} \quad \boldsymbol{\sigma}_h^* = \{\{\nabla u_h\}\} - \tau[\![u_h]\!]. \tag{15b}$$

The average operator is given as $\{\{\square\}\} = (\square^- + \square^+)/2$, and the jump operator is given as $[\![u]\!] = u^- \otimes \boldsymbol{n}^- + u^+ \otimes \boldsymbol{n}^+$. The superscript $\square^-$ donates interior information at a face and the superscript $\square^+$ donates exterior information from the neighbor. The interior and the exterior side of a face can be chosen arbitrarily. Its normal $\boldsymbol{n}$ points in the same direction as $\boldsymbol{n}^-$ does, *i.e.* $\boldsymbol{n} = \boldsymbol{n}^- = -\boldsymbol{n}^+$. For each element, the penalty parameter τ_e is defined as:

$$\tau_e = (k+1)^2 \frac{A(\partial\Omega_e \setminus \Gamma_h)/2 + A(\partial\Omega_e \cap \Gamma_h)}{V(\Omega_e)}, \tag{16}$$

with volume V and surface area A [46]. The actual penalty parameter is chosen at internal faces as $\tau = \max(\tau_{e^-},\, \tau_{e^+})$ and at boundary faces as $\tau = \tau_e$.

Boundary conditions are imposed weakly by defining suitable extension values u^+ as a function of the boundary conditions and the inner solution u^-:

$$\begin{aligned} u^+ &= -u^- + 2g_\mathrm{D}, \quad & \nabla u^+ &= \nabla u^-, \quad & &\text{on } \Gamma_\mathrm{D}, \\ u^+ &= u^-, & \nabla u^+ \cdot \boldsymbol{n} &= -\nabla u^- \cdot \boldsymbol{n} - 2g_\mathrm{N}, & &\text{on } \Gamma_\mathrm{N}. \end{aligned} \tag{17}$$

The face integrals in Equation (15) can be split into inner-face terms $\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})$ and boundary-face terms $(\Gamma_\mathrm{D}, \Gamma_\mathrm{N})$. During insertion of the definitions of the boundary conditions, additional contributions of known quantities arise, which are moved to the right-hand side. The final weak form is to find a function $u_h \in V_h^{DG}$ such that

$$\begin{aligned} &(\nabla v_h,\, \nabla u_h)_{\Omega_e} - (\nabla v_h,\, (u_h - u_h^*)\,\boldsymbol{n})_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})} - (v_h,\, \boldsymbol{\sigma}_h^* \cdot \boldsymbol{n})_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})} \\ &\quad - (\nabla v_h,\, u_h)_{\Gamma_e \cap \Gamma_\mathrm{D}} - (v_h,\, +\nabla u_h \cdot \boldsymbol{n} - 2\tau u_h)_{\Gamma_e \cap \Gamma_\mathrm{D}} \\ &\quad = (v_h,\, f)_{\Omega_h} - (\nabla v_h,\, g_\mathrm{D}\boldsymbol{n})_{\Gamma_e \cap \Gamma_\mathrm{D}} + (v_h,\, 2\tau g_\mathrm{D})_{\Gamma_e \cap \Gamma_\mathrm{D}} - (v_h, g_\mathrm{N})_{\Gamma_e \cap \Gamma_\mathrm{N}} \qquad \forall v_h \in V_h^{DG} \wedge \forall e, \end{aligned} \tag{18}$$

where the red terms relate to the homogeneous parts and the blue terms to the inhomogenous parts of the boundary-face integrals. Inserting the numerical fluxes per Equation (15b) into the inner-face terms results in the following form:

$$-\left(\nabla v_h,\, (u_h^- - \{\{u_h\}\})\,\boldsymbol{n}\right)_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})} - (v_h,\, \{\{\nabla u_h\}\} \cdot \boldsymbol{n} - \tau[\![u_h]\!] \cdot \boldsymbol{n})_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})}\,, \tag{19}$$

which can be specialized for shape functions living on the element on the negative side of the face:

$$-\left(\nabla v_h^-,\, 0.5(u_h^- - u_h^+)\,\boldsymbol{n}\right)_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})} - \left(v_h^-,\, 0.5(\nabla u_h^- + \nabla u_h^+) \cdot \boldsymbol{n} - \tau(u^- - u^+)\right)_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})}\,, \tag{20a}$$

and for shape functions living on the element on the positive side of the face:

$$-\left(\nabla v_h^+,\, 0.5(u_h^- - u_h^+)\,\boldsymbol{n}\right)_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})} + \left(v_h^+,\, 0.5(\nabla u_h^- + \nabla u_h^+) \cdot \boldsymbol{n} - \tau(u^- - u^+)\right)_{\Gamma_e \setminus (\Gamma_\mathrm{D} \cup \Gamma_\mathrm{N})}\,. \tag{20b}$$

On each element, the cell integral on the left-hand side gives rise to an element matrix $\boldsymbol{A}_e$ and the right-hand side to an element load vector $\boldsymbol{b}_e$:

$$A_{e,\,ij} = (\nabla\phi_i, \nabla\phi_j)_{\Omega_e} \quad \text{and} \quad b_{e,\,i} = (\phi_i, f)_{\Omega_e} - (\nabla\phi_i,\, g_\mathrm{D}\boldsymbol{n})_{\Gamma_e \cap \Gamma_\mathrm{D}} + (\phi_i,\, 2\tau g_\mathrm{D})_{\Gamma_e \cap \Gamma_\mathrm{D}} - (\phi_i, g_\mathrm{N})_{\Gamma_e \cap \Gamma_\mathrm{N}}. \tag{21a}$$

Up to here, a certain similarity to CG can be discerned: the orange terms are identical to the definition of Equation (12). However, additional terms to the right-hand side vector (see Equation 18) and to the matrix $\boldsymbol{A}$ arise in the case of DG. On each inner face f, the face integrals on the left-hand side give rise to four matrices ($\boldsymbol{A}_f^{--}$, $\boldsymbol{A}_f^{-+}$, $\boldsymbol{A}_f^{+-}$, $\boldsymbol{A}_f^{++}$):

(21b)

$A_{f,\,ij}^{\square\square}$	$-$	$+$
$-$	$+\left(\phi_i^-,\, \tau\,\phi_j^-\right) - \left(\nabla\phi_i^-,\, \frac{\phi_j^-}{2}\,\boldsymbol{n}\right) - \left(\phi_i^-,\, \frac{\nabla\phi_j^-}{2} \cdot \boldsymbol{n}\right)$	$-\left(\phi_i^-,\, \tau\,\phi_j^+\right) + \left(\nabla\phi_i^-,\, \frac{\phi_j^+}{2}\,\boldsymbol{n}\right) - \left(\phi_i^-,\, \frac{\nabla\phi_j^+}{2} \cdot \boldsymbol{n}\right)$
$+$	$-\left(\phi_i^+,\, \tau\,\phi_j^-\right) - \left(\nabla\phi_i^+,\, \frac{\phi_j^-}{2}\,\boldsymbol{n}\right) + \left(\phi_i^+,\, \frac{\nabla\phi_j^-}{2} \cdot \boldsymbol{n}\right)$	$+\left(\phi_i^+,\, \tau\,\phi_j^+\right) + \left(\nabla\phi_i^+,\, \frac{\phi_j^+}{2}\,\boldsymbol{n}\right) + \left(\phi_i^+,\, \frac{\nabla\phi_j^+}{2} \cdot \boldsymbol{n}\right)$

where $\phi^\pm$ donates shape functions associated with test and solution functions of elements at the interior or the exterior of a face. On each boundary face b, the following contribution arises:

$$A_{b,\,ij} = -\left(\nabla\phi_i,\, \phi_j\right)_{\Gamma_e \cap \Gamma_\mathrm{D}} - \left(\phi_i,\, \nabla\phi_j \cdot \boldsymbol{n} - 2\tau\phi_j\right)_{\Gamma_e \cap \Gamma_\mathrm{D}}\,. \tag{21c}$$

These local quantities are assembled into the global stiffness matrix $\boldsymbol{A}_{DG}$ and the local load vector $\boldsymbol{b}_{DG}$ in the usual finite element way. Due to the disjointed character of the test and the solution functions, the element and face matrices correspond to blocks in the resulting block-sparse matrix.

2.3 Hybrid multigrid preconditioner

The discretization of the Poisson equation with continuous and discontinuous Galerkin methods leads to a linear equation system of the form:

$$\boldsymbol{A}_{\square}\boldsymbol{x} = \boldsymbol{b}_{\square} \qquad \square \in \{CG, DG\}, \tag{22}$$

as shown in Subsections 2.1 and 2.2. The hybrid multigrid algorithm presented here targets the solution of the linear equation system for CG and DG discretization. This Master's thesis focuses on the efficient solution of the equation system resulting from DG discretization, for which optionally – as an auxiliary problem – the continuous version is also solved.

The general multigrid solution process is summarized below (see also Section 1). For levels $l \geq 1$, the following steps are performed:

1. A pre-smoothing is carried out. Subsequently, a new defect $\boldsymbol{d}'$ is computed:
$$\boldsymbol{x} \leftarrow \mathcal{S}(\boldsymbol{d}), \qquad \boldsymbol{d}' \leftarrow \boldsymbol{d} - \mathcal{A}(\boldsymbol{x}). \tag{23a}$$
2. A coarse-grid correction is performed. To perform this, the defect vector is restricted to the coarse grid $l_c = l_f - 1$. The multigrid algorithm is called recursively on this defect vector. The result of the call, the coarse-grid correction $\boldsymbol{x}_c$, is prolongated to the fine grid and added to $\boldsymbol{x}$:
$$\boldsymbol{d}_c \leftarrow \mathcal{R}(\boldsymbol{d}'), \qquad \boldsymbol{x}_c \leftarrow \textbf{Multigrid}(\boldsymbol{d}_c), \qquad \boldsymbol{x} \leftarrow \boldsymbol{x} + \mathcal{P}(\boldsymbol{x}_c). \tag{23b}$$
3. Finally, the same substeps are performed as in step 1, but in reverse order:
$$\boldsymbol{d} \leftarrow -\boldsymbol{d} + \mathcal{A}(\boldsymbol{x}), \qquad \textbf{return}\, \boldsymbol{x} - \mathcal{S}(\boldsymbol{d}). \tag{23c}$$

The recursive multigrid algorithm is called initially with the residuum $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$, as a right-hand side vector, by a preconditioned conjugate gradient solver (see Section 2.3.1). The recursion is terminated if the base case, *i.e.* the coarsest level $l = 0$, is reached. This level has to be treated differently by calling a coarse-grid solver.

The key components of a multigrid solver are:

- an operator $\mathcal{A}$ on each level l, ▷ **matrix-based**/*matrix-free CG/DG*
- a smoother $\mathcal{S}$ on each level l, ▷ *i.a. (block) Jacobi, Chebyshev smoother*
- intergrid operators $\mathcal{P}$ and $\mathcal{R}$ between each level l and $l-1$, ▷ *matrix-free h-,* **p-, c-***transfer*
- a coarse-grid solver on level $l = 0$. ▷ *i.a.* **AMG**, *matrix-free conjugate gradient*

The comments (on the right) list and specify the implementations of the components in the developed hybrid multigrid solver[7]. Detailed descriptions of three individual components (smoother, coarse grid solver, and intergrid operators), as used in the simulations in this Master's thesis, are given in Subsections 2.3.2-2.3.4. Implementation details on the operator serving as the link between matrix-free and matrix-based implementations as well as between CG and DG implementations will be discussed in Section 3.

2.3.1 Preconditioned conjugate gradient

The preconditititioned conjugate gradient (PCG) method is employed as a solver. It is preconditioned by one V-cycle of the hybrid multigrid algorithm.

PCG updates sequentially the solution vector $\boldsymbol{x}$ and the residual vector $\boldsymbol{r}$ until the residual norm is smaller than a tolerance or the maximum number of iterations is exceeded:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \omega\boldsymbol{p}, \qquad \boldsymbol{r} \leftarrow \boldsymbol{r} - \omega\boldsymbol{q}, \tag{24a}$$

where ω is the step length and p the search direction. The step length is updated as follows:

$$\boldsymbol{v} \leftarrow \boldsymbol{A}\boldsymbol{p}, \qquad \omega \leftarrow \delta / \boldsymbol{p}^T\boldsymbol{v}, \tag{24b}$$

and the search direction thus:

$$\boldsymbol{v} \leftarrow \boldsymbol{M}^{-1}\boldsymbol{r}, \qquad \delta' \leftarrow \delta, \qquad \delta \leftarrow \boldsymbol{r}^T\boldsymbol{v}, \qquad \beta \leftarrow \delta/\delta', \qquad \boldsymbol{p} \leftarrow \beta\boldsymbol{p} - \boldsymbol{v}. \tag{24c}$$

The iteration is started with: $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{v} = \boldsymbol{M}^{-1}\boldsymbol{r}_0$, $\delta = \boldsymbol{r}_0^T\boldsymbol{v}$, $\boldsymbol{p}_0 = -\boldsymbol{v}$. $\boldsymbol{M}^{-1}$ is the preconditioner; in the specific instance of the Poisson problem, the presented hybrid multigrid solver is used. For detailed information on PCG, the reader is referred to Braess [14, 185 ff.].

[7] The terms in italics indicate implementations that had already existed in the given h-multigrid solver before the author of this Master's thesis extended it with new features (printed in bold).

2.3.2 Chebyshev smoother

We use a Chebyshev smoother by default on each level as pre- and post-smoother. It has the following update scheme [56, 98]:

$$\boldsymbol{t}^j = \rho^j \boldsymbol{t}^{j-1} - \theta^j \boldsymbol{M}^{-1} \boldsymbol{r}^{j-1}, \tag{25a}$$

$$\boldsymbol{x}^j = \boldsymbol{x}^{j-1} - \boldsymbol{t}^j, \tag{25b}$$

where $\boldsymbol{t}^j$ is a temporary vector, $\boldsymbol{r}^j = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^j$ is the residuum, $\boldsymbol{M}^{-1}$ is the preconditioner, and ρ^j and θ^j are scalar factors.[8] The iteration is started with $\boldsymbol{t}^0 = 0$ and the initial guess $\boldsymbol{x}^0$. The superscript $\square^j$ refers to the inner iterations within a single Chebyshev smoothing step. The number of inner iterations determines the degree of the Chebyshev polynomial. We assume $\boldsymbol{M}$ to correspond to the diagonal of the matrix $\boldsymbol{A}$. The diagonal is precomputed once during the setup phase, using the procedure described in Section 3.2, and is loaded on demand from main memory.

2.3.3 Algebraic coarse-grid solver

We use a conjugate gradient method preconditioned by a single AMG-V-cycle of `Trilinos`' `ML` [31] as a coarse-grid solver on a continuous, piecewise linear coarse space. Similarly to O'Malley et al. [79, 80], we construct this coarse space after explicitly performing a sequence of matrix-free p- and c-transfers. In this way, we can reach the best performance of `ML`.

We have chosen `ML` because of its integration into `deal.II` and regarded it as a black box – no tuning of the parameters was performed[9]. Table 2 below shows the settings used in all simulations with the `deal.II` nomenclature: A single V-cycle is carried out. On each level, two non-overlapping ILU sweeps are performed as pre- and post-smoothing. The coarse-grid problem is solved by the direct sparse solver KLU from the `Trilinos`' `Amesos` package.

Table 2: Settings of `ML` as used in all simulations with the `deal.II` nomenclature

elliptic:	true	constant modes:	-
higher-order elements:	false	smoother sweeps:	2
n cycles:	1	smoother overlap:	0
w cycle:	false	smoother type:	ILU
aggregation threshold:	1e-4	coarse type:	Amesos-KLU

A comparison with alternative coarse-grid solver variants is shown in Section 5.5.

2.3.4 Intergrid operators

Our multigrid algorithm requires prolongators in three instances: h-transfer (for h-MG), p-transfer (for p-MG), and c-transfer (for CG $\leftrightarrow$ DG). The implementation of the p-transfer and the c-transfer operators is the major contribution by the author of this Master's thesis.

The projection from a coarse grid to a fine grid can be generically expressed as a sparse matrix-vector multiplication with the prolongation matrix $\boldsymbol{P}_\square$:

$$\boldsymbol{x}_f = \boldsymbol{P}_\square \boldsymbol{x}_c = \sum_{e \in \{cells\}} \Pi_{e,f}^T \boldsymbol{P}_\square^{(e)} \Pi_{e,c} \left(\boldsymbol{x}_c\right) \qquad \square \in \{c,\, h,\, k\}. \tag{26}$$

As an alternative to the global prolongation, it is possible to express the prolongation algorithm locally, element by element. In that case, the following sequence of operations is performed for each element: element-local quantities are gathered, the prolongation is performed on the element with the element-local prolongation matrix $\boldsymbol{P}^{(e)}$, and the results are added to the global results. In doing so, $\Pi_{e,c}$ and $\Pi_{e,f}$ should be regarded as operators that, besides their actual task (to scatter/gather element-local quantities), also weight values of quantities shared by multiple elements on the fine grid.

[8] The factors ρ^j and θ^j are precomputed and set in such a way that high-frequency modes, corresponding to eigenvalues in the smoothing range $[\lambda_{max}/\alpha, \lambda_{max}]$ with $\alpha = 30$, are damped. An estimation of λ_{max} is found after a sequence of conjugate gradient iterations.

[9] The scope of the tasks of this Master's thesis involved the creation of the conditions for efficiently integrating the matrix-based AMG-algorithms of `ML`, provided by `deal.II`, in our matrix-free framework. This task comprised the following subtasks: extending discrete operators such that they can return their matrix representation (see Section 3.2), as well as implementing p- and c-transfer operators (see Section 2.3.4) to be able to transform the high-order DG-space to a piecewise linear CG-space, which is the most suitable space for `ML`. Tuning the parameters of `ML` should be investigated in a future study.

$$P_2^{h\to 2h} = \begin{bmatrix} +1.000 & & & \\ +0.375 & +0.750 & -0.125 & \\ & +1.000 & & \\ & +1.000 & & \\ -0.125 & +0.750 & +0.375 & \\ & & +1.000 & \end{bmatrix} \qquad P_{2\to 5}^{h} = \begin{bmatrix} +1.000 & & & \\ +0.675 & +0.415 & -0.090 & \\ +0.183 & +0.919 & -0.102 & \\ -0.102 & +0.919 & +0.183 & \\ -0.090 & +0.415 & +0.675 & \\ & & +1.000 & \end{bmatrix}$$

Figure 1: 1D Prolongation matrices for h-transfer of second-order elements and for p-transfer between second-order and fifth-order elements. The examples for h- and p-transfer have been chosen in such a way that they are comparable: the coarse and the fine level have the same number of degrees of freedom in both cases. Similarity of the matrix entries is clearly visible.

The element-local prolongation matrices $\boldsymbol{P}^{(e)}$ for CG and DG do not differ from each other. Whether CG or DG is used, is only reflected in the choice of $\Pi_{e,f}$ and $\Pi_{e,c}$.

The definition of the element-local projection matrix is as follows:

$$\boldsymbol{P}_{\square}^{(e)} = \boldsymbol{M}_{e,f}^{-1} \int_{\Omega_e} \boldsymbol{\phi}_f \boldsymbol{\phi}_c^T \,\mathrm{d}\Omega \qquad \text{with} \qquad \boldsymbol{M}_{e,f} = \int_{\Omega_e} \boldsymbol{\phi}_f \boldsymbol{\phi}_f^T \,\mathrm{d}\Omega, \tag{27}$$

where $\boldsymbol{\phi}_\square$ is the vector of the shape functions. The subscript f stands for the fine grid and c for the coarse grid.

All three transfer operations can be implemented in a matrix-free way. The h-transfer and the p-transfer are implemented using sum factorization. This requires the evaluation of the prolongation matrix for 1D – according to Equation (27) – only once at start-up for every transfer combination occurring in the particular simulation. Details on sum factorization in the context of matrix-vector multiplication $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{u}$ will be given in Section 3.1.

In accordance with Equation (26), the element-by-element definition of the restrictor can be expressed, assuming $\boldsymbol{R} = \boldsymbol{P}^T$, as follows:

$$\boldsymbol{r}_c = \boldsymbol{R}_\square \boldsymbol{r}_f = \boldsymbol{P}_\square^T \boldsymbol{r}_f = \sum_{e\in\{cells\}} \Pi_{e,c}^T \left(\boldsymbol{P}_\square^{(e)}\right)^T \Pi_{e,f}\left(\boldsymbol{r}_f\right) \qquad \square \in \{c,\, h,\, k\}, \tag{28}$$

applying the transpose of the element-local prolongation matrix.

H-transfer: The h-transfer prolongates the results from a coarse cell to its direct children with the same degree k. Projection matrices can be computed based on reference elements for the prolongation to each of the two children in 1D separately, *i.e.* $\boldsymbol{P}_h^{(1)}, \boldsymbol{P}_h^{(2)} \in \mathbb{R}^{(k+1)\times(k+1)}$. The separate projection matrices can be combined afterwards:

$$\boldsymbol{P}_h^{(e)} = \begin{bmatrix} \boldsymbol{P}_h^{(1)} \\ \boldsymbol{P}_h^{(2)} \end{bmatrix} \in \mathbb{R}^{2\cdot(k+1)\times(k+1)}. \tag{29}$$

Figure 1 shows the 1D prolongation matrix for $k = 2$.

P-transfer: The p-transfer is performed between elements of order k_f and order k_c, *i.e.* $\boldsymbol{P}_k^{(e)} \in \mathbb{R}^{(k_f+1)\times(k_c+1)}$. The coarsening sequence can be chosen arbitrarily. We only consider the following three p-sequences:

$$\textcircled{1}\; k_{i-1} = 1, \qquad \textcircled{2}\; k_{i-1} = \max(1, \lfloor k_i/2 \rfloor) \qquad \textcircled{3}\; k_{i-1} = \max(1, k_i - 1). \tag{30}$$

By default, sequence 2 is used because this approach halves the number of degrees of freedom in each direction just as h-MG does. The recursive formula is evaluated for up to order $k = 13$ in the table below (example: $9 \to 4 \to 2 \to 1$):

Table 3: P-transfer sequence ②

k_i	1	2	3	4	5	6	7	8	9	10	11	12	13
k_{i-1}	-	1	1	2	2	3	3	4	4	5	5	6	6

As an example, the element prolongation matrix between $k_f = 5$ and $k_c = 2$ is shown in Figure 1.

C-transfer: For the transfer between continuous and discontinuous elements, the element-local projection matrix simplifies due to $\phi_f = \phi_c$ to a simple identity matrix, *i.e.* $\boldsymbol{P}_c^e = \boldsymbol{I}_{k+1}$. The work of the c-transfer consists only of applying the two permutation matrices in sequence. The result is that the value of the nodes shared by neighboring elements in CG is simply injected into the corresponding nodes in DG during prolongation. In instances of restriction,

residuums belonging to the same vertex are summed up. This makes sense since test functions sharing the same supporting point have to be recombined during the transfer to CG.

In contrast to some authors, we perform each h-, p-, and c-transfer separately. Siefert et al. [90], for example, transfers directly between first-order continuous and high-order discontinuous space. Such a transfer, however, is equivalent to the composition of two transfer operators in the following way (example: $k_c = 1$ and $k_f = 7$):

$$\mathcal{P}_p^{1\to 7} \circ \mathcal{P}_c^1 = \mathcal{P}_{pc}^{1\to 7}, \tag{31}$$

and, if the p-transfer is split up in accordance with the p-transfer strategy introduced above, the transfer proposed by Siefert et al. [90] is equivalent to the following sequence of compositions:

$$\mathcal{P}_p^{3\to 7} \circ \mathcal{P}_p^{1\to 3} \circ \mathcal{P}_c^1 = \mathcal{P}_{pc}^{1\to 7}. \tag{32}$$

The latter formulation highlights the advantage of our method: by splitting up the transfer from high-order to low-order space, we can introduce cheap smoothers on the intermediate coarse grids.

3 Implementation

Section 2 described the discretization of the Poisson equation and the overall algorithm of the hybrid multigrid solver presented in this Master's thesis. Its main components are smoothers, coarse-grid solvers, and three types of intergrid transfer operators. Furthermore, a discrete operator $\mathcal{A}$, which hides the implementation details of the matrix $\boldsymbol{A}$, is needed to provide the minimum set of the following functionalities for the listed components:

- efficient matrix-vector multiplication $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{u}$ or $\boldsymbol{v} = \mathcal{A}(\boldsymbol{u})$ for the multigrid algorithm, for the smoothers, and for the preconditioned conjugate gradient method (see Equations (23), (24), (25)),
- construction of the inverse of the diagonal and/or the block diagonal of the matrix $\boldsymbol{A}$ for the smoothers,
- ability to simply rediscretize the PDE for a coarse mesh and for continuous or discontinuous elements of arbitrary order k, and
- construction of an explicit matrix representation of the matrix $\boldsymbol{A}$ for algebraic coarse-grid solvers, *e.g.* AMG.

Key aspects of efficient matrix-free matrix-vector multiplications for high-order CG and DG discussed by Kronbichler and Kormann [57, 58] are revisited in Subsection 3.1 from a matrix-based point-of-view. Based on the formalization of matrix-free and matrix-based algorithms in this review, we propose in Subsection 3.2 the construction of discrete operators $\mathcal{A}$ capable of providing also the remaining functionalities required by the multigrid solver presented here. Finally, in Subsection 3.3 we address the practical problems that arose during the integration of the algorithm of this solver into an actual finite element library (here: `deal.II` [1]) and the important role of the newly implemented transfer operators in this respect.

3.1 Efficient matrix-free matrix-vector multiplication

This subsection revisits the matrix-free matrix-vector multiplication for continuous Galerkin methods, closely following [57]. The approach used in that paper is applied here also to explain matrix-free matrix-vector multiplications for discontinuous Galerkin methods.

Continuous Galerkin Methods

The traditional way to perform a matrix-vector multiplication for a linear system of equations arising from a continuous Galerkin discretization consists of two distinct steps [88, p. 65]:

- Step 1: the construction of matrix $\boldsymbol{A}_{CG}$ by element assembly

$$\boldsymbol{A}_{CG} = \sum_{e \in \{cells\}} \Pi_e^T \boldsymbol{A}_e \Pi_e. \tag{33a}$$

  This step consists of the construction of the element stiffness matrices $\boldsymbol{A}_e$ for each element e (as per Equation (12)) and their insertion into the global stiffness matrix with the help of element permutation matrices Π_e[10].

- Step 2: the application of matrix $\boldsymbol{A}_{CG}$ to an input vector $\boldsymbol{u}$

$$\boldsymbol{v} = \boldsymbol{A}_{CG}\boldsymbol{u}. \tag{33b}$$

This approach – considered by Saad [88, p. 65] to be more appealing than the approach we will present next – is the standard for first-order finite element methods, yet it does become a bottleneck for high-order methods, as discussed in [59] and in Subsection 4.2 of this paper. The main reason for that is the increasing Number of Non-Zeros (NNZ) in the matrix. The NNZ per row is approximately proportional to the number of degrees of freedom per element and increases polynomially with the order $\mathcal{O}\left(k^d\right)$. Since sparse matrix-vector multiplications are inherently memory-bound, the increase in NNZ translates directly to a rapid decrease in the performance of matrix-vector multiplication for high-order elements. Readers interested in the topic 'sparse matrix formats' (*e.g.* Compressed Row Storage and Jagged Diagonal Storage) and their performance are referred to Hager and Wellein [37, p. 86-91].

Based on the observation that many iterative solvers do not need the actual matrix $\boldsymbol{A}$ but rather the effect of $\boldsymbol{A}$, *i.e.* the result of the matrix-vector multiplication $\boldsymbol{A}\boldsymbol{u}$, one can derive the idea that instead of precomputing the matrix, the

[10] For the sake of simplicity, we do not detail the constraints (Dirichlet boundary and hanging nodes) here. The reader may consider the imprinting of constraints as a subtask of Π_K. Interested readers are referred to [57].

Table 4: Left column: comparison between basis change with sum factorization and naïve basis change; Right column: comparison of the work of matrix-free and standard computation of the diagonal and block entries of matrices ($\square^*$: normalized by the number of degrees of freedom per block p^d; $\square^{**}$: normalized by the number of entries per block $p^{2\cdot d}$; $p = k+1$)

	sum-factorization	naïve
work*	$\mathcal{O}(p)$	$\mathcal{O}(p^d)$
memory traffic*	$\mathcal{O}(1)$	$\mathcal{O}(p^d)$
arithmetic intensity*	$\mathcal{O}(p)$	$\mathcal{O}(1)$

	matrix-free	standard
block**	$\mathcal{O}(p)$	$\mathcal{O}(p^d)$
diagonal*	$\mathcal{O}(p^{d+1})$	$\mathcal{O}(p^d)$

matrix entries could be "computed" on the fly. This idea corresponds to an interchange of the summations as shown here:

$$\boldsymbol{A}_{CG}u = \left(\sum_{e\in\{cells\}} \Pi_e^T \boldsymbol{A}_e \Pi_e\right) u = \sum_{e\in\{cells\}} \Pi_e^T \boldsymbol{A}_e \left(\Pi_e u\right), \tag{34}$$

and leads to an algorithm with smaller matrix-vector multiplications $\boldsymbol{v}_e = \boldsymbol{A}_e \boldsymbol{u}_e$ at the element level. To make this work, element local values $\boldsymbol{u}_e$ have to be extracted via $\boldsymbol{u}_e = \Pi_e \boldsymbol{u}$, and local contributions $\boldsymbol{v}_e$ have to be added into the global result vector via $\boldsymbol{v} = \Pi_e^T \boldsymbol{v}_e$. In this respect, the multiplication with the element stiffness matrix $\boldsymbol{v}_e = \boldsymbol{A}_e \boldsymbol{u}_e$ should not be considered strictly as an actual matrix-vector multiplication, but rather as an operator application:

$$\boldsymbol{A}_{CG}u = \sum_{e\in\{cells\}} \Pi_e^T \mathcal{A}_e \left(\Pi_e u\right), \tag{35}$$

since the structure of $\boldsymbol{A}_e$ can be exploited to construct a sequence of cheap operations applied to an input vector $\boldsymbol{u}_e$. To explain this with the aid of an example, let us turn back to our model problem, the Laplace operator. By inserting the definition of the element stiffness matrix of the Laplace operator into $\boldsymbol{v}_e = \boldsymbol{A}_e \boldsymbol{u}$ and replacing the integrals with quadratures, the result is:

$$v_e^{(i)} = \sum_{j=1}^{(k+1)^d} \underbrace{\sum_q \left(\hat{\nabla}\hat{\phi}_i\left(\hat{\boldsymbol{x}}_q\right)^T \cdot \left(J_k^{-T}\left(\hat{\boldsymbol{x}}_q\right)\right)^T \cdot w_q \cdot \left|\det J_k\left(\hat{\boldsymbol{x}}_q\right)\right| \cdot J_k^{-T}\left(\hat{\boldsymbol{x}}_q\right) \cdot \hat{\nabla}\hat{\phi}_j\left(\hat{\boldsymbol{x}}_q\right)\right)}_{A_{e,ij}} u_k^{(j)}, \tag{36}$$

where $\hat{\phi}_i$ donates the unit cell basis functions, $\hat{\boldsymbol{x}}_q$ the position of the quadrature point on the unit cell, $\hat{\nabla}$ the gradient on the unit cell, and $J_k(\hat{x}_q)$ is the Jacobian of the transformation from the unit to the real cell. Moving the outer loop inside results in the following form:

$$v_e^{(i)} = \sum_q \left(\hat{\nabla}\hat{\phi}_i\left(\hat{\boldsymbol{x}}_q\right)^T \cdot \left(J_k^{-T}\left(\hat{\boldsymbol{x}}_q\right)\right)^T \cdot w_q \cdot \left|\det J_k\left(\hat{\boldsymbol{x}}_q\right)\right| \cdot J_k^{-T}\left(\hat{\boldsymbol{x}}_q\right) \cdot \sum_{j=1}^{(k+1)^d} \hat{\nabla}\hat{\phi}_j\left(\hat{\boldsymbol{x}}_q\right) u_k^{(j)}\right). \tag{37}$$

Three distinct steps (highlighted in the equation in **orange**, **blue**, and **red**) that are executed in sequence can be identified and have been generalized in [57]. For the Laplace operator, they are:

1. computation of the gradients on the unit cell for all quadrature points,
2. determination of the gradients in real space and weighting of the contribution of each quadrature point, and
3. multiplication by unit-cell gradient of shape functions and summation over all quadrature points.

The complexity of step 2 is obviously $\mathcal{O}(1)$ per quadrature point. For steps 1 and 3, sum factorization [71, 81] can be employed. This scheme exploits the tensorial structure of the shape functions to reduce the complexity of the basis change to $\mathcal{O}(k+1)$ per degree of freedom. The overall complexity can be reduced from $\mathcal{O}((k+1)^{2d})$ to $\mathcal{O}((k+1)^{d+1})$ per element if sum factorization is used instead of a naïve matrix-vector multiplication. Additionally, the memory traffic is significantly reduced as shown in Table 4. This reduction takes place, although additional data structures regarding the mapping between real and reference space have to be loaded.

Discontinuous Galerkin Methods

For discontinuous Galerkin methods, besides cell contributions, inner-face and boundary-face matrix contributions also have to be computed. Following the matrix-based notation used for continuous Galerkin methods, we can express the

Table 5: Matrix-free notation for application of the Laplace operator

- $\mathcal{A}_k$: cell integral ($\rightarrow$ CG, DG):

$$\left(\nabla v^-,\, \nabla u^-\right)_{\Omega_e}$$

- $\mathcal{A}_f$: inner-face integral - contribution to $\square^-$ ($\rightarrow$ DG):

$$-\left(\nabla v^-,\, \frac{u^- - u^+}{2}\,\boldsymbol{n}\right)_{\Gamma_e\setminus(\Gamma_\mathrm{D}\cup\Gamma_\mathrm{N})} - \left(v^-,\, \frac{\nabla u^-\cdot\boldsymbol{n} + \nabla u^+\cdot\boldsymbol{n}}{2} - \tau\left(u^- - u^+\right)\right)_{\Gamma_e\setminus(\Gamma_\mathrm{D}\cup\Gamma_\mathrm{N})}$$

- $\mathcal{A}_f$: inner-face integral - contribution to $\square^+$ ($\rightarrow$ DG):

$$-\left(\nabla v^+,\, \frac{u^- - u^+}{2}\,\boldsymbol{n}\right)_{\Gamma_e\setminus(\Gamma_\mathrm{D}\cup\Gamma_\mathrm{N})} + \left(v^+,\, \frac{\nabla u^-\cdot\boldsymbol{n} + \nabla u^+\cdot\boldsymbol{n}}{2} - \tau\left(u^- - u^+\right)\right)_{\Gamma_e\setminus(\Gamma_\mathrm{D}\cup\Gamma_\mathrm{N})}$$

- $\mathcal{A}_b$: boundary-face integral ($\rightarrow$ DG):

$$\left(\nabla v^-,\, u^-\right)_{\Gamma_e\cap\Gamma_\mathrm{D}} + \left(v^-,\, -\nabla u^-\cdot\boldsymbol{n} + 2\tau u^-\right)_{\Gamma_e\cap\Gamma_\mathrm{D}}$$

original matrix-vector multiplication as the sum of three simpler matrix-vector multiplications:

$$\boldsymbol{A}_{DG}\boldsymbol{u} = (\boldsymbol{A}_C + \boldsymbol{A}_F + \boldsymbol{A}_B)\boldsymbol{u} = \boldsymbol{A}_C\boldsymbol{u} + \boldsymbol{A}_F\boldsymbol{u} + \boldsymbol{A}_B\boldsymbol{u}, \tag{38}$$

where $\square_C$ is the cell contribution, $\square_F$ is the inner-face contribution, and $\square_B$ the boundary contribution. The contribution of the cell stiffness matrix $\boldsymbol{A}_C$ is computed in exactly the same way as shown above for CG in Equation (21a):

$$\boldsymbol{A}_C u = \sum_{e\in\{cells\}}^{+} \Pi_e^T \mathcal{A}_e\left(\Pi_e\boldsymbol{u}\right). \tag{39}$$

The only difference is that another set of permutation matrices is used. The block entries contribute to disjoint blocks in the matrix. $\sum^{+}$ represents - analogically to the symbol $\biguplus$ from the set theory - disjoint additions. For inner faces, we obtain:

$$\boldsymbol{A}_F\boldsymbol{u} = \sum_{f\in\{faces\}} \left[\begin{array}{cc} (\Pi_f^-)^T & (\Pi_f^+)^T \end{array}\right] \left[\begin{array}{cc} \boldsymbol{A}_f^{--} & \boldsymbol{A}_f^{-+} \\ \boldsymbol{A}_f^{+-} & \boldsymbol{A}_f^{++} \end{array}\right] \left[\begin{array}{c} \Pi_f^- \\ \Pi_f^+ \end{array}\right] \boldsymbol{u}, \tag{40}$$

with Π_f^- and Π_f^+, being the permutation matrices of the degrees of freedom residing on the negative and the positive side of the face. The definition of A_f^{--}, A_f^{-+}, A_f^{+-}, and A_f^{++} can be found in Equation (21b). The same optimization steps used for the cell stiffness matrices can be applied here; we can rewrite the formula, using the operator $\mathcal{A}_f$:

$$\boldsymbol{A}_F\boldsymbol{u} = \sum_{f\in\{faces\}} \left[\begin{array}{cc} (\Pi_f^-)^T & (\Pi_f^+)^T \end{array}\right] \mathcal{A}_f\left(\left[\begin{array}{c} \Pi_f^- \\ \Pi_f^+ \end{array}\right] \boldsymbol{u}\right), \tag{41}$$

and apply sum factorization inside the operator. This makes the matrix-vector multiplication on cell faces highly efficient. The application of boundary faces is analogous to that of cell matrices as per Equation (21c):

$$\boldsymbol{A}_b\boldsymbol{u} = \sum_{b\in\{boundary\}}^{+} \Pi_b^T \mathcal{A}_b\left(\Pi_b\boldsymbol{u}\right). \tag{42}$$

The sequence in Equation (38), according to which cell, face and boundary evaluations (Equations (39), (41), and (42)) are performed after each other, should not be considered static. By contrast, cell, face and boundary evaluations are grouped in such a way that the data can be reused from the cache.

If no matrices are assembled, but the operators are applied directly, it is legitimate to drop the matrix notation and to introduce a notation inspired by the weak form of Equation (18). The notation includes explicitly face contributions on both sides of an inner face (see Table 5) and clearly shows which quantities (value, gradient) are needed on the quadrature points and what operations have to be performed on these points. This notation is the basis of the matrix-free implementation in `deal.II` [1] and in the application codes in which we use the presented hybrid multigrid solver.

3.2 Extracting a matrix

The discussion in Subsection 3.1 has shown that a matrix-free evaluation is a highly efficient way to perform matrix-vector multiplications for CG and DG. It is the reason why our application codes, and the operators in particular, are built around highly efficient matrix-free kernels. Let us go on now to discuss how the matrix-free implementation of the operators can be used to compute their matrix-based representation that is needed for the smoothers and for AMG. To be precise, we need the inverse of either the diagonal or the block diagonal of matrix $\boldsymbol{A}$ on every multigrid level, and we also need the actual explicit matrix representation of $\boldsymbol{A}$ only on the coarsest, lowest-order level.

In order to derive an efficient way of computing the global matrix $\boldsymbol{A}$ in a matrix-free framework in which operators are expressed as they are in Table 5, let us again turn to the element stiffness matrices as they are defined in Equations (12), (21a), (21b) and (21c). Based on the observation that the multiplication of an arbitrary matrix with a basis vector e_i (where all entries are zeros, only the ith entry is 1) returns the ith column of the matrix:

$$\textbf{column}(\boldsymbol{A}_e)_i = \boldsymbol{A}_e e_i, \tag{44}$$

it is obvious that the matrix-free operator $\mathcal{A}_e$ also does the same:

$$\textbf{column}(\boldsymbol{A}_e)_i = \mathcal{A}_e(e_i). \tag{45}$$

The full element matrix can be reconstructed by a sequence of basis-vector applications. This algorithm can be easily generalized for handling cell, inner-face and boundary-face element matrices:

$$\boldsymbol{A}_\square = [\quad \mathcal{A}_\square(e_i) \quad \textbf{for} \quad i \quad \textbf{in} \quad [0, \textbf{len}(\boldsymbol{u}_\square)[\quad], \tag{46}$$

with $\square \in \{c,\ f,\ b\}$. The computed element matrices can be stored in appropriate global data structures such as the diagonal in a vector, the block diagonal in a vector of full matrices, and the global stiffness matrix in a sparse matrix. Table 4 proves the efficiency of the presented approach for computing the full block of an element stiffness matrix. The computational complexity of this approach is significantly lower, compared to a standard implementation based on three-level nested loops consisting of iterations over all ansatz-functions, test-functions and quadrature points. The complexity estimate is valid for the computation of the explicit matrix representation as well as of the block diagonal.

For computing the diagonal of an element matrix, the same algorithm can be used as before, but only entries on the diagonal are kept, *i.e.*

$$\text{diag}(\boldsymbol{A}_\square) = [\quad \mathcal{A}_\square(e_i)[i] \quad \textbf{for} \quad i \quad \textbf{in} \quad [0, \textbf{len}(\boldsymbol{u}_\square)[\quad]. \tag{47}$$

This somewhat wasteful approach has a worse complexity than the standard implementation, as shown in Table 4. This implementation only has to iterate over all quadrature points and over all test-functions in a two-level nested loop (ansatz-function is chosen to be the same as the test-function). In practice, however, our approach turns out to be efficient due to the highly optimized building blocks (see Section 4.2).

The discussion of Subsections 3.1 and 3.2 concludes that:

1. given a DG implementation, it is straightforward to obtain a CG implementation by not looping over faces in a matrix-free implementation case or by not assembling face stiffness matrices in the case of matrix-based implementations, and that
2. given a matrix-free implementation, it is possible to construct a matrix-based implementation via a columnwise reconstruction of the matrix.

In other words, given a matrix-free framework for DG, it is straightforward to mimic matrix-free CG and the appropriate matrix-based versions. This advantageous connection between the four implementations is depicted in Figure 2. Based on the aforementioned conclusions, the author of this Master's thesis has restructured the discrete operators such that they, by providing the presented hybrid multigrid algorithm with a clearly defined interface, enable its functioning in a transparent way.

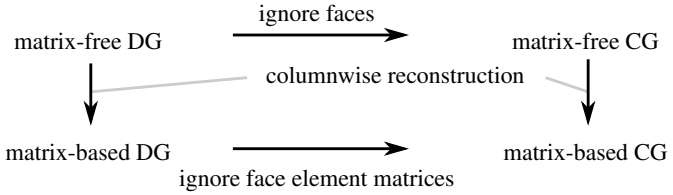

Figure 2: Relation of matrix-free/matrix-based and continuous/discontinuous Galerkin methods. Matrices can be constructed simply via columnwise reconstruction using matrix-free methods. DG can be simplified to CG by ignoring the flux terms on the faces.

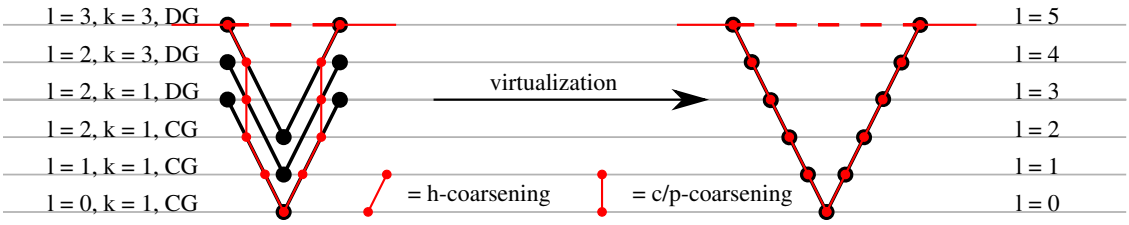

Figure 3: Virtualization of V-cycles belonging to separate dof-handlers to a single V-cycle

3.3 Working with existing FE infrastructure

This subsection describes how to integrate the presented hybrid multigrid solver into the Finite Element library `deal.II` [1]. The description applies to any high-order FE library as long as it supports h-multigrid as well as DG and CG discretization.

The FE library `deal.II` [1] has a strong support for h-multigrid. The user generally provides a coarse mesh consisting of macro cells. Each macro cell is refined adaptively and recursively by halving the cell in each direction, leading to a forest-of-trees of quadtrees (2D) or octrees (3D). The information regarding the hierarchy of cells is saved internally in the triangulation. Based on the triangulation and on a selected finite element (`FE_Q` for continuous and `FE_DGQ` for discontinuous elements) with a user-specified degree k, the degrees of freedom can be setup on the active cells[11]. The information is saved in a dof-handler. Optionally, degrees of freedom necessary for multigrid can be setup on local, non-active cells on all levels. This process leads to a fully populated dof-handler.

If one assumes that the additional data structures scale directly with the number of additional unknowns, memory consumption overhead of using h-multigrid implementation can be estimated using a geometric series:

$$\mathcal{M}_V = \sum_{i=0}^{l} \left(\frac{1}{2^d}\right)^i \leq \sum_{i=0}^{\infty} \left(\frac{1}{2^d}\right)^i = \frac{1}{1-0.5^d} = \begin{cases} 1.3333 & \text{for } d=2 \\ 1.1429 & \text{for } d=3. \end{cases} \tag{48}$$

It is obvious that the current implementation of the dof-handler in `deal.II` suits neither p-multigrid nor the approximation space idea because it does not support the creation of a hierarchy of changing element types that have varying polynomial orders. Writing a new dof-handler is impractical for many reasons. One of them is the large amount of useful utility functions provided by/for the dof-handler, many of which have been used in the implementation of the hybrid multigrid solver presented here.

As a way out, one could create fully populated dof-handlers for each element type (see in Figure 3, left-hand side). In this context, each multigrid level can be identified by the following triple:

$$(\texttt{level}, \texttt{degree}, \texttt{FE-type}) \in \mathbb{N}_0 \times \mathbb{N} \times \{\texttt{FE\_Q}, \texttt{FE\_DGQ}\}, \tag{49}$$

where multigrid levels with the same `degree` and `FE-type`, *i.e.* $(*, \texttt{degree}, \texttt{FE-type})$, share the same dof-handler. The label `level` specifies the level inside a dof-handler. The following transition functions have been implemented:

$$(\texttt{level}, \texttt{degree}, \texttt{FE-type}) \xrightarrow{h} (\texttt{level-1}, \texttt{degree}, \texttt{FE-type}), \tag{50a}$$

$$(\texttt{level}, \texttt{degree}, \texttt{FE-type}) \xrightarrow{p} (\texttt{level}, \textbf{next}(\texttt{degree}), \texttt{FE-type}), \tag{50b}$$

$$(\texttt{level}, \texttt{degree}, \texttt{FE\_DGQ}) \xrightarrow{c} (\texttt{level}, \texttt{degree}, \texttt{FE\_Q}). \tag{50c}$$

As the characters above the arrows indicate, these transitions correspond directly to the three transfer operators introduced in Section 2.3.4. During h-transfer, the level inside the dof-handler is changed. A p-transfer and a c-transfer lead to the change of the dof-handler, while keeping the level constant. By choosing appropriate transfer functions, the complexity of the existence of multiple V-cycles is hidden, and the user has the impression of working only on a single cycle. This hiding of complexity is called "virtualization" in Figure 3.

The fact that this naïve approach, as described above, does not introduce excessive overhead is shown – ignoring the FE type – in the following estimation of memory consumption for p-coarsening strategy ② for degree k and refinement

[11] Active cells are cells that are not refined and thus make up the fine grid.

level l (with the same assumption as in Equation (48)):

$$\mathcal{M}_F = \sum_{j=0}^{l_k} \sum_{i=0}^{l} \underbrace{\left(\frac{1}{2^d}\right)^j \left(\frac{1}{2^d}\right)^i}_{\substack{\text{cost of level} \\ (l-i,\, k_{l_k - j})}} = \left[\sum_{j=0}^{l_k} \left(\frac{1}{2^d}\right)^j\right] \cdot \left[\sum_{i=0}^{l} \left(\frac{1}{2^d}\right)^i\right] \leq \mathcal{M}_V^2 = \begin{cases} 1.7778 & \text{for } d = 2 \\ 1.3061 & \text{for } d = 3, \end{cases} \tag{51}$$

where $l_k = \lceil \log_2 (k+1) \rceil - 1$ is the number of p-coarsening levels, not counting the lowest-order level.

Because this approach leads at most to an additional overhead as large as h-multigrid does, which is acceptable in our use cases, it was employed in the implementation of the hybrid multigrid solver presented here.

4 Performance analysis of main multigrid components

This section provides a short introduction to the results presented in Sections 5-7. In Subsection 4.1, the hardware used for the simulations is described. Subsections 4.2 and 4.3 take a look at two components of the hybrid multigrid solver presented in this Master's thesis, the discrete operator and the transfer operators, from a performance point of view.

All C++ code was compiled using the GNU compiler gcc, version 7.3.0, with optimization target AVX2 (Haswell). The minimum runtime out of ten experiments is presented in the next sections.

4.1 Hardware

All simulations and performance tests of this Master's thesis were performed on the Linux-Cluster CoolMUC-2, provided by the Leibniz Supercomputing Centre (LRZ), in Garching, Germany. An overview of the configuration of the cluster, extracted from the website of LRZ[12], is shown in Table 6.

Table 6: Left: overview of the configuration of the Linux-Cluster CoolMUC-2; Right: bandwidth measurements for a single node (BW: bandwidth output by STREAM benchmark; BW†: bandwidth taking also read for ownership into account)

Architecture	Intel Xeon E5-2697 v3 ("Haswell")
Cores per node	28
RAM per node	64 GB
Max job nodes	60
Max job cores	1680

Function	BW [GB/s]	BW† [GB/s]
Copy	83	124
Scale	83	124
Add	88	118
Triad	93	124

The CoolMUC-2 cluster is equipped with an FDR14 Infiniband interconnect and 28-core Intel Xeon E5-2697 v3[13] Haswell-based nodes. The processor offers the AVX2 instruction set, which operates either on four double-precision or on eight single-precision floating-point numbers in parallel. The maximum accumulated bandwidth of a single node has been measured with the STREAM benchmark [70], which has been adjusted for MPI. The detailed results of these measurements are shown in Table 6. A maximum bandwidth of about 120GB/s was measured. Additional information on the thread topology and the cache topology of the nodes has been extracted by the tool `likwid-topology` from the `LIKWID`-suite [85, 96] and is shown in Appendix A.

Per default, simulations have been performed on a single node with one single-threaded MPI process per core, *i.e.* with 28 MPI processes. In Subsections 5.8 and 5.9, we will present strong- and weak-scaling results, in which we one by one increase the number of nodes to 60, the maximum number of nodes for a job.

For the throughput experiments analyzed in this section, the problem size of approximately 50 million unknowns was used such that the solution vectors (and all other global data) need to be fetched from main memory rather than from caches.

4.2 Discrete operator

In our code, the discrete operator, an implementation of the abstract class `OperatorBase`, hides the implementation details of matrix-free and matrix-based CG and DG. Besides providing the highly efficient matrix-free matrix-vector multiplication `vmult` for CG and DG on the basis of the description given in Section 3.1, the discrete operator also provides helper functions for constructing the (inverse) diagonal, the (inverse) block diagonal, and the explicit matrix representation of the given operator, based on the procedures described in Section 3.2. In the following, we will only consider the construction of the (inverse) diagonal and of the explicit matrix representation since only they are needed by the Chebyshev smoother and by AMG, used in this Master's thesis.

The (inverse) diagonal is generally computed once during the startup by the function `update`, according to Equation (47), and is stored in a distributed vector. The Chebyshev smoother reads, when needed, this vector from memory, performs an element-wise multiplication with the source vector `src`, and stores the result in `dst`. The vectors `src` and `dst` are generally selected to be the same so that the same bandwidth as in the case of `Add` and `Triad` in the STREAM benchmark can be expected (2 reads + 1 store).

[12] Overview of the cluster configuration: `https://www.lrz.de/services/compute/linux-cluster/overview/`

[13] Product specification by Intel:
`https://ark.intel.com/products/81059/Intel-Xeon-Processor-E5-2697-v3-35M-Cache-2_60-GHz`

The setup of a sparse matrix[14] is a two-step process: initially, the sparsity pattern is determined and then the entries in the matrix are determined. The latter step comprises the computation of the element matrices according to Equation (46) and the assembly of the element matrices into the global matrix. It should be noted that the current implementation computes the whole stiffness matrices in a vectorized fashion for 4/8 elements in parallel and only at the end inserts the entries into the matrix in one go[15].

Figure 4 shows the throughput of the application of the matrix-free `vmult`, of the application of the (inverse) diagonal, and of the sparse matrix-vector multiplication (SpMV) for all combinations of CG/DG, 2D/3D, and Cartesian/curved mesh for the Laplace operator. Additionally, the costs of the corresponding setup phases are shown. We define throughput as the processed number of degrees of freedom per time unit. In the case of the setup of a sparse matrix, the time unit corresponds to the time to build the matrix[16]. In Figure 4, the following observations can be made:

- For all cases, the application of the (inverse) diagonal has an order-independent throughput of ≈ 5GDoF/s, which exactly meets the expectation $(120[\text{GB/s}]/24[\text{Byte/DoF}])$.
- The throughput of the sparse-matrix based algorithms and of the computation of the (inverse) diagonal decreases with the complexity $\mathcal{O}\left(k^{2\cdot d}\right)$ for the considered degree range, indicating a work of $\mathcal{O}\left(k^{d}\right)$ per degree of freedom and per diagonal entry. While the work complexity for sparse-matrix based algorithms is consistent with the expectations, the work complexity for the computation of the (inverse) diagonal exceeds the expectations since theoretical estimations in Section 3.2 have shown a worse complexity $\mathcal{O}\left(k^{d+1}\right)$. This reveals the excellent implementation of the sum-factorization kernels in `deal.II`, which hides computation behind memory transfer. In the case of CG, a SpMV is faster than the computation of the diagonal; in the case of 3D DG, the time to setup the diagonal is comparable to the time of one SpMV.
- Matrix-free `vmult` is faster than SpVM for all but for first order elements. This is not surprising since the throughput of `vmult` is only weakly dependent on k.

Due to its fundamental role in our hybrid multigrid solver, the performance of the matrix-free `vmult` should be analyzed in more detail in the following. Table 7 summarizes the maximal throughput of `vmult` for all combinations, as extracted from Figure 4. Generally, the following observations can be made regarding the throughput: three-dimensional simulations are 30-50% slower than two-dimensional ones, using a non-Cartesian (curved) mesh leads to a penalty of 30-60%, discretization with DG results in a normalized overhead of 15-30%, and the throughput of curved CG and of curved DG is comparable.

Table 7: Maximum throughput [GDoF/s] of matrix-free `vmult` (in brackets: k_{max} with $1 \leq k \leq 9$)

CG	2D	3D
Cartesian	2.5 (6)	1.7 (4)
Curved	1.3 (9)	0.7 (9)

DG	2D	3D
Cartesian	1.7 (9)	1.1 (4)
Curved	1.2 (9)	0.6 (4)

Figure 5 shows the measurements of the cache transfer of the matrix-free `vmult`, taken with the tool `likwid-perfctr` of the `LIKWID`-suite. Measured transferred data volume per degree of freedom is shown for 3D, along with theoretically expected ideal values. The expected bandwidth to RAM is 120GB/s (see Subsection 4.1). Simple estimation of the data transfer per degree of freedom is provided in Table 8. It considers in all cases read for ownership as well as in the case of CG indices to be loaded, in the case of curved mesh mapping information ($\boldsymbol{J}$ and $|\boldsymbol{J}| \times w$), and in the case of DG additional face quantities. Measurements match well with expectations. Differences for DG might be related to an increase in MPI communication.

[14] We use sparse matrix for both CG and DG. CRS, a sparse-matrix format used in Trilinos [31], has to store the actual non-zero matrix entries as well as following information in two additional data structures [37]: column indices of each non-zero matrix entry and pointers to the start of each row. In the case of DG, a reasonable alternative would be to use a block-sparse matrix, which makes storing the column indices of each non-zero entry superfluous. This would lead to halving of the data volume to load and consequently to a speedup of approximately 2 for sparse matrix-vector multiplications (SpMV) with single-precision floating point numbers. Experiments have shown that it is true, however, setting up a block-sparse matrix in `deal.II` [1] is significantly slower than setting up a simple sparse matrix. This observation and the fact that we use matrix-based algorithms only on the coarsest level, lead us to use the suboptimal data structure for now and to postpone the integration of block-sparse matrices as an optimization strategy when the algebraic coarse problem becomes a bottleneck.

[15] Since we reconstruct element matrices columnwise, a more sound approach would be not to wait until the end of the construction of the whole element matrix, but rather to insert entries into the matrix columnwise as soon as they are available.

[16] In contrast to Kronbichler and Wall [59], we normalize by the actual number of degrees of freedom.

Table 8: Expected data transfer per degree of freedom during matrix-free `vmult` (for V=double and I=float: [V]=8 and [I]=4; numbers in **red** indicate 'read for ownership')

	Cartesian	curved
CG	$\frac{(2+\mathbf{1})\cdot k^d[\mathrm{V}]+1\cdot(k+1)^d[\mathrm{I}]}{k^d}\cdot\frac{\text{Byte}}{\text{DoF}}$	$\frac{(2+\mathbf{1})\cdot k^d[\mathrm{V}]+(k+1)^d\cdot\left(1[\mathrm{I}]+d^2[\mathrm{V}]+1[\mathrm{V}]\right)}{k^d}\cdot\frac{\text{Byte}}{\text{DoF}}$
DG	$(2+\mathbf{1})[\mathrm{V}]\cdot\frac{\text{Byte}}{\text{Dof}}$	$\frac{\left((2+\mathbf{1}+d^2)\cdot(k+1)^d+d\cdot(k+1)^{d-1}(2\cdot d+1)\right)[\mathrm{V}]}{(k+1)^d}\cdot\frac{\text{Byte}}{\text{DoF}}$

Table 9: Expected memory transfer per degree of freedom for the transfer operators (for V=double and I=float: [V]=8 and [I]=4; numbers in **red** indicate 'read for ownership'

	restriction	prolongation
CG-DG	$\frac{(k+1)^d[\mathrm{V}]+1[\mathrm{I}]+\mathbf{2}\cdot k^d[\mathrm{V}]+k^d[\mathrm{I}]}{(k+1)^d}\cdot\frac{\text{Byte}}{\text{Dof}}$	$\frac{\mathbf{2}\cdot(k+1)^d[\mathrm{V}]+1[\mathrm{I}]+k^d[\mathrm{V}]+k^d[\mathrm{I}]}{(k+1)^d}\cdot\frac{\text{Byte}}{\text{Dof}}$
P (DG)	$\frac{(k+1)^d[\mathrm{V}]+\mathbf{2}\cdot(\lfloor k/2\rfloor+1)^d[\mathrm{V}]}{(k+1)^d}\cdot\frac{\text{Byte}}{\text{Dof}}$	$\frac{\mathbf{2}\cdot(k+1)^d[\mathrm{V}]+(\lfloor k/2\rfloor+1)^d[\mathrm{V}]}{(k+1)^d}\cdot\frac{\text{Byte}}{\text{Dof}}$
P (CG)	$\frac{k^d[\mathrm{V}]+\mathbf{2}\cdot\lfloor k/2\rfloor^d[\mathrm{V}]+(k+1)^d[\mathrm{I}]+(\lfloor k/2\rfloor+1)^d[\mathrm{I}]}{k^d}\cdot\frac{\text{Byte}}{\text{Dof}}$	$\frac{\mathbf{2}\cdot 2\cdot k^d[\mathrm{V}]+\lfloor k/2\rfloor^d[\mathrm{V}]+(k+1)^d[\mathrm{I}]+(\lfloor k/2\rfloor+1)^d[\mathrm{I}]}{k^d}\cdot\frac{\text{Byte}}{\text{Dof}}$
H (DG)	$\frac{2^d[\mathrm{V}]+\mathbf{2}\cdot 1[\mathrm{V}]}{2^d}\cdot\frac{\text{Byte}}{\text{Dof}}$	$\frac{\mathbf{2}\cdot 2^d[\mathrm{V}]+1[\mathrm{V}]}{2^d}\cdot\frac{\text{Byte}}{\text{Dof}}$

In the case of 3D DG, a drop in throughput and an increase in loaded data volume per degree of freedom can be observed for $k \geq 5$. A calculation by hand shows that the memory consumption of the loaded data of 4/8 elements is $(k+1)^3 \cdot 32\text{Byte} \approx 6.9\text{KB}$ for $k = 5$. This value is significantly lower than the size of the CPU private L1-cache (32KB, see Appendix A), indicating that the drop in throughput might be due to non-optimal reusage of data already loaded for cell evaluation in cases of face evaluation.

The observation that matrix-based matrix-vector multiplication for CG is at least as fast as the matrix-free version for linear elements (see Figure 4) justifies – from a performance point-of-view – the change from matrix-free to matrix-based implementation on the coarsest level using linear elements in the developed hybrid multigrid solver.

For the sake of completeness, a Roofline model [43, 99] is shown in Figure 6 for the given hardware. Performance ceilings are inserted, based on the maximum memory bandwidth and on the theoretical peak performance, which can be estimated as follows:

$$\mathrm{P}_{\max} = \underbrace{\frac{\text{Instruction}}{\text{Time}}}_{\text{Clock frequency}} \times \underbrace{\frac{\text{Flops}}{\text{Instruction}}}_{\text{ILP \& SIMD}} \times \text{Core count}. \tag{52}$$

With 2.6GHz, 16Flops/Instr.[17], and 28 cores, we get $\mathrm{P}_{\max} = 1164\text{GFLOP/s}$. The `vmult`-operation of the Laplace operator is largely memory-bound, and nearly a quarter of peak performance can be reached.

4.3 Transfer operators

The hybrid multigrid solver presented here has three transfer operations (h-, p-, c-transfer), each with two methods: `restrict_and_add` and `prolongate`. In the following, we take a look at the performance of the p-transfer and the c-transfer operators. Similarly, as in Subsection 4.2, we compare measured and expected throughput for 2D and 3D. Results are independent of the type of mesh since mapping information does not have to be loaded. The measured values are plotted with solid lines in Figure 7. The dashed lines represent the expected throughput, which are based on the formulas presented in Table 9.

[17]The value 16 can be derived with the following computation: 2 FP units × 2 (for FMA) × 4 SIMD lanes for DP (see `https://en.wikichip.org/wiki/intel/microarchitectures/haswell_(client)`)

General observation based on Figure 7 is that the restriction is faster than the prolongation in all cases. The reason for this is that, in cases of prolongation, the destination vector is living on the fine grid, containing more degrees of freedom, *i.e.* read for ownership leads to the fact that these degrees of freedom are touched twice in main memory, resulting in a lower overall performance. The following further observations can be made:

- P-transfer, in cases of DG, matches well with expected values. The maximum throughput for restriction is $\geq$ 10GDoF/s, which is higher than the application of the (inverse) diagonal. The reason for this is significantly less read for ownership. The throughput of prolongation is 6-7GDoF/s.
- P-transfer, in cases of CG, also matches well with expected values for high order. The discrepancy between the results and the expectations for low order is probably because of lower memory bandwidth due to irregular memory access (which can also be observed for `vmult`). Since indices for all degrees of freedom on the coarse and on the fine space have to be loaded, the throughput of p-transfer for CG is continuously lower than for DG. Additionally, in the cases of prolongation an explicit zeroing of the destination vector is required because the degrees of freedom on the fine grid are shared.
- C-transfer has overall the lowest throughput of all three cases and does not match with expected values at all ($\geq 50\%$ difference). This is due to the implementation limitations of `deal.II`, which does not allow for work on vectors from DG and CG space (with ghost cells) at the same time with the same `Matrixfree` object. An explicit copy of the data, contained in the DG vector, to the right format is required[18].

In Section 5.4, we will discuss the cost of performing a single transfer operation in the light of the total cost of the complete multigrid algorithm.

[18] An alternative implementation, which did not require the creation of a temporal array, showed that it is possible to come close to the expected throughput also in cases of the c-transfer.

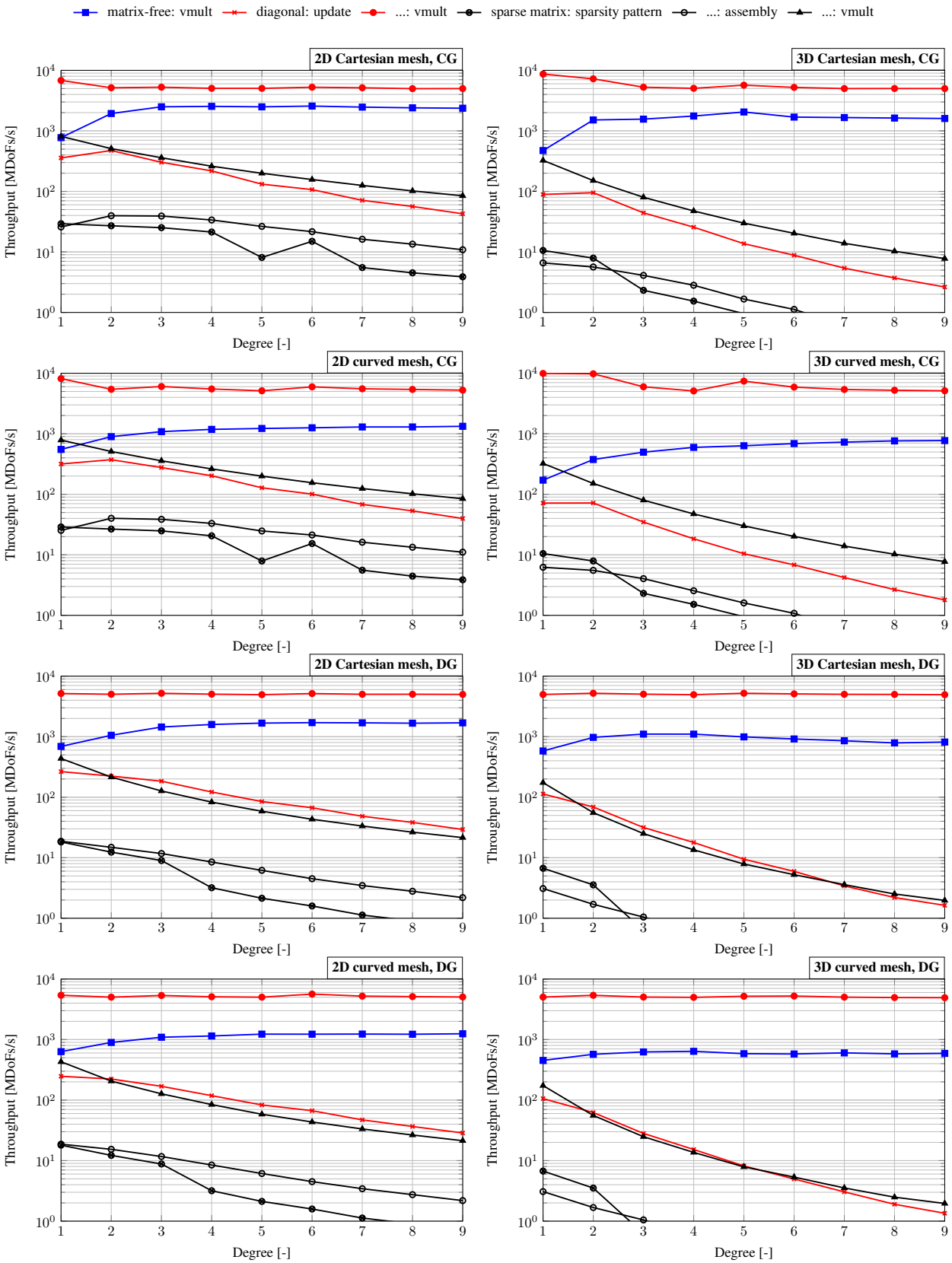

Figure 4: Performance of matrix-free and matrix-based operations of the Laplace operator, provided for all combinations in the parameter space {2D, 3D}×{CG, DG}×{Cartesian, curved mesh}

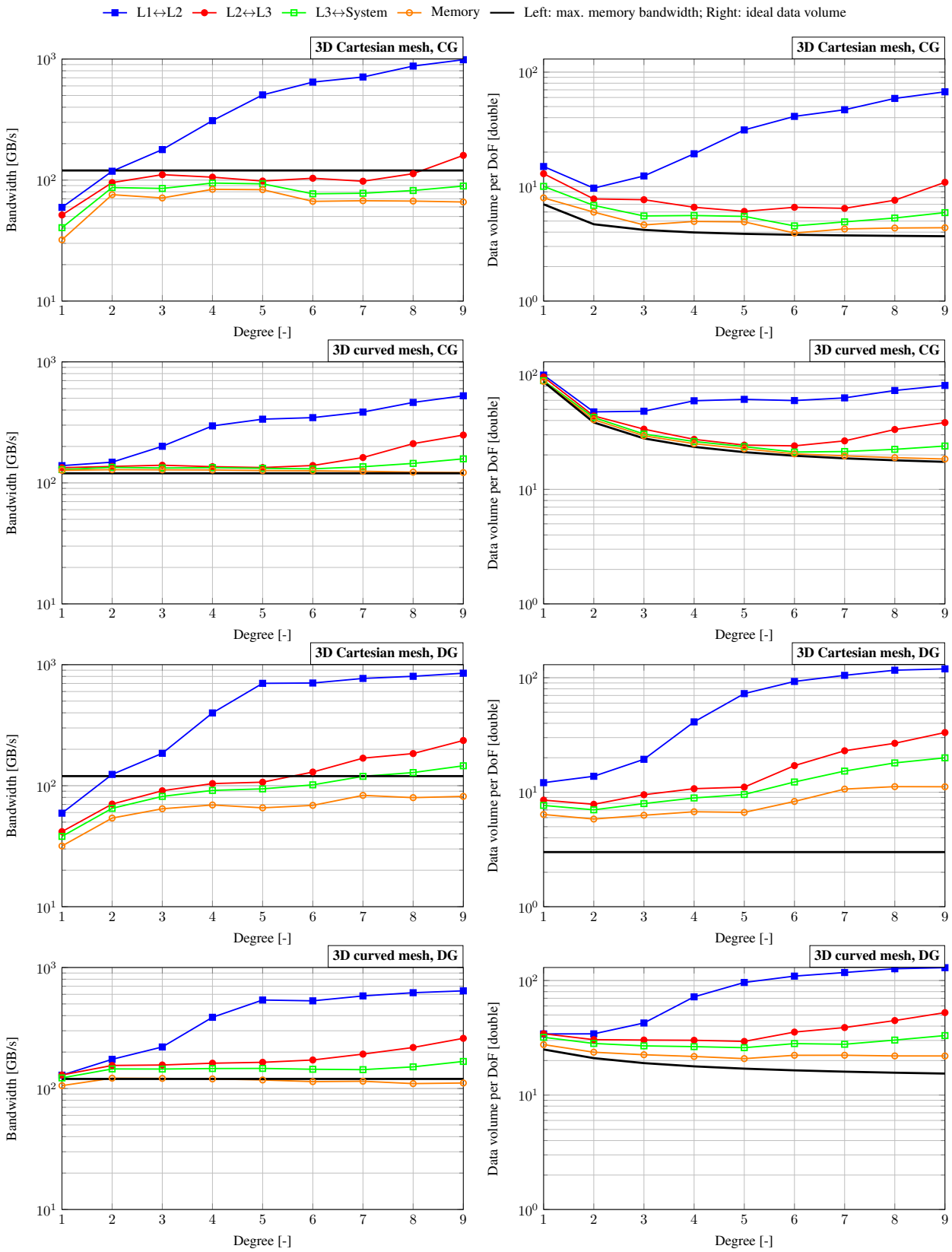

Figure 5: Cache transfer (bandwidth and data volume) for 3D Laplace operator, measured with LIKWID[85, 96]

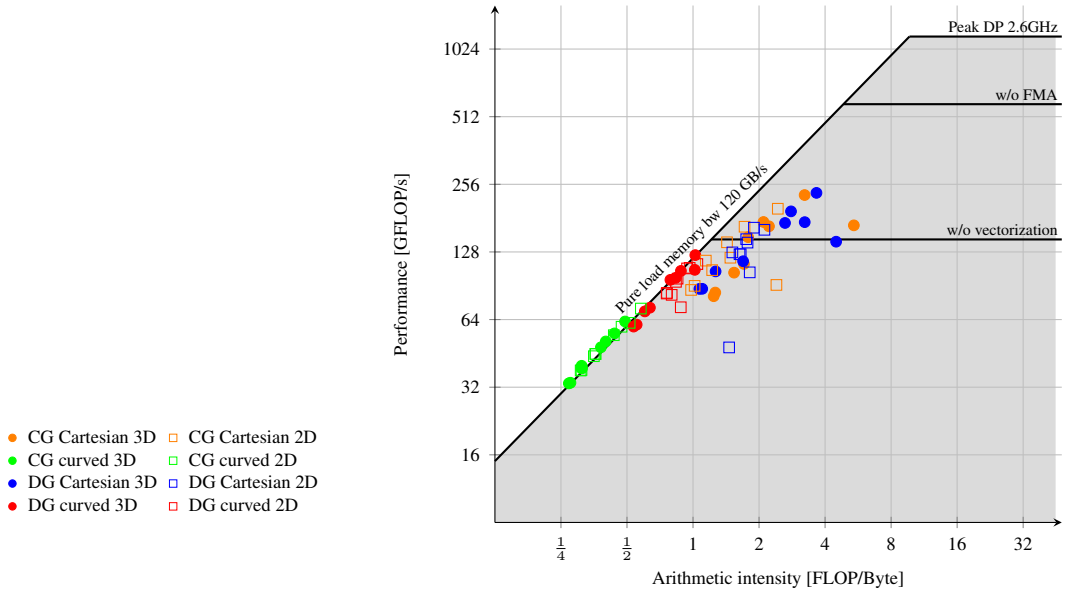

Figure 6: Roofline model for matrix-free `vmult` of the Laplace operator with $P_{max} = 1164$GFLOP/s for $1 \leq k \leq 9$.

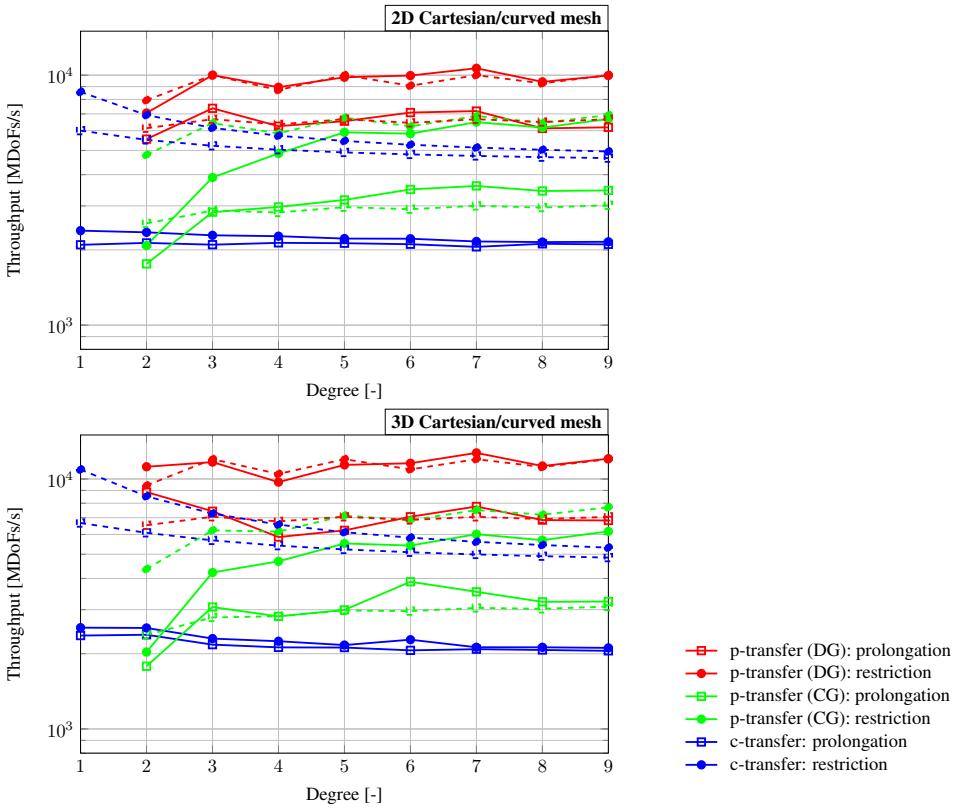

Figure 7: Performance of p- and c-transfer for 2D and 3D. Dashed lines indicate the expected values according to Table 9. Excellent agreement can be observed for p-transfer. The reason for the significant difference between expectations and measurements ($> 50\%$) in cases of c-transfer is the current implementation, in which a temporal copy of the degrees of freedom on the DG grid is created.

5 Application: Poisson problem

This section investigates the performance of the developed hybrid multigrid solver in the context of the Poisson problem as stated in Equation (7). In Subsection 2.1 and 2.2, this equation has been discretized for continuous and discontinuous finite element methods.

Subsection 5.1, giving a problem description, introduces the setup of the experiments, and Subsection 5.2 outlines the default configuration of the hybrid multigrid solver. The convergence behavior of the multigrid solver is analyzed in Subsection 5.3. Subsection 5.4 discusses node-level performance. It investigates the costs of each component of the multigrid algorithm, identifying possible variants to improve the performance. Subsections 5.5-5.7 analyze these variants and their benefits, considering different ways to solve the coarse-grid problem and various p-coarsening strategies. Finally, Subsections 5.8 and 5.9 take a look at the parallel performance of the developed hybrid multigrid solver by investigating the strong and weak scalability, using up to 1680 CPUs.

5.1 Problem description

We consider the Poisson equation with analytical solution [59]

$$u(\boldsymbol{x}) = \left(\frac{1}{\alpha\sqrt{2\pi}}\right)^d \sum_{j=1}^{3} \exp\left(-\left\|\boldsymbol{x}-\boldsymbol{x}_j\right\|^2/\alpha^2\right), \tag{53}$$

given as a sum of three Gaussians centered at the positions

$$\boldsymbol{x}_j = \left\{(-0.5,\,0.5,\,0.25)^T,\,(-0.6,\,0.5,\,-0.125)^T,(0.5,\,-0.5,\,-0.125)^T\right\}, \tag{54}$$

and of width $\alpha = \frac{1}{5}$. Equation (53) is visualized in Figure 8 for 2D. The corresponding source term is given by:

$$f(\boldsymbol{x}) = \left(\frac{1}{\alpha\sqrt{2\pi}}\right)^d \sum_{j=1}^{3} \left(2\cdot d - 4\left\|\boldsymbol{x}-\boldsymbol{x}_j\right\|^2/\alpha^2\right)\cdot\frac{\exp\left(-\left\|\boldsymbol{x}-\boldsymbol{x}_j\right\|^2/\alpha^2\right)}{\alpha^2}. \tag{55}$$

The Poisson problem is solved on the unit cube $\Omega = (-1,+1)^d$ with $d \in \{2,\,3\}$ and homogenous Dirichlet boundary conditions. The mesh is obtained by globally refining one macro cell l times. Points are placed according to the manifold:

$$x_i' = x_i + \alpha \prod_{0\le j\le d} \sin\left(0.5\cdot\beta\cdot\pi\cdot(x_i+1)\right), \tag{56}$$

with $\alpha = 0.1$ and $\beta = 2.0$, rendering a non-Cartesian mesh. Spatial discretizations with continuous and discontinuous elements of order $3 \le k \le 13$ are considered. The total number of degrees of freedom for CG is $\left(2^l\cdot k_f\right)^d$ and for DG $\left(2^l\cdot(k_f+1)\right)^d$.

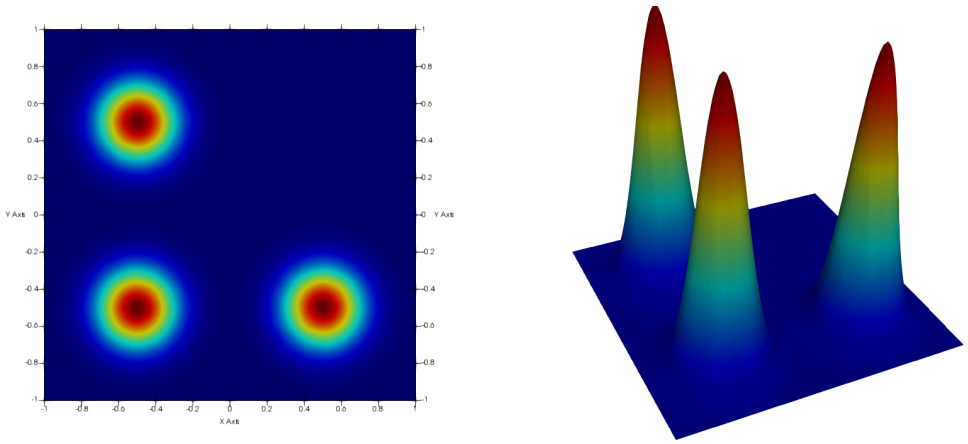

Figure 8: Visualization of the function given by Equation (53) for 2D

Table 10: Range of number of cycles n_{10} for all configurations, extracted from Tables 11 and 12

CG	2D	3D
p-MG	5-7	5-10
h-MG	5-10	5-11

DG	2D	3D
p-MG	7-16	11-23
h-MG	8-16	11-22

5.2 Default configuration of hybrid multigrid solver

For solving the Poisson problem[19], a preconditioned conjugate gradient solver is used as a solver. This solver is iterated until the L_2-norm of the residual has been reduced by 10^{-8} compared to the residual norm of the initial guess[20] in the conjugate gradient solver or the absolute value of the residual norm goes below 10^{-20}. It is preconditioned either by a matrix-free h-multigrid preconditioner or by a matrix-free p-multigrid preconditioner. Since multigrid is only used for preconditioning, the whole V-cycle is done in single precision. A matrix-free Chebyshev smoother (with polynomial degree 5, smoothing range 20, and 20 iterations eigenvalue calculation) is employed as a smoother, which uses the diagonal of the system matrix as a preconditioner.

To construct coarser levels, two methods are employed: h-coarsening and p-coarsening. In cases of p-coarsening, the p-coarsening strategy $k_{i-1} = \max(1, \lfloor k_i/2 \rfloor)$ is applied.

PCG (with absolute tolerance 10^{-20} and relative tolerance 10^{-2}) is also used as a coarse-grid solver. In cases of h-multigrid, PCG is preconditioned by the diagonal of the coarse matrix and in cases of p-multigrid, by AMG (see Subsection 2.3.3 for the configuration of AMG).

In the case of DG and p-multigrid, we switch from linear DG space to linear CG space via c-transfer for linear elements. Since we consider the linear DG space to be a multigrid level on its own, we also perform pre- and post-smoothing on this level.

5.3 Convergence

This subsection compares the convergence dependency of the developed hybrid multigrid solver on increasing degree k and refinement l in four cases for 2D and 3D: CG + h-MG and CG + p-MG as well as DG + h-MG and DG + p-MG. We consider h-multigrid with one macro cell as an optimal reference.

Tables 11 and 12 summarize the results of performed simulations for polynomial degrees $3 \leq k \leq 12$ as well as for global refinements $1 \leq l \leq 9$ (2D) and $1 \leq l \leq 5$ (3D), using three assessment criterions: average multigrid convergence rate after the n$^{\text{th}}$ cycle [92]:

$$\rho = \sqrt[n]{||\boldsymbol{r}_n||_2/||\boldsymbol{r}_0||_2}, \tag{57a}$$

the number of cycles n_{10} to reduce the absolute norm of the residuum by ten orders [92]:

$$n_{10} = \lceil -10/\lg \rho \rceil, \tag{57b}$$

and the throughput

$$r = \frac{\text{actual number of dofs}}{\text{time to solution}}. \tag{57c}$$

The following observations are stated in Tables 11 and 12:

- All solver variants converge, even though we rediscretized the operator on every level and used SIP in the case of DG. The latter observation matches well with results of Helenbrook and Atkins [40] since we also adjust the penalty parameter on every level according to the polynomial degree.
- The difference in the number of iterations between h-MG and p-MG is small for both CG and DG, as had been shown already by Sundar et al. [95] for continuous finite elements. Generally, h-MG is faster for low-order problems ($k < 5$). For high-order problems, p-MG needs 1-3 iterations less due to its better order independence, making its throughput higher, as also shown by Figure 10.

[19] The default configuration described here has been also used for solving the convection–diffusion problem (Section 6) and the Navier–Stokes equations (Section 7) with small adjustments. These adjustments are discussed in the corresponding sections.

[20] All conjugate gradient solvers are started with a zero initial guess.

Table 11: Convergence table for $3 \leq k \leq 7$

			$k=3$				$k=4$				$k=5$				$k=6$				$k=7$			
			CG		DG		CG		DG		CG		DG		CG		DG		CG		DG	
		#refs	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG
2D	n_{10}	1	5	5	8	8	5	5	7	8	5	5	8	9	5	5	8	9	5	5	9	10
		2	5	5	9	9	5	5	8	9	5	5	10	10	5	5	10	11	6	6	12	12
		3	6	5	10	9	5	5	8	9	6	5	10	11	6	5	11	11	6	6	12	13
		4	6	5	9	9	5	5	7	9	6	5	10	11	6	5	10	11	6	6	11	13
		5	7	5	9	9	5	5	7	9	6	6	10	10	6	6	9	10	7	6	10	12
		6	6	5	9	9	5	5	7	9	6	6	9	10	6	6	9	10	7	6	10	11
		7	7	5	8	9	5	5	7	8	6	6	9	10	6	6	8	10	7	6	10	11
		8	6	5	8	8	5	5	7	8	6	6	9	9	6	6	8	10	7	6	10	11
		9	6	5	8	8	5	5	7	8	6	6	9	9	6	6	8	10	7	6	10	11
	ρ	1	0.004	0.004	0.043	0.039	0.005	0.006	0.027	0.045	0.006	0.006	0.056	0.063	0.006	0.006	0.050	0.070	0.007	0.009	0.076	0.086
		2	0.008	0.005	0.072	0.068	0.005	0.005	0.048	0.064	0.010	0.006	0.099	0.100	0.008	0.008	0.092	0.111	0.012	0.012	0.137	0.141
		3	0.015	0.005	0.084	0.065	0.007	0.006	0.043	0.068	0.016	0.007	0.099	0.107	0.012	0.009	0.103	0.108	0.017	0.014	0.144	0.154
		4	0.021	0.006	0.077	0.065	0.009	0.007	0.037	0.070	0.019	0.010	0.090	0.106	0.018	0.010	0.083	0.107	0.021	0.016	0.115	0.147
		5	0.022	0.007	0.061	0.066	0.009	0.008	0.032	0.066	0.014	0.010	0.083	0.095	0.018	0.010	0.066	0.093	0.024	0.017	0.094	0.128
		6	0.019	0.008	0.056	0.062	0.009	0.008	0.033	0.060	0.013	0.012	0.072	0.085	0.016	0.011	0.057	0.086	0.024	0.018	0.089	0.120
		7	0.022	0.008	0.056	0.057	0.009	0.009	0.032	0.056	0.014	0.013	0.069	0.078	0.019	0.012	0.056	0.083	0.025	0.019	0.087	0.115
		8	0.017	0.009	0.056	0.054	0.007	0.010	0.033	0.055	0.012	0.014	0.071	0.077	0.016	0.013	0.056	0.081	0.025	0.020	0.086	0.113
		9	0.019	0.009	0.056	0.054	0.008	0.009	0.033	0.055	0.013	0.015	0.071	0.076	0.018	0.014	0.056	0.080	0.025	0.021	0.086	0.110
	r [MDoF/s]	1	7.3e-2	6.5e-2	4.8e-2	4.0e-2	1.1e-1	1.1e-1	5.6e-2	7.0e-2	1.6e-1	1.5e-1	6.6e-2	7.7e-2	2.1e-1	1.7e-1	8.6e-2	8.9e-2	2.6e-1	1.8e-1	9.3e-2	8.5e-2
		2	1.7e-1	1.3e-1	1.0e-1	8.5e-2	2.4e-1	2.3e-1	1.2e-1	1.4e-1	2.9e-1	3.1e-1	1.5e-1	1.6e-1	5.0e-1	4.0e-1	1.9e-1	1.7e-1	5.2e-1	3.8e-1	1.9e-1	1.8e-1
		3	4.3e-1	3.2e-1	3.1e-1	2.6e-1	7.6e-1	6.1e-1	4.8e-1	4.2e-1	9.3e-1	8.3e-1	5.1e-1	4.3e-1	1.3e+0	1.1e+0	5.9e-1	5.3e-1	1.7e+0	1.1e+0	6.6e-1	5.3e-1
		4	1.3e+0	1.0e+0	9.2e-1	8.5e-1	2.3e+0	1.7e+0	1.4e+0	1.1e+0	2.8e+0	1.9e+0	1.5e+0	1.2e+0	3.9e+0	2.6e+0	1.8e+0	1.5e+0	5.0e+0	3.2e+0	2.0e+0	1.5e+0
		5	4.5e+0	3.3e+0	3.3e+0	2.5e+0	7.2e+0	5.4e+0	4.3e+0	3.4e+0	8.5e+0	6.0e+0	4.4e+0	3.8e+0	1.1e+1	7.9e+0	6.2e+0	4.6e+0	1.4e+1	9.3e+0	6.6e+0	4.6e+0
		6	8.7e+0	9.6e+0	6.2e+0	6.3e+0	1.3e+1	1.5e+1	7.9e+0	8.2e+0	1.5e+1	1.6e+1	8.9e+0	8.8e+0	2.0e+1	1.9e+1	1.1e+1	1.0e+1	2.4e+1	2.1e+1	1.1e+1	9.6e+0
		7	1.8e+1	2.2e+1	1.1e+1	1.2e+1	2.5e+1	3.0e+1	1.2e+1	1.4e+1	2.4e+1	2.6e+1	1.3e+1	1.3e+1	2.7e+1	2.8e+1	1.4e+1	1.4e+1	2.9e+1	2.8e+1	1.4e+1	1.3e+1
		8	2.2e+1	3.0e+1	1.3e+1	1.4e+1	2.7e+1	3.3e+1	1.3e+1	1.5e+1	2.3e+1	2.4e+1	1.2e+1	1.2e+1	2.0e+1	2.1e+1	1.2e+1	1.2e+1	1.9e+1	1.9e+1	1.1e+1	9.7e+0
		9	1.5e+1	2.2e+1	9.8e+0	1.1e+1	1.8e+1	1.7e+1	1.0e+1	1.1e+1	1.5e+1	1.6e+1	9.7e+0	9.6e+0	1.5e+1	1.5e+1	9.8e+0	9.6e+0	1.6e+1	1.6e+1	9.2e+0	8.5e+0
3D	n_{10}	1	5	5	12	11	5	5	11	11	5	5	13	13	5	5	13	13	5	6	15	15
		2	5	5	14	13	5	5	13	13	6	5	15	15	6	6	15	15	7	7	18	17
		3	6	5	14	12	6	5	12	13	7	6	15	15	6	6	15	15	7	7	17	17
		4	7	5	14	13	6	5	11	12	7	6	14	14	7	6	14	14	8	8	16	16
		5	7	5	13	13	6	6	11	12	7	6	13	14	7	6	13	14	8	8	15	16
	ρ	1	0.005	0.005	0.131	0.121	0.004	0.005	0.108	0.117	0.005	0.005	0.150	0.165	0.006	0.008	0.152	0.165	0.010	0.015	0.199	0.213
		2	0.008	0.005	0.175	0.147	0.006	0.005	0.149	0.158	0.015	0.009	0.214	0.198	0.011	0.010	0.207	0.207	0.023	0.024	0.265	0.254
		3	0.019	0.006	0.180	0.143	0.011	0.008	0.128	0.150	0.023	0.012	0.195	0.200	0.017	0.016	0.200	0.200	0.031	0.031	0.250	0.247
		4	0.026	0.008	0.182	0.151	0.015	0.010	0.117	0.146	0.025	0.017	0.174	0.189	0.024	0.020	0.178	0.191	0.039	0.038	0.220	0.231
		5	0.033	0.010	0.165	0.148	0.012	0.011	0.106	0.138	0.024	0.019	0.162	0.177	0.028	0.021	0.163	0.179	0.043	0.040	0.199	0.218
	r [MDoF/s]	1	3.7e-1	3.1e-1	1.2e-1	1.2e-1	5.7e-1	5.6e-1	1.8e-1	1.9e-1	8.1e-1	6.7e-1	2.1e-1	1.7e-1	9.5e-1	7.2e-1	2.2e-1	1.7e-1	1.1e+0	5.9e-1	2.1e-1	1.5e-1
		2	1.3e+0	9.9e-1	5.8e-1	5.2e-1	2.2e+0	2.1e+0	8.0e-1	7.8e-1	2.8e+0	2.8e+0	9.0e-1	7.5e-1	3.8e+0	2.6e+0	1.0e+0	7.6e-1	4.8e+0	2.9e+0	9.7e-1	6.7e-1
		3	3.6e+0	4.7e+0	2.4e+0	2.5e+0	5.7e+0	8.4e+0	3.1e+0	3.2e+0	9.2e+0	9.0e+0	3.4e+0	3.0e+0	1.2e+1	1.0e+1	3.5e+0	2.9e+0	1.2e+1	9.0e+0	3.3e+0	2.4e+0
		4	1.1e+1	1.6e+1	6.0e+0	6.6e+0	1.5e+1	2.0e+1	7.0e+0	6.8e+0	1.5e+1	1.7e+1	6.2e+0	5.5e+0	1.6e+1	1.5e+1	5.4e+0	4.7e+0	1.2e+1	1.1e+1	4.4e+0	4.0e+0
		5	8.9e+0	1.4e+1	5.1e+0	6.1e+0	1.1e+1	1.2e+1	5.4e+0	5.5e+0	1.1e+1	1.1e+1	4.6e+0	4.7e+0	9.3e+0	1.1e+1	4.3e+0	4.5e+0	9.8e+0	9.5e+0	4.1e+0	3.7e+0

Table 12: Convergence table for $8 \leq k \leq 12$

			$k=8$				$k=9$				$k=10$				$k=11$				$k=12$			
			CG		DG		CG		DG		CG		DG		CG		DG		CG		DG	
		#refs	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG	pMG	hMG
2D	n_{10}	1	5	5	9	10	6	6	10	11	5	6	10	12	6	7	11	13	6	7	11	13
		2	5	6	12	13	6	7	14	14	6	7	14	15	7	8	15	16	7	8	16	16
		3	5	6	11	13	6	7	13	15	6	7	13	15	7	8	14	16	7	8	14	16
		4	5	6	10	12	6	7	11	14	6	7	11	14	7	8	13	15	7	9	12	15
		5	5	6	10	12	6	7	11	14	6	8	11	14	7	9	12	15	7	9	12	15
		6	5	6	9	12	6	8	11	13	6	8	10	13	7	9	12	15	7	9	12	15
		7	5	6	9	12	6	8	11	13	7	8	10	13	7	9	12	14	7	9	12	14
		8	5	7	9	11	6	8	11	13	6	8	10	13	7	9	12	14	7	10	12	14
	ρ	1	0.006	0.010	0.065	0.093	0.010	0.015	0.086	0.116	0.009	0.018	0.091	0.133	0.013	0.027	0.123	0.153	0.016	0.027	0.116	0.158
		2	0.010	0.015	0.143	0.158	0.017	0.023	0.173	0.180	0.015	0.028	0.181	0.195	0.029	0.040	0.211	0.223	0.030	0.047	0.219	0.225
		3	0.010	0.015	0.115	0.158	0.018	0.028	0.153	0.196	0.020	0.032	0.148	0.197	0.032	0.052	0.175	0.232	0.028	0.055	0.174	0.237
		4	0.010	0.018	0.086	0.147	0.020	0.034	0.120	0.181	0.021	0.037	0.118	0.183	0.035	0.056	0.151	0.215	0.036	0.059	0.145	0.214
		5	0.010	0.018	0.078	0.140	0.019	0.036	0.120	0.173	0.021	0.038	0.109	0.175	0.033	0.062	0.141	0.205	0.034	0.065	0.140	0.204
		6	0.009	0.019	0.069	0.129	0.018	0.038	0.116	0.166	0.021	0.040	0.099	0.166	0.033	0.066	0.128	0.198	0.030	0.070	0.128	0.194
		7	0.009	0.020	0.069	0.124	0.018	0.040	0.116	0.154	0.022	0.042	0.098	0.153	0.034	0.069	0.128	0.193	0.031	0.075	0.129	0.184
		8	0.008	0.022	0.069	0.121	0.016	0.041	0.119	0.150	0.021	0.045	0.099	0.148	0.033	0.071	0.137	0.184	0.029	0.078	0.139	0.180
	r [MDoF/s]	1	2.2e-1	1.9e-1	1.1e-1	7.7e-2	2.5e-1	1.7e-1	1.1e-1	8.5e-2	3.6e-1	1.9e-1	1.2e-1	8.1e-2	3.3e-1	1.7e-1	1.3e-1	8.2e-2	3.7e-1	1.7e-1	1.3e-1	8.3e-2
		2	5.3e-1	4.1e-1	1.9e-1	1.8e-1	6.2e-1	4.0e-1	2.1e-1	1.9e-1	7.4e-1	4.4e-1	2.4e-1	1.9e-1	7.2e-1	4.6e-1	2.3e-1	1.8e-1	8.2e-1	4.2e-1	2.5e-1	1.8e-1
		3	1.5e+0	1.3e+0	7.3e-1	6.1e-1	2.1e+0	1.2e+0	7.9e-1	5.6e-1	2.5e+0	1.3e+0	9.1e-1	5.8e-1	2.4e+0	1.2e+0	9.5e-1	5.8e-1	2.8e+0	1.3e+0	1.0e+0	5.9e-1
		4	6.0e+0	3.8e+0	2.3e+0	1.7e+0	6.0e+0	3.5e+0	2.4e+0	1.7e+0	7.1e+0	3.9e+0	2.7e+0	1.8e+0	6.9e+0	3.7e+0	2.7e+0	1.7e+0	7.7e+0	3.9e+0	2.9e+0	1.7e+0
		5	1.4e+1	1.1e+1	6.3e+0	4.5e+0	1.7e+1	9.9e+0	6.5e+0	4.4e+0	1.9e+1	1.1e+1	7.0e+0	4.6e+0	1.8e+1	9.9e+0	7.1e+0	4.4e+0	1.9e+1	1.0e+1	7.2e+0	4.4e+0
		6	2.9e+1	2.3e+1	1.2e+1	9.9e+0	2.6e+1	2.0e+1	1.0e+1	8.3e+0	2.7e+1	2.0e+1	1.2e+1	8.2e+0	2.3e+1	1.8e+1	1.1e+1	7.6e+0	2.4e+1	1.7e+1	1.1e+1	7.3e+0
		7	3.3e+1	2.8e+1	1.4e+1	1.2e+1	2.7e+1	2.2e+1	1.2e+1	1.1e+1	2.6e+1	2.1e+1	1.2e+1	1.0e+1	2.0e+1	1.6e+1	1.1e+1	7.8e+0	1.9e+1	1.3e+1	10.0e+0	8.0e+0
		8	2.2e+1	1.8e+1	1.1e+1	9.1e+0	1.8e+1	1.4e+1	8.6e+0	8.1e+0	1.7e+1	1.4e+1	9.3e+0	7.7e+0	1.4e+1	1.2e+1	7.6e+0	6.9e+0	1.4e+1	1.0e+1	7.3e+0	6.8e+0
3D	n_{10}	1	5	6	14	16	6	7	17	18	6	7	17	18	7	8	19	19	7	8	19	19
		2	7	7	18	18	8	8	20	20	8	9	21	21	9	10	23	22	9	10	23	22
		3	7	8	15	18	8	9	18	19	8	9	18	19	10	10	19	21	9	11	19	21
		4	7	8	14	17	8	9	16	18	8	10	17	18	10	11	18	21	10	11	19	21
	ρ	1	0.007	0.015	0.191	0.225	0.017	0.023	0.242	0.267	0.018	0.027	0.246	0.260	0.031	0.044	0.287	0.297	0.033	0.047	0.281	0.286
		2	0.024	0.032	0.272	0.274	0.046	0.056	0.314	0.308	0.043	0.062	0.319	0.320	0.073	0.089	0.359	0.342	0.077	0.095	0.358	0.350
		3	0.028	0.038	0.209	0.259	0.055	0.061	0.264	0.290	0.052	0.071	0.265	0.291	0.080	0.098	0.291	0.331	0.074	0.107	0.295	0.328
		4	0.029	0.048	0.185	0.241	0.052	0.077	0.237	0.272	0.052	0.084	0.248	0.278	0.079	0.116	0.275	0.323	0.079	0.122	0.279	0.321
	r [MDoF/s]	1	1.1e+0	5.7e-1	1.9e-1	1.3e-1	9.3e-1	5.6e-1	1.7e-1	1.1e-1	9.3e-1	4.4e-1	1.5e-1	9.8e-2	8.4e-1	4.2e-1	1.5e-1	8.6e-2	8.6e-1	3.6e-1	1.4e-1	8.5e-2
		2	4.9e+0	2.4e+0	9.2e-1	5.9e-1	4.7e+0	2.1e+0	9.0e-1	5.2e-1	5.0e+0	2.0e+0	7.8e-1	4.4e-1	4.2e+0	1.7e+0	8.1e-1	4.2e-1	4.4e+0	1.7e+0	7.2e-1	3.8e-1
		3	1.1e+1	8.5e+0	3.5e+0	2.3e+0	1.0e+1	7.0e+0	3.1e+0	2.0e+0	1.0e+1	6.7e+0	2.7e+0	1.8e+0	9.4e+0	5.7e+0	2.5e+0	1.5e+0	8.4e+0	4.7e+0	2.1e+0	1.4e+0
		4	1.1e+1	9.0e+0	4.5e+0	3.6e+0	9.4e+0	7.5e+0	3.7e+0	2.9e+0	9.1e+0	7.2e+0	3.1e+0	2.6e+0	7.9e+0	6.2e+0	2.9e+0	2.2e+0	7.6e+0	6.0e+0	2.6e+0	2.1e+0

- In all cases, CG outperforms DG. The iteration count of CG is up to 50% less than the one of DG. Since the throughput of the matrix-free matrix-vector multiplication `vmult` for CG and DG on non-Cartesian mesh is comparable, a reduction of the iteration count by up to 50% means that the throughput of CG is up to $2\times$ higher than the throughput of DG. This, in turn, leads – if it is considered that CG has fewer degrees of freedom – to CG being up to more than $2\times$ faster in solving an equivalent linear system of equations.
- Overall, p-MG seems to work better for CG than for DG. Increasing the order k from 3 to 12 leads in the case of CG to an increase of iterations by 1-3, but in the case of DG by 3-9.

5.4 Node-level performance

This subsection analyzes the node-level performance of the presented hybrid multigrid solver for CG/DG and 2D/3D. The amount of time spent on each multigrid level and spent in each operation is looked at in detail. As an example, the configuration $k = 6$, $l = 10$ (2D) and $l = 5$ (3D) with p-multigrid is considered. In the case of CG, there are three multigrid levels for the analyzed configuration: $k = 1,\ 3,\ 6$. In the case of DG, there is an additional coarse level with continuous elements.

In Figure 9, two plots are shown for each spatial discretization-dimension configuration. The left pie diagram indicates the fraction of time of each multigrid level in the overall algorithm. The right bar diagram shows each level on its own and the fraction of time spent in each function call on that level. The function calls are: pre-smoothing (incl. an additional matrix-vector multiplication to update the defect), restriction, coarse-grid correction, prolongation, and post-smoothing. Furthermore, the expected value of the fraction of time spent on the coarse-grid correction of each level is indicated with a horizontal red line: its value is according to

$$\frac{\text{cost of correction}}{\text{inclusive cost of level}} = \frac{\sum_{i=1}^{\infty} 0.5^{d\cdot i}}{\sum_{i=0}^{\infty} 0.5^{d\cdot i}} = \frac{\sum_{i=0}^{\infty} 0.5^{d\cdot i} - 1}{\sum_{i=0}^{\infty} 0.5^{d\cdot i}} = 0.5^{d}, \tag{58}$$

25% for 2D and 12.5% for 3D.

Generally, the following observations can be made, based on Figure 9. Only about 15% of the time is spent in the conjugate gradient solver, the remainder is spent in the multigrid preconditioner. The most expensive operations on each multigrid level are the pre- and post-smoothing steps. The costs of transfer operations are negligible. A significant fraction of time is spent on the coarse-grid correction of each level. This fraction of time matches well with the aforementioned expectations for CG. For DG, however, the coarse-grid corrections are more expensive. A possible explanation for this observation can be derived from Figure 10, which shows the time to solution for increasing problem size (expressed by number of degrees of freedom) and the throughput for solving the pressure Poisson problem. It clearly shows a strong increase in throughput in the case of CG with decreasing problem size as the loaded data fits better into the cache. This effect is not that pronounced in the case of DG. For 3D DG, it can be also observed that the fraction of time spent on solving the matrix-based coarse-grid problem (41%) is significantly higher than the expected value. This might be due to the fact that the data structures of the calling matrix-free level fit into the cache (making the smoothing steps relatively cheap), whereas the AMG data structures do not.

Figure 11 generalizes the investigation of the costs of the coarse-grid problems, by taking a look at a large variety of degree k- and refinement l-combinations for DG. The general observations are that the coarse-grid problems tend to be more expensive than the expectations and the coarse-grid problems become cheaper in proportion to the costs of the overall algorithm with increasing problem size.

The following general possibilities to accelerate the algorithm are the result of the previous discussion. The references in brackets indicate which topic will be discussed in which subsection:

- reducing the costs of the coarse-grid problem, *e.g.* by using a simpler coarse-grid preconditioner (see Subsection 5.5) or by replacing the AMG-preconditioned coarse-grid solver PCG with a single AMG V-cycle (see Subsection 5.6),
- reducing the number of multigrid levels, *e.g.* by increasing the jumps between p-levels in the case of p-MG (see Subsection 5.7) or by solving the coarsest problem using discontinuous elements in the case of DG, leading to one level less (see Subsection 5.6), or
- reducing the number of V-cycles, *e.g.* by selecting stronger (but potentially more expensive) smoothers. This topic is outside the scope of this Master's thesis. We will only consider the influence of the aforementioned design decisions on the total number of cycles.

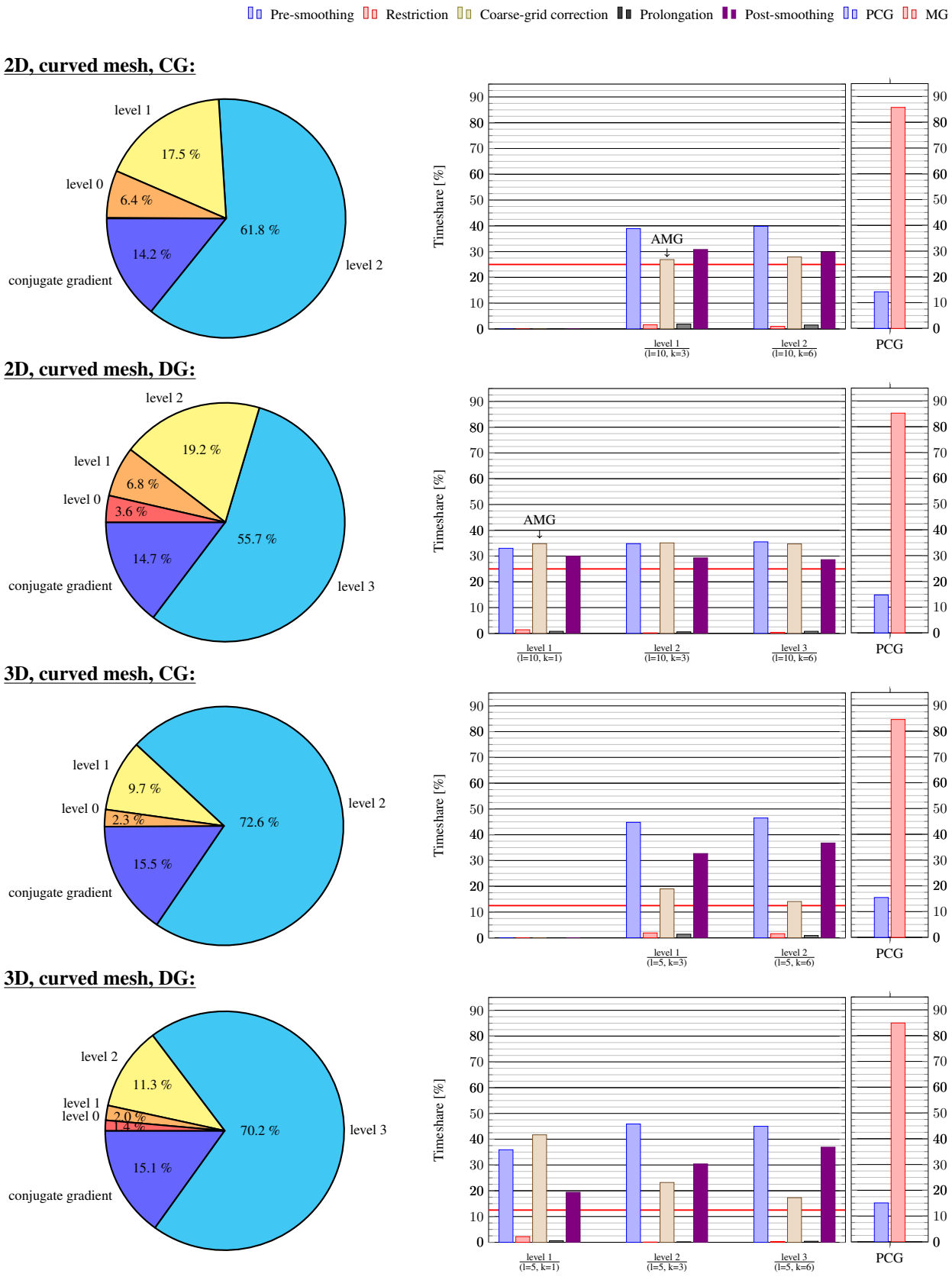

Figure 9: Profiling of conjugate gradient solver preconditioned by p-MG for $k = 6$ and $l = 10/5$ for 2D/3D

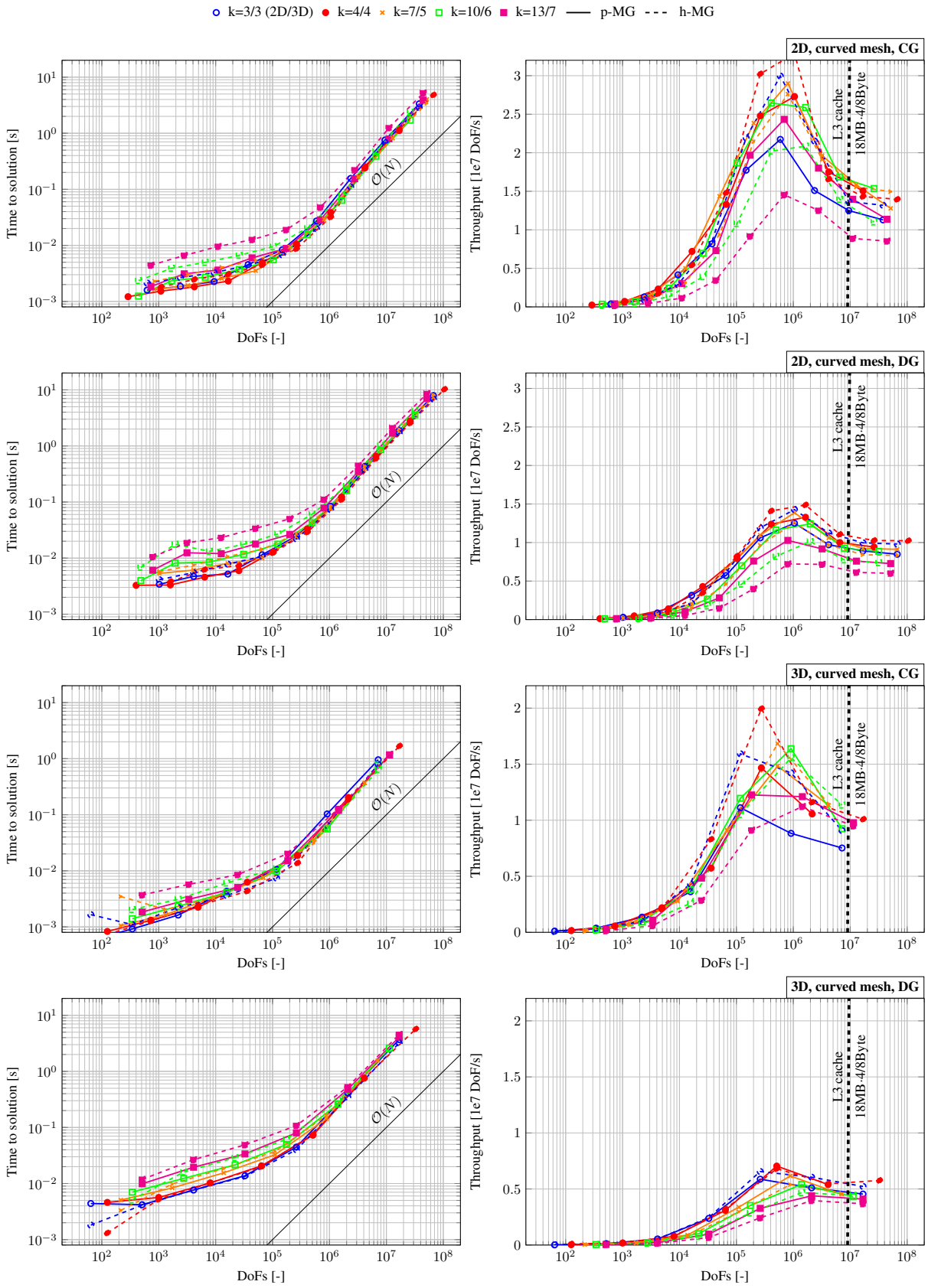

Figure 10: Time to solution and throughput for different degrees k and refinements l on a single node

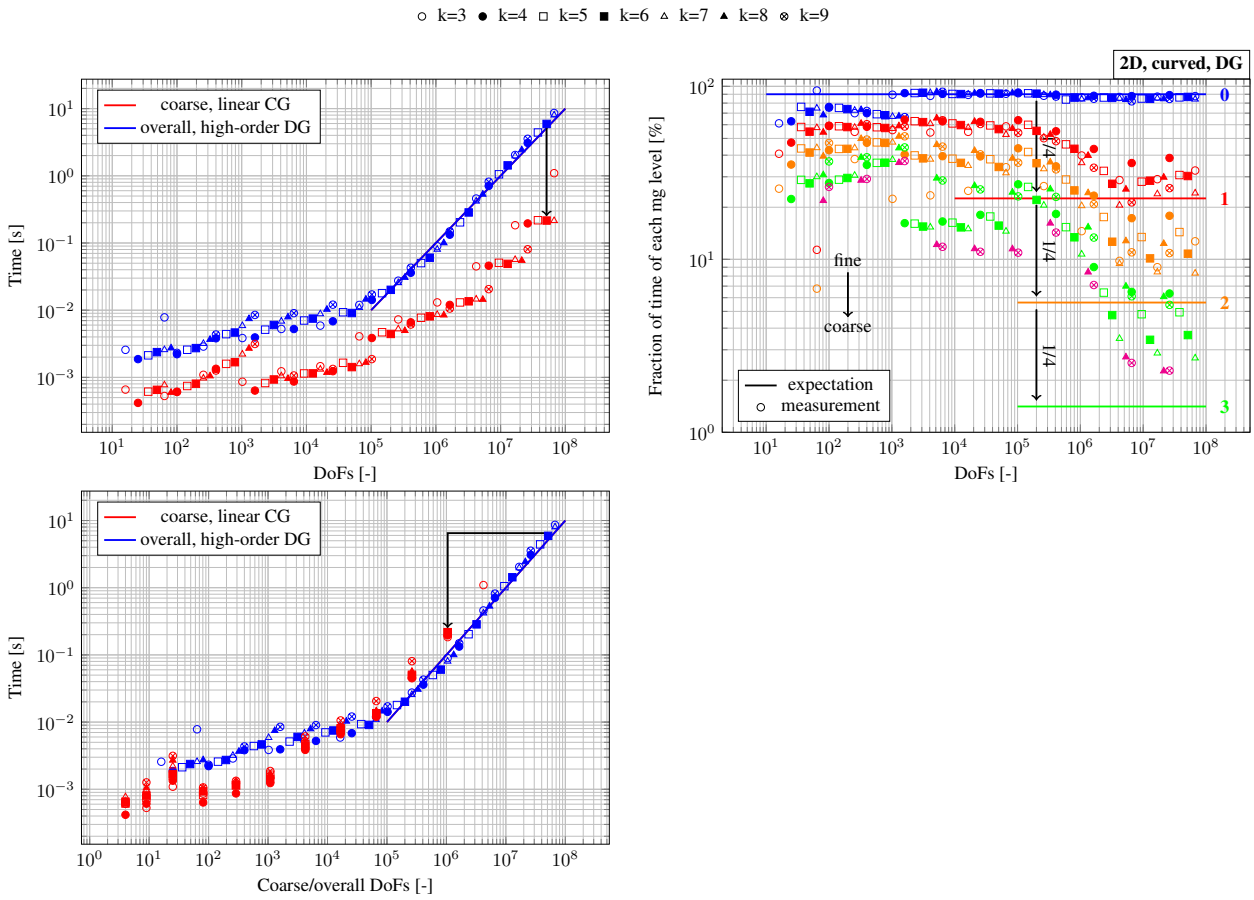

Figure 11: Analysis of the multigrid levels for 2D DG with p-MG for different k and l. Left: time to solution for solving the fine-grid problem and time spent on the corresponding coarse-grid problem (indicated exemplarily by an arrow for one configuration). Right: fraction of time of each level. The levels are enumerated increasingly from the fine to the coarsest level, starting with 0.

5.5 Coarse-grid preconditioner: AMG vs. simple iterative solvers (point Jacobi, Chebyshev solver)

The h-multigrid solver described in [59] uses PCG preconditioned by point Jacobi as a coarse-grid solver. It works well if it is possible to create enough multigrid levels such that the coarse-grid problem can be reduced as much as possible, ideally, to 1 macro cell. Generally, however, this is not the case. For instance, in the case of p-multigrid the problem size can be only decreased by the factor of $\left(\frac{k_f}{k_c}\right)^d$ for CG or by $\left(\frac{k_f+1}{k_c+1}\right)^d$ for DG, rendering $N_{DoFs} = \mathcal{O}\left(2^{d \cdot l}\right)$ unknowns on the coarse level. Increasing the refinement level l also translates directly into an increase in unknowns on the coarsest level and leads to a slowdown of the convergence of PCG applied on the coarsest level. The reason for this slowdown is the increase in the number of iterations of PCG on the coarsest level to meet the same tolerance criterion with $N_{iter} \sim \sqrt{\kappa} \sim \sqrt{1/h^2} = 1/h \sim \sqrt[d]{N_{DoFs}}$, leading to an increase of time to solution with $\mathcal{O}(N_{iter} \cdot N_{DoFs}) = \mathcal{O}(N_{DoFs}^{1.5})$ for 2D. The disproportional increase in the costs of the coarse-grid solver on the coarsest level leads to a preponderance of the coarse-grid solver in the whole multigrid algorithm and, consequently, to the preponderance of the multigrid preconditioner in the whole solution procedure, as shown in Figure 12.

The same observations can be made if the point Jacobi coarse-grid solver is replaced by a Chebyshev solver (see Subsection 2.3.2): the time spent on solving the coarse-grid problem also increases with $\mathcal{O}(N_{DoFs}^{1.5})$, leading to the preponderance of the coarse-grid solver in the whole solution process. The multigrid algorithm, which is solved on the coarse grid by Chebyshev iterations, is faster for small problem sizes (by $\leq$42%) and slower for large problem sizes (by $\leq$45%) than if it were solved by the point Jacobi solver.

By contrast, the iteration numbers of AMG are only weakly dependent on problem size $\mathcal{O}(N_{DoFs})$, *i.e.* on the refinement level. If AMG is used as preconditioner of the coarse-grid solver PCG, the fraction of time of the coarse-grid

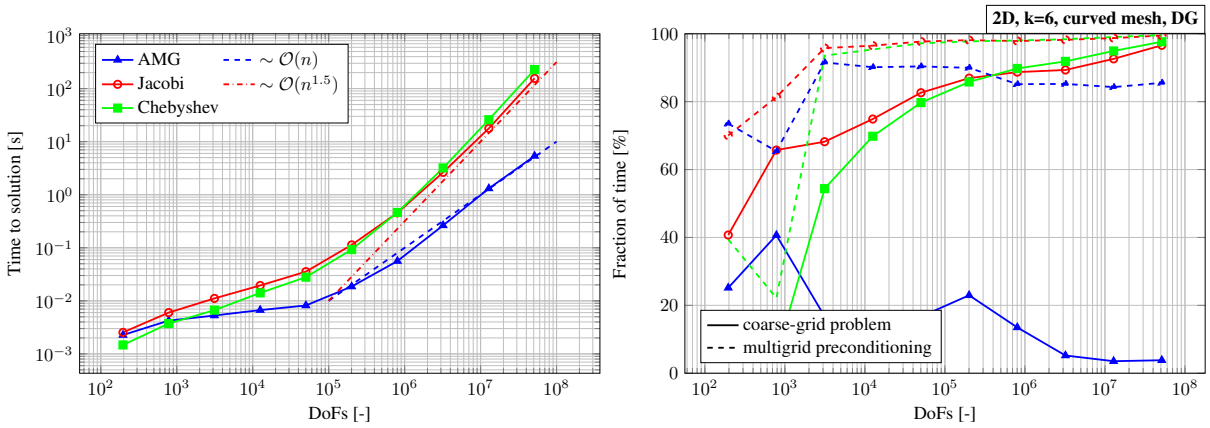

Figure 12: Comparison of different preconditioners (AMG, point Jacobi, Chebyshev solver) for the coarse-grid solver PCG

problem stays approximately the same as the problem size increases, as does the fraction of time of the whole multigrid preconditioner (see Figure 12).

The observation that AMG iterations are independent of problem size underlies the benefit of using an optimal algorithm for a high number of unknowns also on the coarse level.

5.6 Algebraic coarse-grid solver

We use PCG preconditioned by AMG by default as a coarse-grid solver in the developed hybrid multigrid solver when the polynomial degree has been reduced to one via a sequence of p-coarsening steps. For DG, we use in addition the auxiliary solution from the coarse-grid problem solved for continuous elements (CG). In the following analysis, the default coarse-grid solver should be referred to as `PCG-MG (continuous)`, which we will compare with two alternative coarse-grid solver versions:

1. `MG (continuous)`, which replaces PCG with a single AMG V-cycle step, and
2. `PCG-MG (discontinuous)`, which solves the coarse-grid problem with PCG for DG discretization, reducing the number of levels by one.

In order to judge the performance of one method against the other, we solve the 2D Poisson problem discretized with DG for a large amount of combinations of degrees $3 \leq k \leq 9$ and refinements l such that the number of degrees of freedom is $< 8162^2$. The results are presented in Figure 13. The default configuration shows the best performance overall, for which the reason is twofold, namely a moderate number of iterations with only a few coarse-grid iterations. By contrast, `MG (continuous)` shows a significant increase in V-cycles. Consequently, the expensive operation applications on the fine levels have to be performed more often, leading to a deteriorated performance despite having the cheapest coarse-grid problem. `PCG-MG (discontinuous)` needs as many iterations as `PCG-MG (continuous)` does (in isolated cases it needs less). However, it is the slowest version due to the slow convergence of the library `ML` for DG (up to 9 coarse-grid iterations are needed per iteration for a significantly larger system of equations). As a consequence, the coarse-grid problem takes up to 80% of the overall time. This observation underscores the necessity of AMG implementations specialized for DG discretization. Such a solver becomes crucial once convection-dominated problems are to be solved, in which a switch to continuous space with the currently implemented c-transfer operator does not provide a useful auxiliary-space correction.

5.7 P-sequences

Three different p-sequence approaches were introduced in Subsection 2.3.4. Sorted according to the resulting number of multigrid levels, they are as follows:

$$\textcircled{1}\ k_{i-1} = 1, \qquad \textcircled{2}\ k_{i-1} = \max\left(1,\ \lfloor k_i/2 \rfloor\right), \qquad \textcircled{3}\ k_{i-1} = \max\left(1,\ k_i - 1\right). \tag{59}$$

By default, we use approach ② because of its similarity to the coarsening properties of h-multigrid. The choice of a p-sequence strategy has a direct influence on the number of multigrid levels and consequently on the cost of the

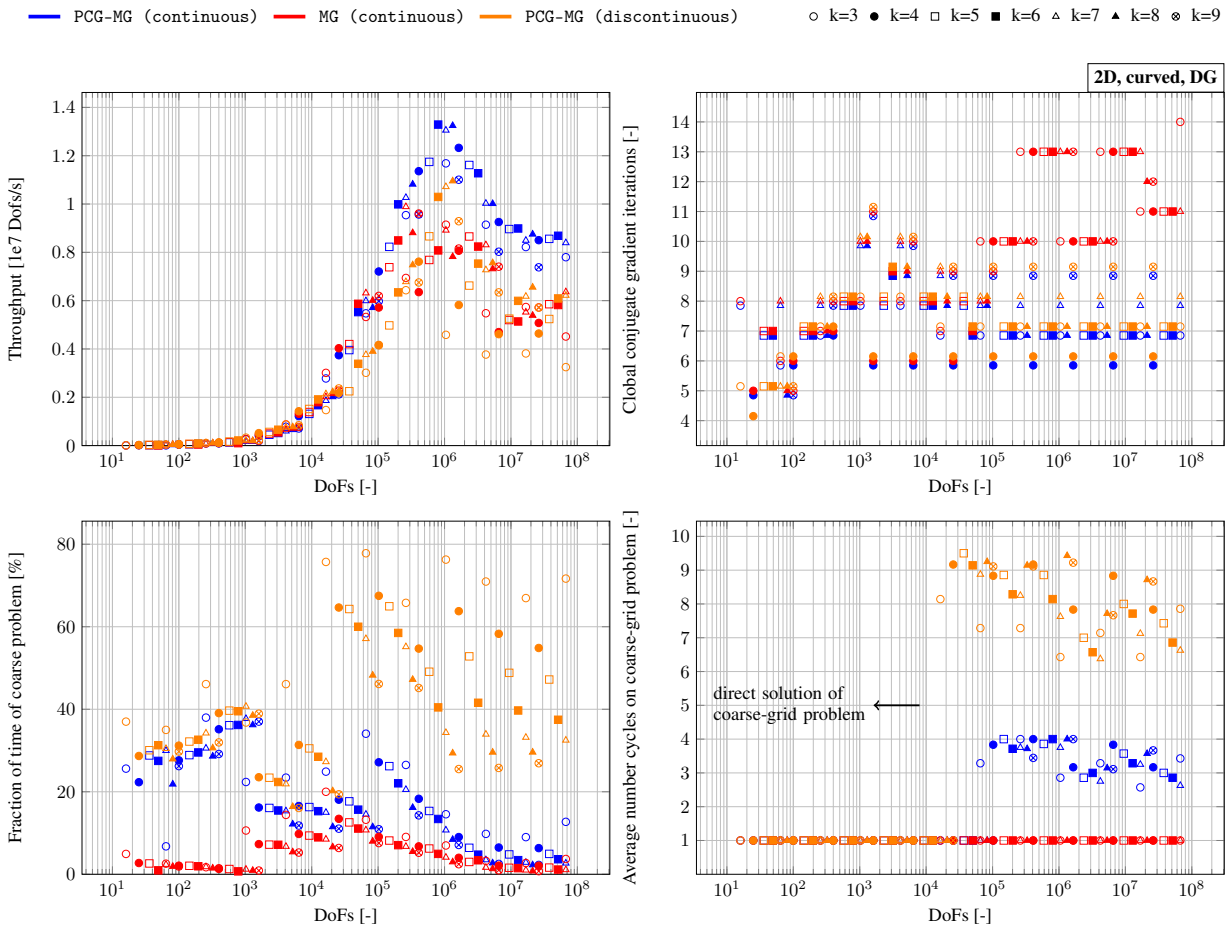

Figure 13: Comparison of three versions of the algebraic coarse-grid solver: PCG with fixed relative tolerance and CG coarse-grid discretization (base case: `PCG-MG (continuous)`), one multigrid preconditioning step with CG coarse-grid discretization (`MG (continuous)`), and PCG with DG coarse-grid discretization (`PCG-MG (discontinuous)`).

coarse-grid problem as well as on the quality of the preconditioning, which - for its part - influences the total number of iterations. As a consequence, it has a direct influence on overall performance.

Figure 14 presents the order dependency of the iteration number of the three approaches for a problem of fixed size. While approach ③ shows an order independence, the number of iterations of approach ① increases linearly with increasing order. The latter was also observed by Bastian et al. [13] and Siefert et al. [90]. Approach ②, which halves the orders of the fine levels during coarsening, shows only a weak dependency on the order and has only a few more iterations than approach ③.

Until now, we have considered the number of iterations. Let us now take a look at the cost of a single preconditioning step. The cost of approach ② has been discussed already in Subsection 5.4. Prompted by the statistical evaluation of the cost of each level in Figure 9 for approach ②, the same plots are created also for approach ① and ③ in Figure 15. In the given example for $k = 6$, one preconditioning step is only slightly cheaper in approach ① than in approach ②. The reason for this is that the number of levels is decreased only by one in approach ①. For approach ③, a well-visible significant increase in the cost of the coarse-grid corrections and thereby the increase of the overall cost of preconditioning can be observed where more than 90% of the execution time is spent in the preconditioner in this case as opposed to the generally observed 85%.

The opposing trends in iteration numbers and in the cost of preconditioning in the case of approach ① and ③ produce similar overall performance trends for both coarsening strategies. As shown in Figure 14, the throughput decreases with increasing order in the case of both approaches with approach ③ being slightly faster. A good compromise between

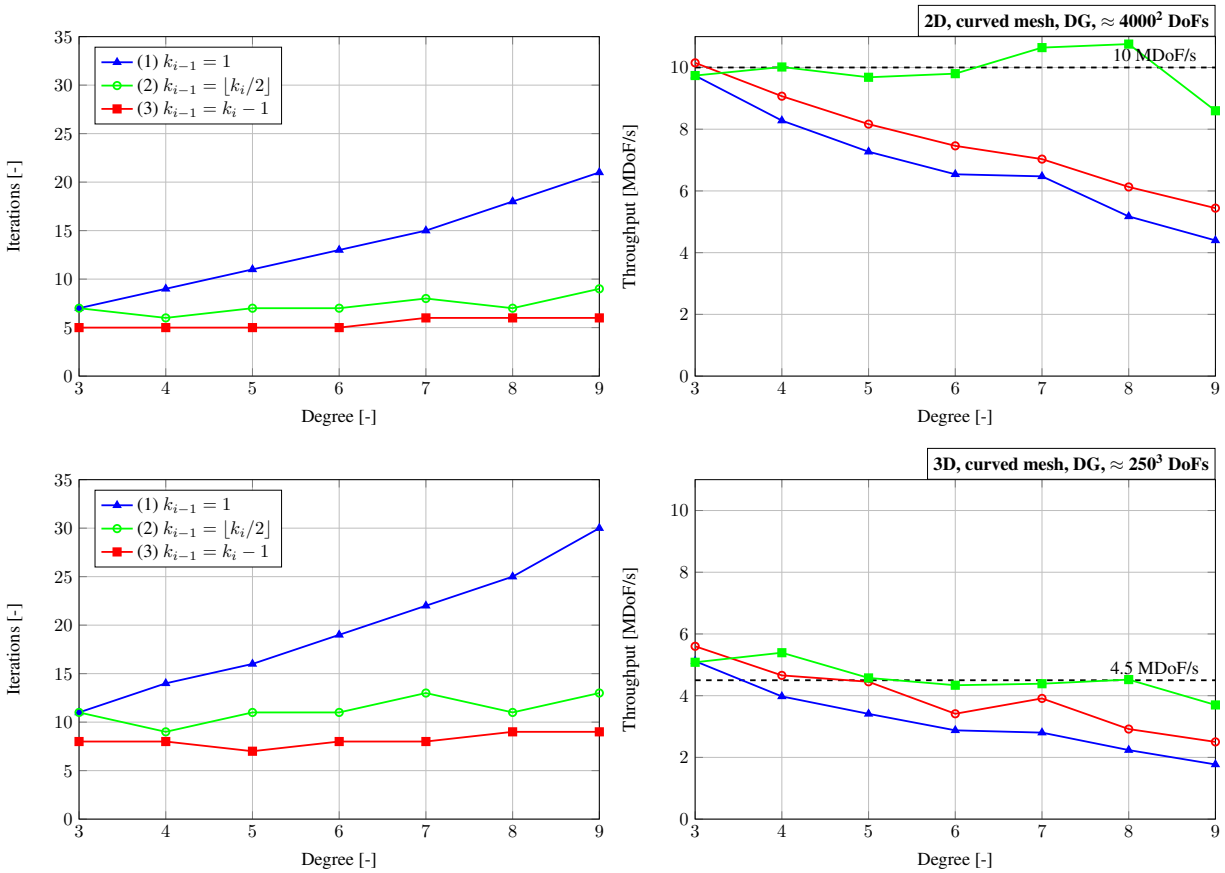

Figure 14: Different p-sequences

approach ① and ③, is approach ②, which needs a few more iterations than approach ③ and whose iterations are negligible more expensive than the ones by approach ①. Approach ② exhibits an order-independent throughput for the considered order range, maintaining 10MDoF/s for 2D and 4.5MDoF/s for 3D over the whole considered degree range. These values are about twice as high as for approach ① with $k = 9$. The speedup corresponds directly to the reduced number of iterations by about 50%. We believe that the integration of approach ② would be a remedy against the strong order dependency of the multigrid solver presented by Bastian et al. [12, 13] and Siefert et al. [90].

Besides the performance, it is essential to also consider memory consumption by the three analyzed p-sequence approaches. The maximal value regarding memory consumption (see also Equation (48) in Subsection 3.3) can be estimated for all three cases using the following formulas:

$$\mathcal{M}_1 = 1 + \frac{2^d}{(k+1)^d} \ll 2 = \mathcal{O}(1), \qquad \mathcal{M}_2 \leq \frac{1}{1-0.5^d} \leq 2 = \mathcal{O}(1), \qquad \mathcal{M}_3 = \frac{\sum_{i=1}^{k}(i+1)^d}{(k+1)^d} \leq k = \mathcal{O}(k). \tag{60}$$

While approach ① and ② are independent of degree k, the memory consumption of approach ③ is bounded by k, making it unsuitable for very high orders and resulting in this approach running out of memory earlier.

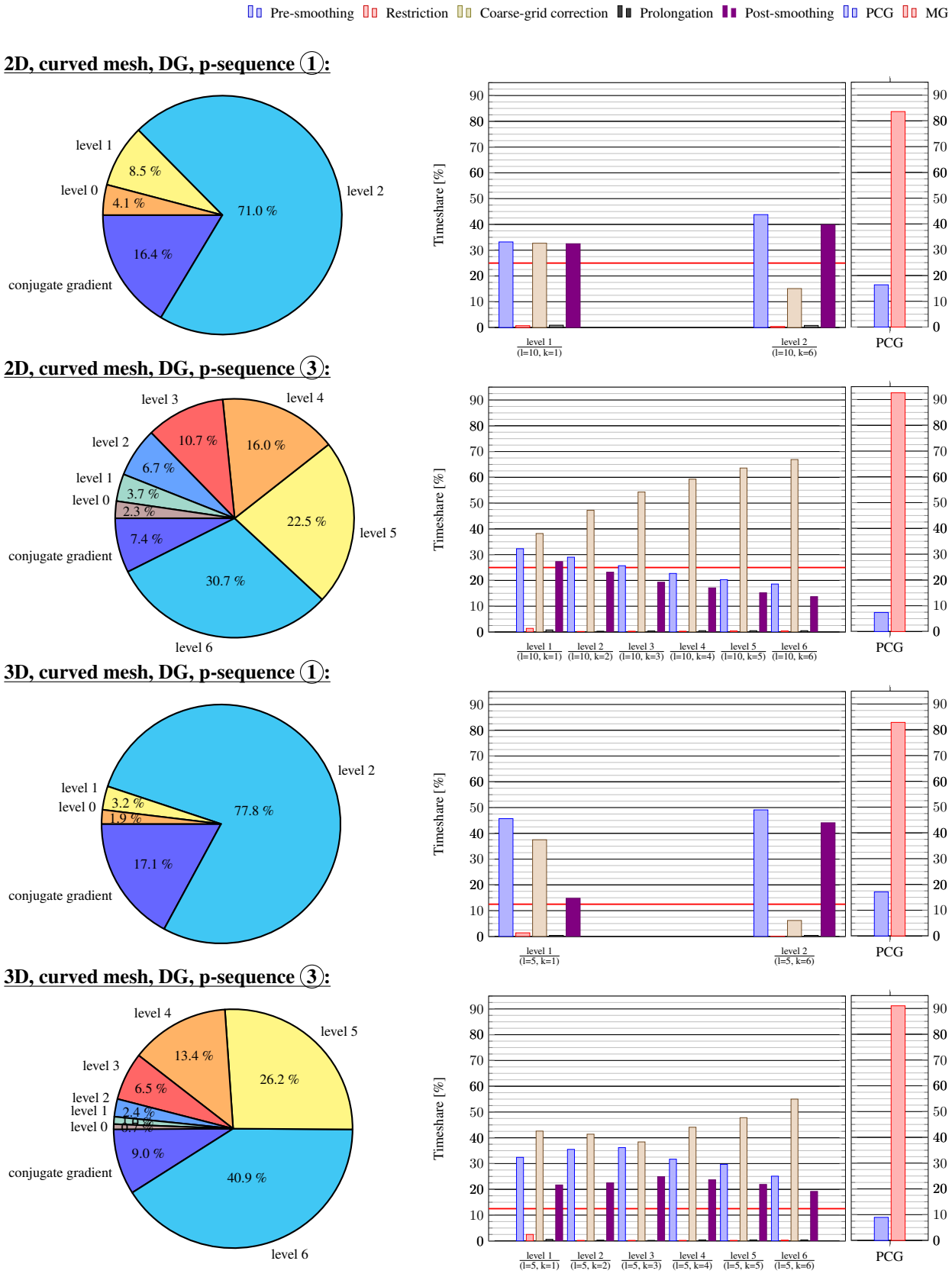

Figure 15: Profiling of the preconditioned conjugate gradient solver for $k = 6$ and $l = 10/5$ for 2D/3D DG in the case of alternative p-sequence strategies (①: $k_{i-1} = 1$; ③: $k_{i-1} = k_i - 1$)

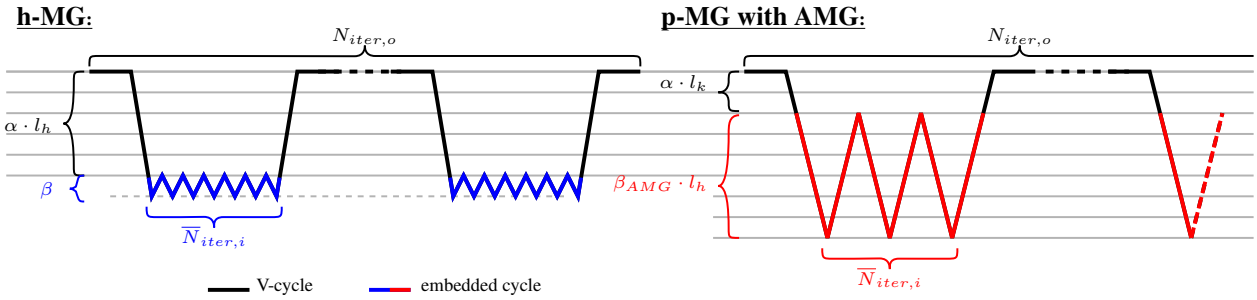

Figure 16: Visualization of the strong-scaling limit model for h-MG and p-MG with AMG

5.8 Strong scaling

Figures 17-24 present strong-scaling results for 2D/3D, h-MG/p-MG, and CG/DG. For this purpose, problems of fixed size $k = 6$, $l = 7, 8, 9, 10, 11, 12$ (2D) and $k = 4$, $l = 5, 6, 7, 8$ (3D) were solved on an increasing number of compute nodes. A maximum of 60 nodes with a total of 1680 CPUs were used. For each configuration, time to solution, accumulated throughput, throughput per CPU, speedup $S_n = T_{1 \text{ node}}/T_{n \text{ nodes}}$, parallel efficiency $E_n = S_n/n$, and fraction of time of the coarse-grid problem are shown for $1 \leq n \leq 60$ nodes. To be able to make a fair comparison of the fraction of time of the coarse-grid problem within the context of h-MG and p-MG, measured timings of multiple coarse levels have been summed up to an effective coarse value such that the number of 'fine' levels equals the number of p-MG 'fine' levels[21].

We observe ideal strong scaling of both the p- and h-multigrid solvers (even superlinear speedup is observed for simulations with a high refinement level $l \geq 9$) until a low threshold where communication latency becomes dominant. Until that threshold, timings of p-MG and h-MG are comparable. However, thenceforward they start to differ significantly due to differing costs for the coarse-grid problem in the case of p-MG with AMG and of h-MG (90% vs. 75% of the overall time is spent on the coarse-grid problem). A maximum throughput of 970MDoF/s for 2D and 230MDoF/s for 3D was reached in the case of CG with h-MG. With p-MG, maximum throughput of 810GDoF/s for 2D and 270GDoF/s for 3D was reached for CG.

Modeling the strong-scaling limit: Based on the gathered data, it is possible to construct a performance model for the strong-scaling limit. In doing so, it helps to subdivide the multigrid algorithm into two parts (V-cycles and embedded cycles on the coarse level (for PCG and AMG)) according to Figure 16. A performance model for such a system might look like this:

$$t = N_{iter,o} \cdot l_o \cdot \alpha + N_{iter,i} \cdot \beta = N_{iter,o} \cdot (l_o \cdot \alpha + \overline{N}_{iter,i} \cdot \beta), \tag{61a}$$

where l_o is the number of matrix-free multigrid levels, α is the cost of one matrix-free multigrid level, and β is the cost of one embedded cycle. The symbols $N_{iter,o}$ and $N_{iter,i}$ denote the number of outer and inner cycles, *i.e.* V-cycles and embedded cycles. $\overline{N}_{iter,i}$ is the average number of embedded cycles per V-cycle. Measurements (see Figure 25) have shown that the time spent on one matrix-free multigrid level is approximately $\alpha = 0.4$ms and is the same on every level since it is not determined by the amount of work to be performed but rather by the number of the communication instances, which is the same on every level except on the coarsest level[22]. In the following, the definitions of the model parameters β and l_o are refined, and actual values for β and $\overline{N}_{iter,i}$ are presented for p-MG and h-MG, collected representatively from 2D CG simulations (see Figure 17 and 18).

For h-multigrid, post-processing of the measurements results in:

$$l_o = l_h, \qquad \overline{N}_{iter,i} \cdot \beta = 0.0016. \tag{61b}$$

As also shown in Figure 25, the minimal time to solution increases linearly with an increasing number of refinements l_h. The time to solution $\overline{N}_{iter,i} \cdot \beta$ on the coarsest grid (1 macro cell) is independent of the number of refinements. Figure 26 visualizes the fraction of time of each level in a pie plot for 1 node and 60 nodes. It clearly shows the different behavior of h-multigrid in the strong-scaling limit: in contrast to 1 node, where the cost of each level is $\approx 0.5^d$ of the cost of the parent level, the cost of each level is approximately the same in the case of 60 nodes. The cost of the coarsest level is as expensive as in the case of 1 node since the processing of a single element is not parallelized.

[21] As an example, let us consider the multigrid levels for $l = 6$ and $k = 6$ for CG for p-MG and h-MG:

p-MG: $\underbrace{(6,6) \rightarrow (6,3)}_{\text{'fine'}} \rightarrow \underbrace{(6,1)}_{\text{coarse}}$ vs. h-MG: $\underbrace{(6,6) \rightarrow (5,6)}_{\text{'fine'}} \rightarrow \underbrace{(4,6) \rightarrow (3,6) \rightarrow (2,6) \rightarrow (1,6)}_{\text{effective coarse}}$.

[22] In the following, we drop the unit s.

For p-multigrid, it is useful to refine the performance model. P-multigrid uses, in the developed hybrid multigrid solver, AMG as coarse-grid preconditioner. AMG creates algebraically internally l_{AMG} levels of decreasing size. The cost of an embedded cycle consequently arises from the product $\beta = l_{AMG} \cdot \beta_{AMG}$ where β_{AMG} is the cost of one AMG level. For the sake of simplicity, we set $l_{AMG} = l_h$. The resulting model is as follows:

$$l_o = l_k, \qquad \beta = l_h \cdot \beta_{\text{AMG}}, \qquad \beta_{\text{AMG}} = 0.00029 \pm 0.00014, \qquad \overline{N}_{iter,i} = 4. \tag{61c}$$

Please note that the cost of an AMG level β_{AMG} varies strongly as shown in Figure 25. The value is generally ≈ 0.0002. For $l = 7$ and $l = 9$, the value is surprisingly high. This is also apparent in Figure 18 where the coarse-grid problems for $l = 7$ and $l = 9$ are approximately equally expensive, and the coarse-grid problem for $l = 8$ is significantly cheaper. Figure 26 visualizes the fraction of time spent on each multigrid level for 1 and 60 nodes in a pie diagram. In contrast to h-MG, the majority of time is spent on solving the coarse-grid problem in the scaling limit.

Based on the strong-scaling limit model derived above for p-MG, we can deduce an alternative strategy to minimize the time to solution for simulations running close to the scaling limit (for the sake of simplicity we assume $\alpha = \beta_{AMG}$):

$$\frac{t_0}{\alpha} = N_{iter,o} \cdot \left(l_k + \overline{N}_{iter,i} \cdot l_h \right) \stackrel{!}{=} \min, \tag{62}$$

which requires a reduction in the total number of level visits (no matter if outer or embedded levels). This strategy is in clear contrast to the normal strategy in which the primary goal is to reduce the operation evaluations on the fine grid, readily accepting additional operations on coarser levels. To clarify, let us revisit the observations in Subsection 5.6 where we saw that using coarse PCG with AMG as preconditioner significantly outperforms the coarse-grid solver, in which only one V-cycle of AMG is employed because it requires only half of the number of iterations. Inserting the numbers $N'_{iter,o} = 2 \cdot N_{iter,o}$ and $\overline{N}'_{iter,i} = 1$ into Equation (62) results in the following speedup estimate, in favor of the single V-cycle coarse-grid solver:

$$\frac{N_{iter,o}}{2 \cdot N_{iter,o}} \cdot \frac{l_k + 4 \cdot l_h}{l_k + 1 \cdot l_h} \overset{\substack{l_h=5 \\ l_k=2}}{=} 1.57. \tag{63}$$

This hypothesis requires verification by future experiments. The ad-hoc strong-scaling limit model presented above should be refined and generalized for all configurations of the developed hybrid multigrid solver (incl. DG, hp-multigrid). Furthermore, comparisons to performance models documented in the literature should be conducted.

As a final remark, it should be noted here that we can "learn" from h-MG also in regard to the context of p-MG with AMG. H-multigrid performs significantly more embedded cycles than p-multigrid ($\times 100$). Each embedded cycle is very cheap because it is performed on a system of exactly one macro cell owned by only one process, requiring absolutely no communication between embedded cycles. Working with only one CPU in the context of p-MG with AMG is obviously not feasible, however, a repartitioning of the coarse-grid problem and its assignment to a small working set of CPUs (maybe one node) might be sufficient to reduce the costs of the embedded algebraic cycles such that performing multiple embedded cycles would not affect the strong-scaling limit in a negative way. Similar repartitioning of the mesh on coarse-multigrid level has been employed by Rudi et al. [87].

5.9 Weak scaling

As a by-product of the strong-scaling analysis, we can mimic a weak-scaling analysis by connecting data points with the same $\frac{dofs}{procs}$, as shown in Figures 17-24. Excellent weak scalability for constant degree k can be observed for all considered multigrid variants, expressed as nearly horizontal lines. This was expected because of the size-independent nature of multigrid algorithms.

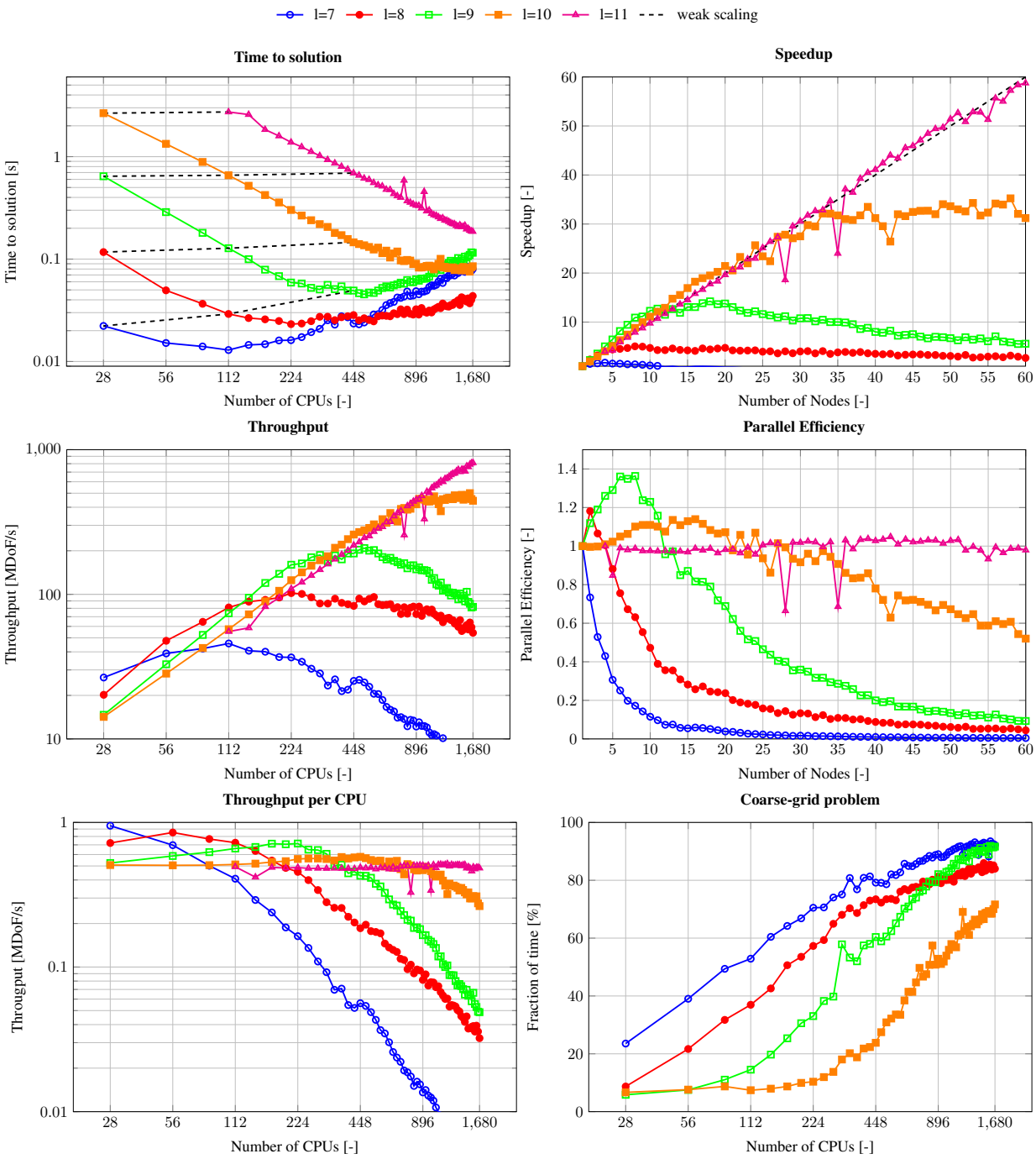

Figure 17: Strong scaling for 2D CG curved mesh with p-MG (k=6)

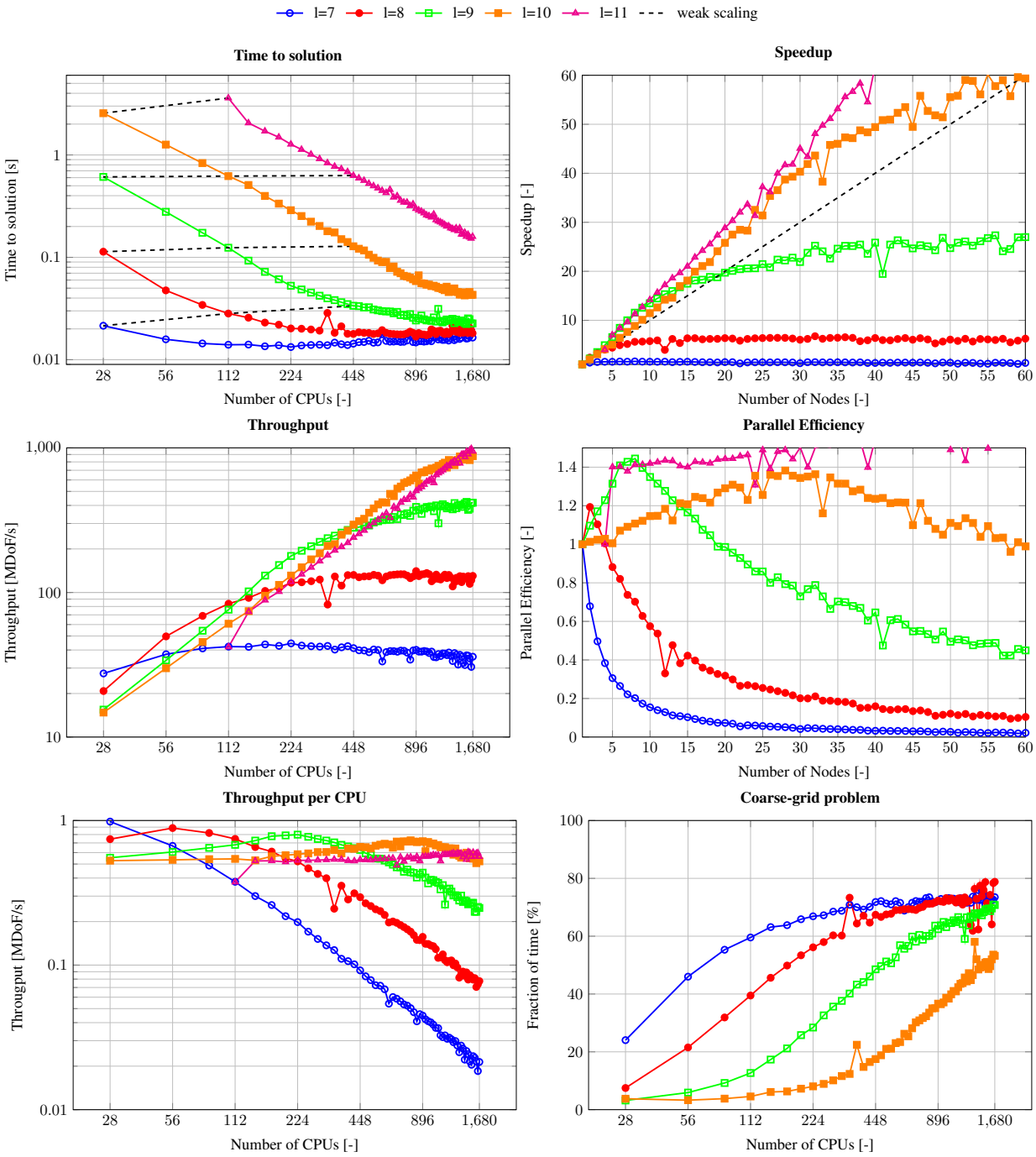

Figure 18: Strong scaling for 2D CG curved mesh with h-MG (k=6)

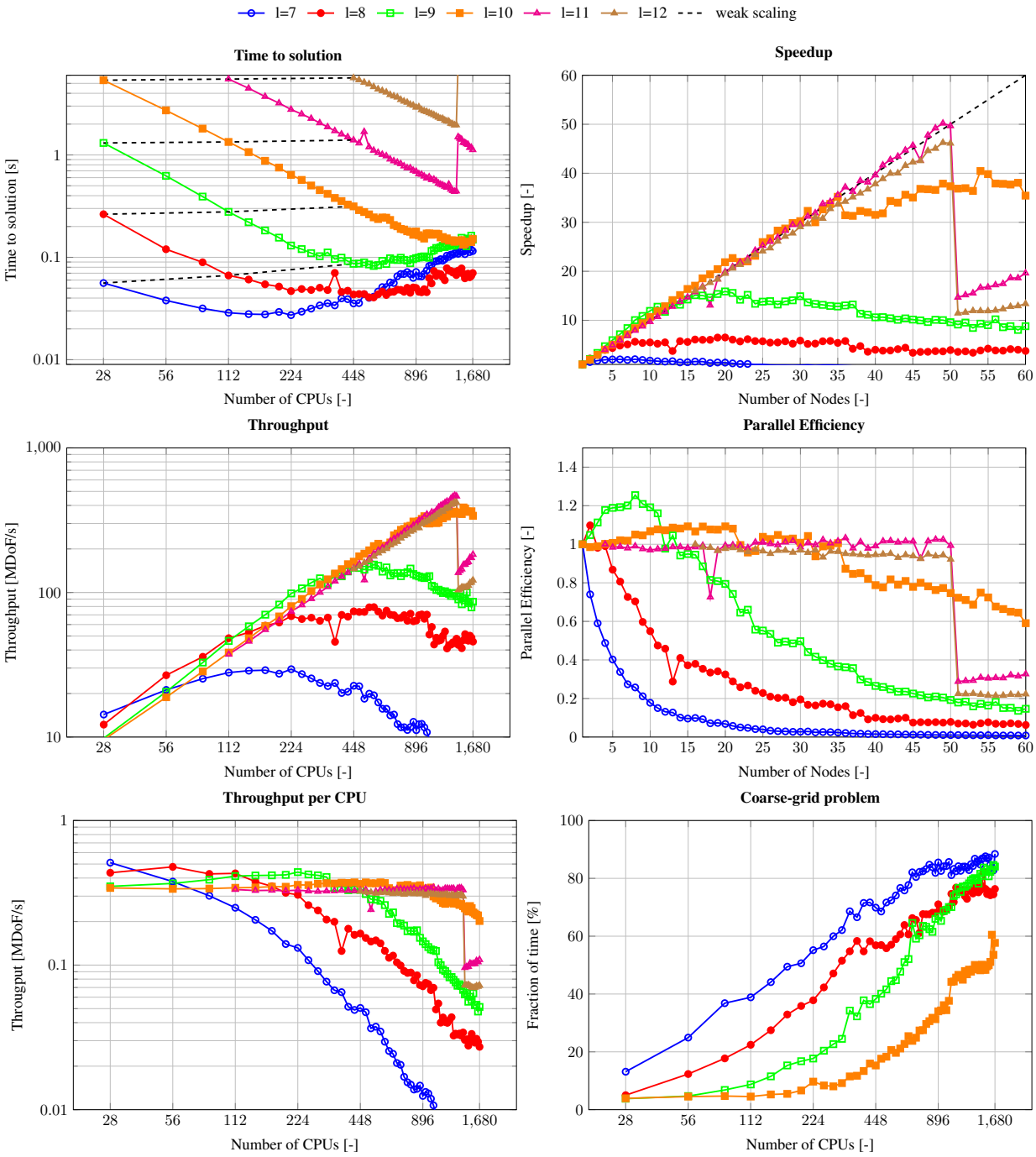

Figure 19: Strong scaling for 2D DG curved mesh with p-MG (k=6)

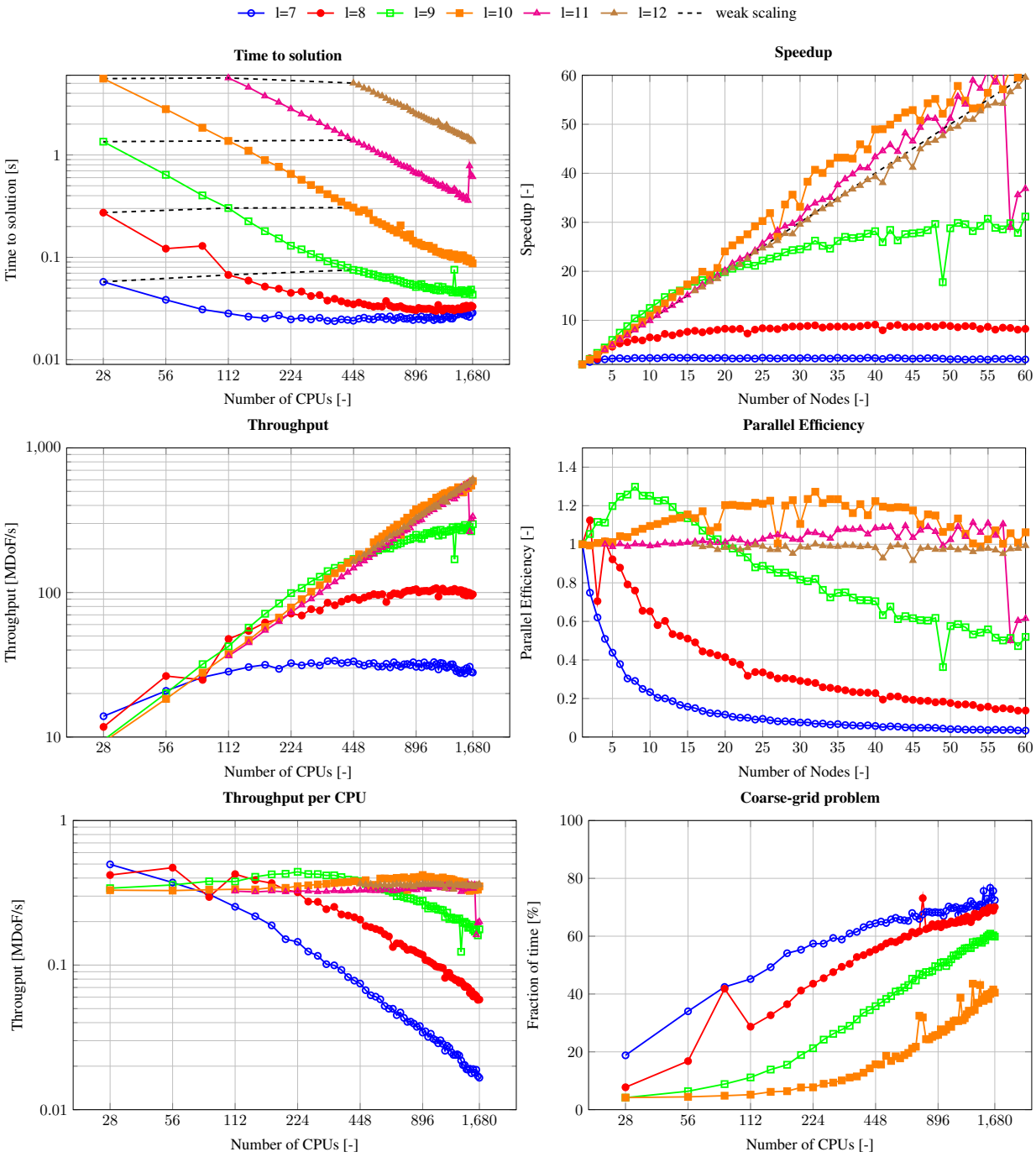

Figure 20: Strong scaling for 2D DG curved mesh with h-MG (k=6)

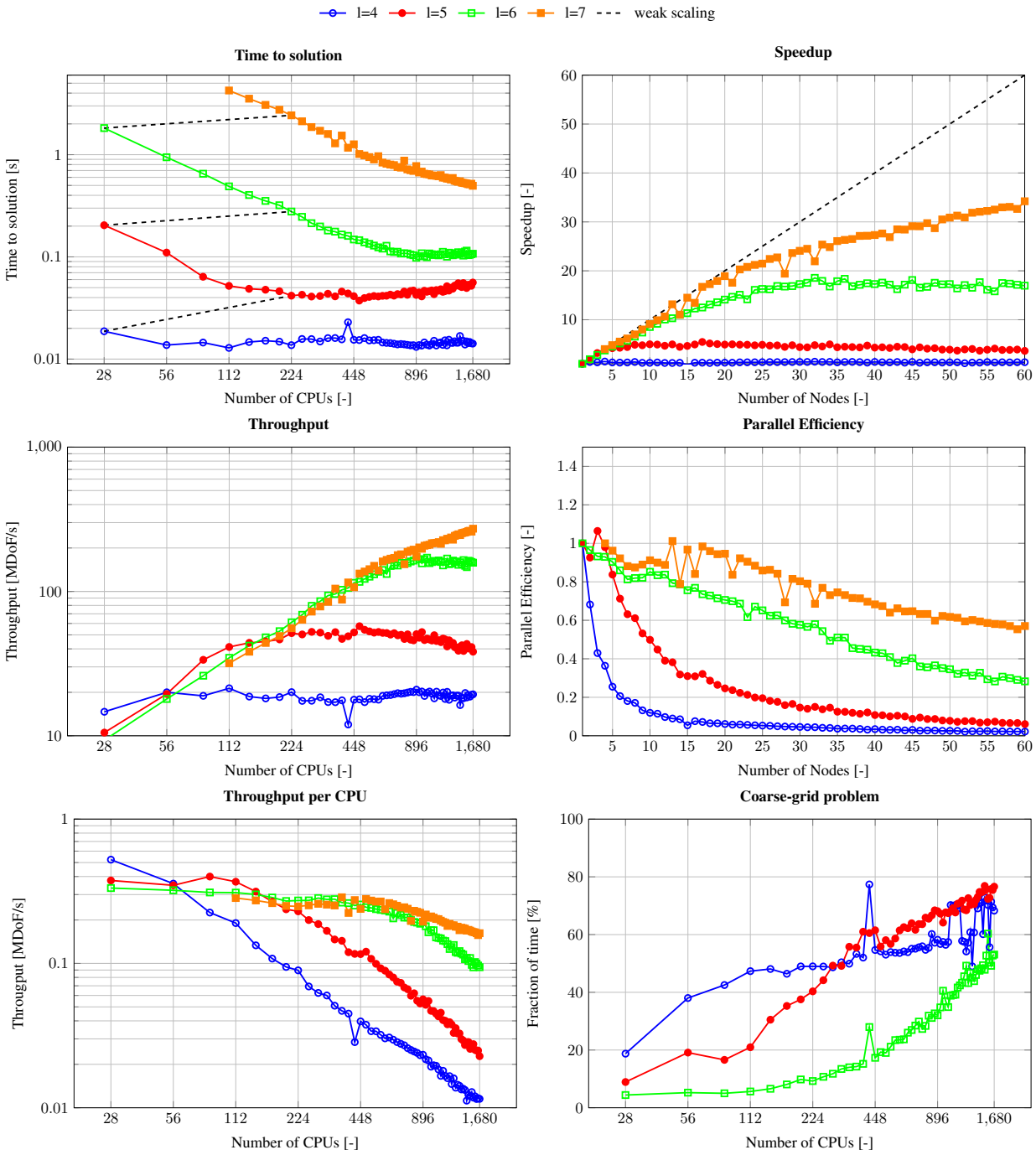

Figure 21: Strong scaling for 3D CG curved mesh with p-MG (k=4)

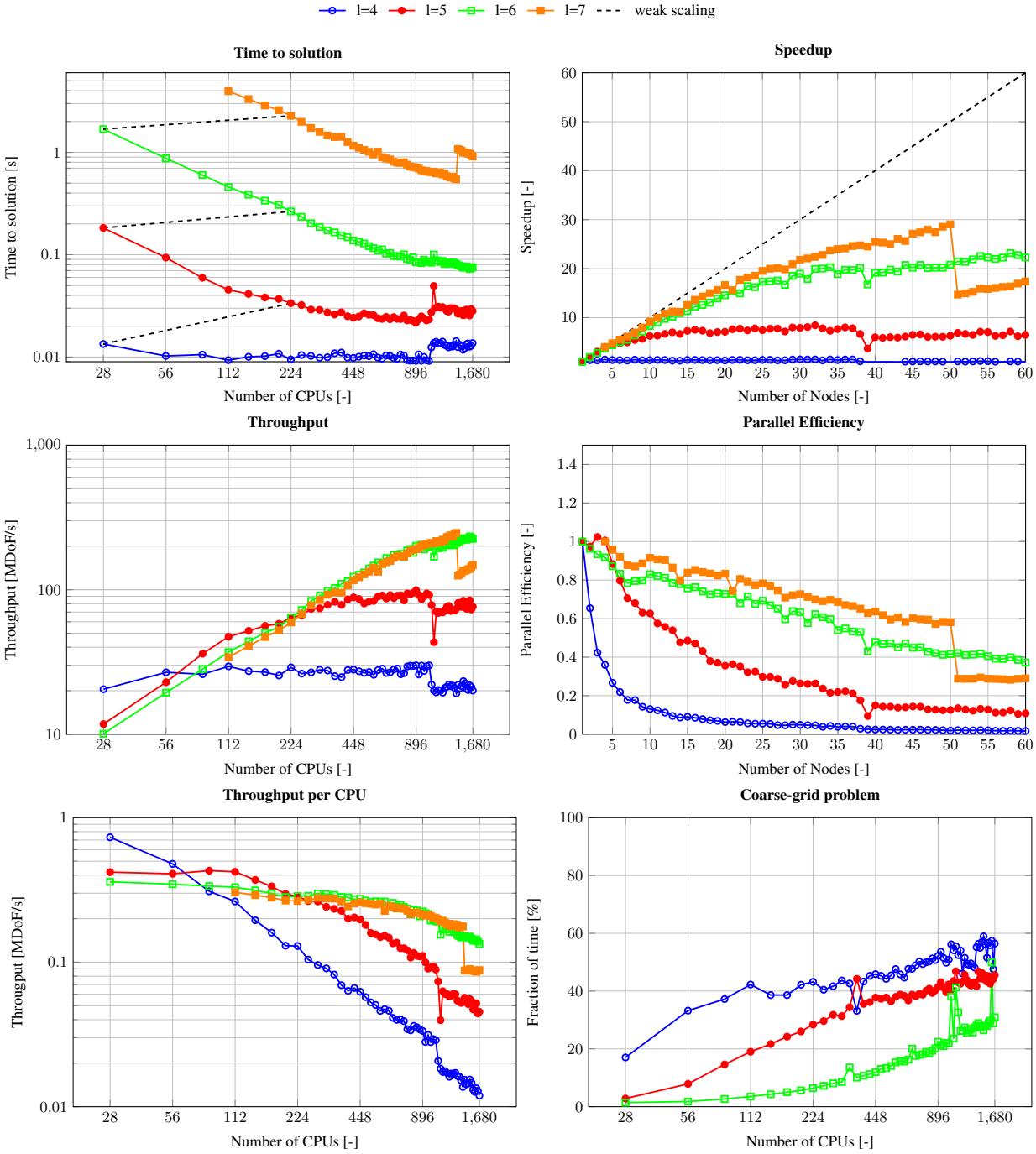

Figure 22: Strong scaling for 3D CG curved mesh with h-MG (k=4)

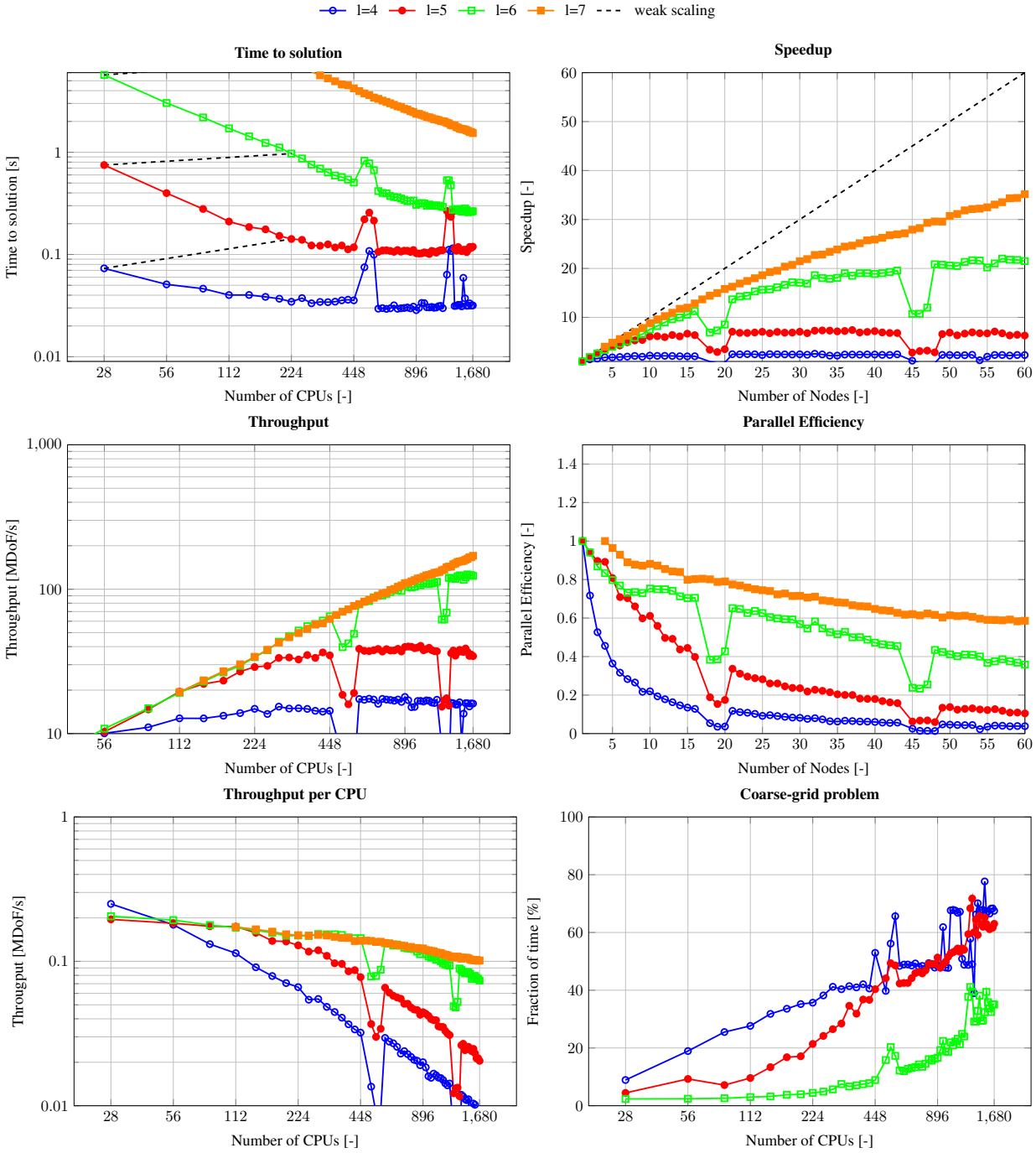

Figure 23: Strong scaling for 3D DG curved mesh with p-MG (k=4)

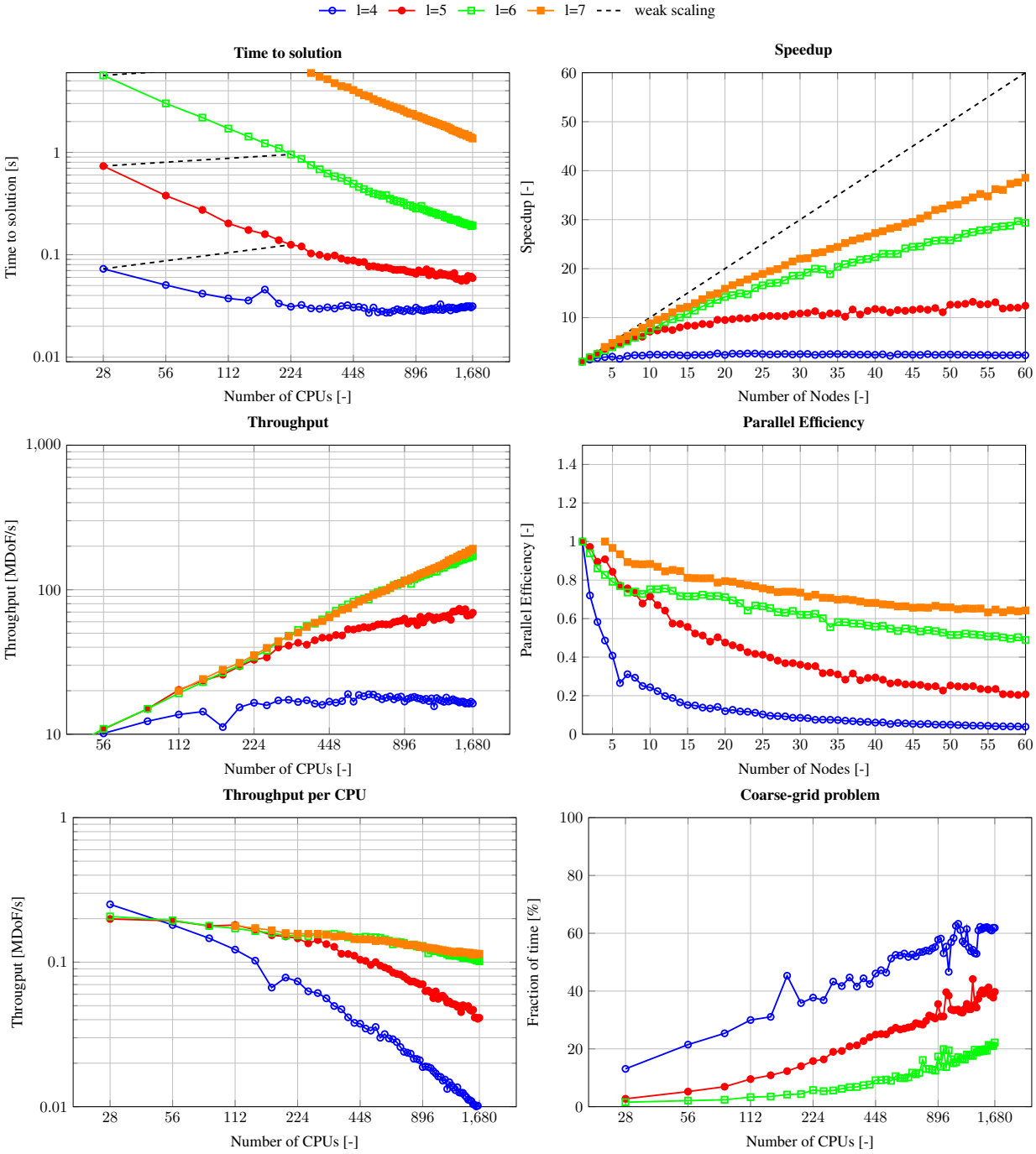

Figure 24: Strong scaling for 3D DG curved mesh with h-MG (k=4)

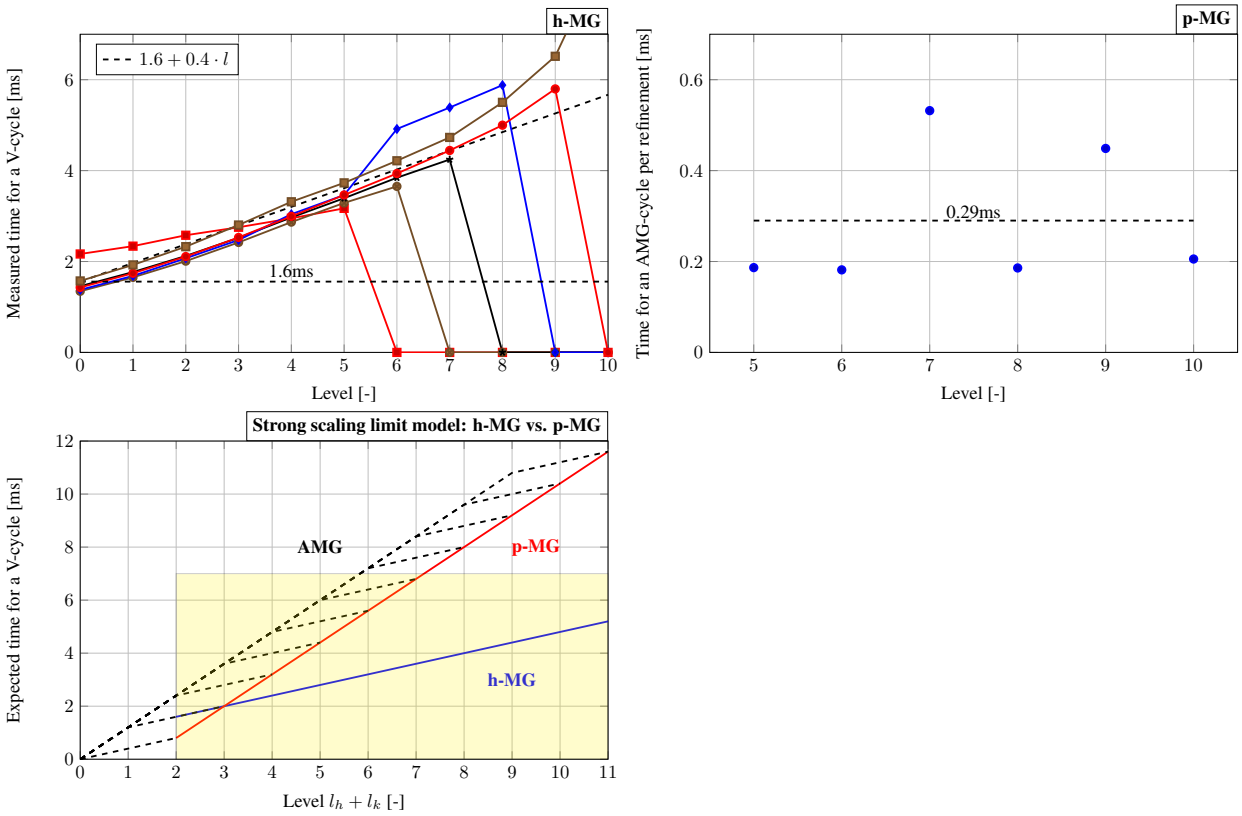

Figure 25: Modeling of the strong-scaling limit for 2D CG with p-MG and h-MG

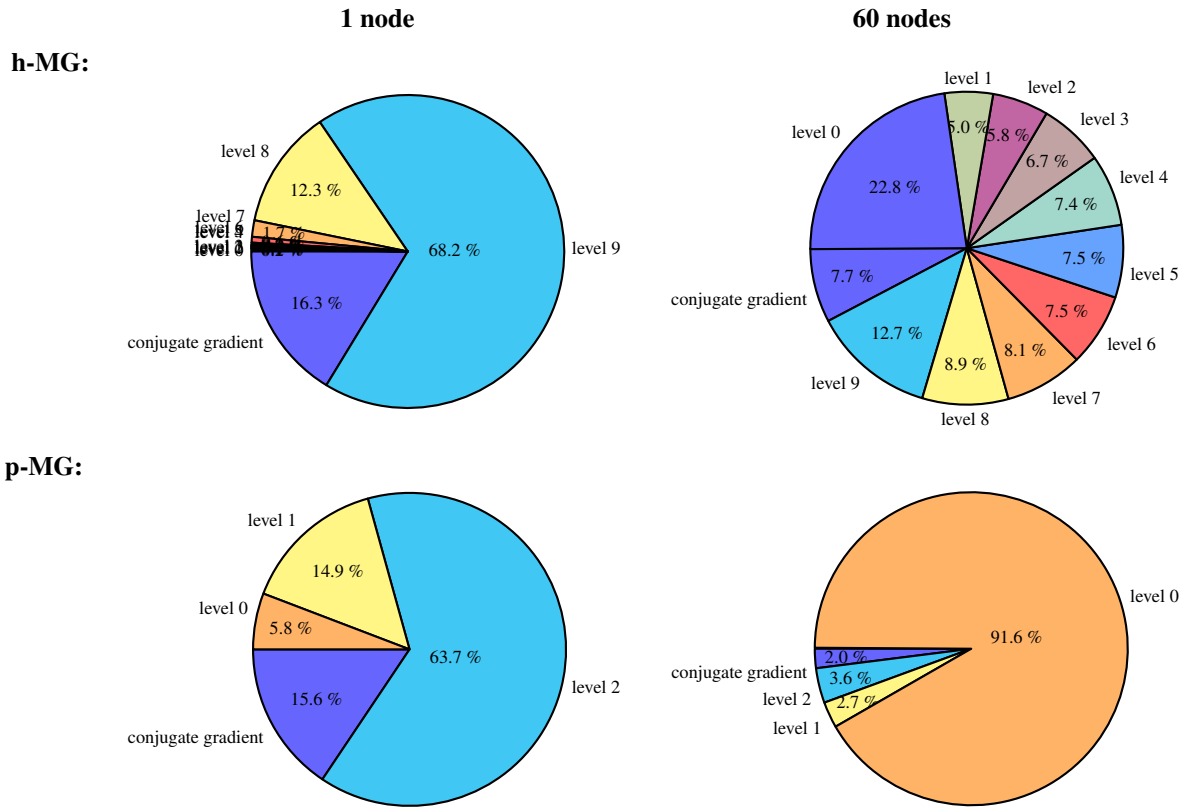

Figure 26: Profiling of PCG for $k = 6$ and $l = 9$ for p-MG/h-MG (with 1 and 60 nodes) for 2D CG

6 Application: convection–diffusion equation

In this section, the solution of the convection–diffusion equation by the presented hybrid multigrid solver is considered:

$$\frac{\partial u}{\partial t} + \nabla \cdot (\boldsymbol{a}\, u) - \nabla \cdot (\kappa \nabla u) = f. \tag{64}$$

The following analysis is restricted to the solution of this equation for stationary and sourceless problems, *i.e.* $\partial u/\partial t = 0$ and $f = 0$.

This section is structured as follows. Subsection 6.1 presents the spatial discretization of the convection–diffusion equation. Subsection 6.2 investigates the suitability of the developed hybrid multigrid solver for convection-dominated problems, by presenting numerical results for the test case called boundary-layer problem.

6.1 Spatial discretization

The DG discretization of Equation (64) reads as follows for the diffusive term, using SIP (see also Subsection 2.2):

$$\begin{aligned} &(\nabla v_h,\, \nabla u_h)_{\Omega^e} - (\nabla v_h,\, (u_h - u_h^*)\, \boldsymbol{n})_{\Gamma^e} - (\nabla v_h,\, \sigma_h^* \cdot \boldsymbol{n})_{\Gamma^e}\,, \\ &\qquad u_h^* = \{\{u_h\}\}, \qquad \boldsymbol{\sigma}_h^* = \{\{\nabla u_h\}\} - \tau [\![u_h]\!], \end{aligned} \tag{65a}$$

and for the convective term:

$$-\,(\nabla v_h,\, \boldsymbol{a} u_h)_{\Omega^e} + (v_h,\, (\boldsymbol{a} u_h)^*\, \cdot \boldsymbol{n})_{\Gamma^e}\,. \tag{65b}$$

For the flux term $(\boldsymbol{a} u_h)^*$, we use the local Lax–Friedrichs method. For a detailed derivation of Equation (65), the reader is referred to [23].

6.2 Numerical results for the boundary-layer problem

In order to investigate the suitability of the presented hybrid multigrid solver for convection-dominated problems, the quasi-2D boundary-layer problem is considered. The square domain $\Omega = [-1, +1]^2$ is flowed through from left to right with uniform velocity $\boldsymbol{a} = (1, 0)^T$. At the inlet ($u = 1$) and the outlet ($u = 0$), Dirichlet boundary conditions are applied. At the remaining boundaries, homogeneous Neumann boundary conditions are applied. Diffusivity is uniform and is varied between $0.001 \leq \kappa \leq 1000$ so that Péclet numbers of range $0.002 \leq \mathrm{Pe} = \frac{||\boldsymbol{a}||_\infty \cdot L}{\kappa} \leq 2000$ can be investigated. The coarse grid is a single macro cell, which is uniformly refined l times. The number of refinements is varied between $0 \leq l \leq 10$, and four polynomial degrees ($k = 3, 5, 9, 13$) are considered. The boundary-layer problem is preconditioned by p-MG with AMG, approximately inverting the full convection-diffusion operator. A Chebyshev smoother is applied on all multigrid levels. The preconditioned conjugate gradient method has been replaced by the flexible generalized minimal residual (FGMRES) method on the fine level and by the generalized minimal residual (GMRES) method on the coarse level. The diffusive term and the convective term of Equation (65) are implemented in separate discrete operators, which are applied sequentially to a vector.

Figure 27 shows the throughput $\frac{\text{dofs}}{\text{time to solution}}$ depicted over the element Péclet number

$$\mathrm{Pe}_e = \frac{||\boldsymbol{a}||_\infty \cdot h}{2\kappa} = \frac{||\boldsymbol{a}||_\infty \cdot L}{\kappa} \cdot \frac{1}{2^{l+1}} = \mathrm{Pe} \cdot \frac{1}{2^{l+1}}, \tag{66}$$

for the investigated polynomial degrees and refinements. The following observations can be made, based on Figure 27. Throughput (and also the iteration count) is independent of the element Péclet number up to an order-independent threshold, the critical element Péclet number $\mathrm{Pe}_e = 1$. For $\mathrm{Pe}_e > 1$, the drastic increase in iteration numbers[23] leads to a rapid drop in throughput. Based on these observations, the following relationship between refinement level l and the maximal Péclet number can be derived:

$$\mathrm{Pe}_e \overset{!}{=} 1 \quad \rightarrow \quad \mathrm{Pe}_{\max}(k, l) = \mathrm{Pe}_{\max}(l) = 2^{l+1}, \tag{67}$$

corresponding to a uniform refinement such that for all elements $\mathrm{Pe}_e \leq 1$ is fulfilled [84].

For refinement levels $7 \leq l \leq 10$, the observed throughput lies between $3.0 - 4.3$MDoF/s. These values are significantly lower than the values measured in the case of the solution of the Poisson-problem (see Figure 10). The main reason

[23] It is not surprising, that for increasing element Péclet number, the iteration numbers increase since oscillations occur for $\mathrm{Pe}_e > 1$ and oscillations become widespread for $\mathrm{Pe}_e \gg 1$ [34]. Furthermore, the chosen Chebyshev smoother is unsuited for strongly unsymmetrical matrices as it is the case, for example, for $\mathrm{Pe}_e > 1$.

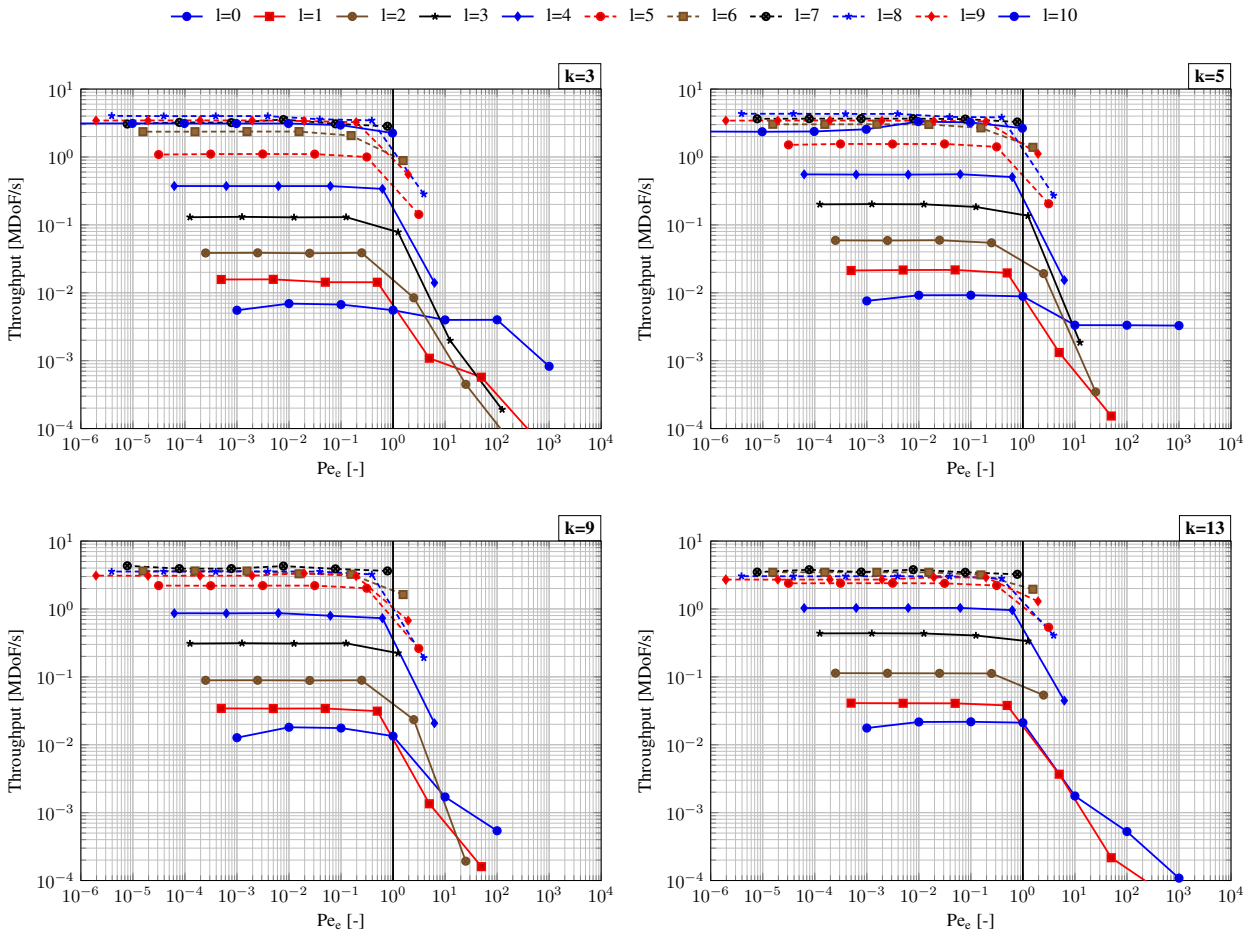

Figure 27: Dependency of the throughput of the convection–diffusion operator on Pe_e for different polynomial degrees k and refinement levels l on a Cartesian 2D mesh for p-MG with AMG

for this is that all data has to be fetched from main memory twice as often as in the Poisson problem case because the convection operator and the diffusion operator are applied sequentially. Merging of the two operator applications[24] promises a speedup by a factor up to 2.

In future work, the influence of matrix-free implementations of alternative smoothers [84] and of anisotropic multgrid transfer operators [39] on the value of the critical element Péclet number shall be investigated. Additionally, the discretization of the coarse-grid problem shall be examined. In the investigation presented in this section, we used CG on the coarse grid since it is the default setting of the developed hybrid multigrid solver. For convection-dominated problems, however, CG is not stable, resulting in a poor auxiliary-space solution. Rediscretizing the coarse-grid problem with DG is an alternative. However, we have demonstrated in Section 5.6 that the AMG library used by the presented hybrid multigrid solver is not suitable for DG. Therefore, alternative libraries shall be tested regarding their suitability for DG. Alternatively, it would be possible to rediscretize the coarse-grid problem with SUPG, as was done by Mascarenhas et al. [69], requiring the implementation of a new matrix-free upwind-weighted restriction operator.

[24] The benefit of merging operator applications has been discussed by Fehn et al. [27] for convection and diffusion operators inside a compressible Navier–Stokes solver as well as by Kronbichler and Allalen [56] inside a Chebyshev smoother.

7 Application: incompressible Navier–Stokes equations

Section 7, the final application section presents the results of the developed hybrid multigrid solver as a part of the in-house Navier–Stokes solver `INDEXA`, which stands for: A high-order discontinuous Galerkin solver for turbulent incompressible flow towards the EXA scale. `INDEXA` was first presented by Fehn [23] and new features have been developed and added since then (see [24–28, 53]).

Subsection 7.1 demonstrates the underlying governing equations of the incompressible Navier–Stokes solver `INDEXA` as well as its temporal and spatial discretization, focusing on the pressure Poisson equation. In Subsection 7.2, numerical results are presented for the FDA benchmark nozzle problem, which we discretize with a non-trivial coarse-grid mesh. Furthermore, the benefits of the developed hybrid multigrid solver for such problems are discussed from a performance point of view.

7.1 Governing equations and numerical discretization

The unsteady incompressible Navier–Stokes equations consist of the momentum and the continuity equation on a domain $\Omega \subset \mathbb{R}^d$:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{u}) + \nabla p = \boldsymbol{f} \quad \text{in } \Omega \times [0,T], \tag{68a}$$

$$\nabla \cdot \boldsymbol{u} = 0 \quad \text{in } \Omega \times [0,T], \tag{68b}$$

where $\boldsymbol{f}$ donates the body force vector, $\boldsymbol{u}$ the velocity vector, and p the kinematic pressure. The flux tensor is $\boldsymbol{F}(\boldsymbol{u}) = \boldsymbol{F}_c(\boldsymbol{u}) - \boldsymbol{F}_v(\boldsymbol{u}) = \boldsymbol{u} \otimes \boldsymbol{u} - \nu \nabla \boldsymbol{u}$ where ν denotes constant kinematic viscosity. For detailed information on the choice of appropriate boundary conditions, the reader is referred to [24]. As an initial condition, a velocity field is prescribed that fulfills both the divergence-free constraint (see Equation (68b)) and the Dirichlet boundary conditions.

Temporal discretization: high-order dual-splitting projection scheme

We use the dual-splitting projection scheme [52] based on BDF time integration of order $J = 2$, which consists of the following four solution substeps:

$$\frac{\gamma_0 \hat{\boldsymbol{u}} - \sum_{i=0}^{J-1} \left(\alpha_i \boldsymbol{u}^{n-i}\right)}{\Delta t} = -\sum_{i=0}^{J-1} \left(\beta_i \nabla \cdot \boldsymbol{F}_c(u^{n-i})\right) + \boldsymbol{f}\left(t_{n+1}\right), \tag{69a}$$

$$-\nabla^2 p^{n+1} = -\frac{\gamma_0}{\Delta t} \nabla \cdot \hat{\boldsymbol{u}}, \tag{69b}$$

$$\hat{\hat{\boldsymbol{u}}} = \hat{\boldsymbol{u}} - \frac{\Delta t}{\gamma_0} \nabla p^{n+1}, \tag{69c}$$

$$\frac{\gamma_0}{\Delta t} \boldsymbol{u}^{n+1} - \nabla \cdot \boldsymbol{F}_v\left(\boldsymbol{u}^{n+1}\right) = \frac{\gamma_0}{\Delta t} \hat{\hat{\boldsymbol{u}}}, \tag{69d}$$

where the coefficients γ_0 and α_i are connected to the BDF time integration. In the first substep, the body force term and the convective term are treated to determine an intermediate velocity $\hat{\boldsymbol{u}}$. The convective term is determined, using an extrapolation scheme of order J with coefficients β_i. In the second substep, the pressure Poisson equation is solved with divergence of $\hat{\boldsymbol{u}}$ forming the right-hand side. With the help of the received pressure field p^{n+1}, $\hat{\boldsymbol{u}}$ is projected in the third substep onto the space of divergence-free vectors, resulting in $\hat{\hat{\boldsymbol{u}}}$. Finally, in the fourth substep the treatment of the viscous term returns the final velocity $\boldsymbol{u}^{n+1}$.

Spatial discretization: high-order discontinuous Galerkin discretization

The spatial discretization of Equation (69) – based on the discontinuous Galerkin method – is performed using an approach similar to the one already presented in the context of the Poisson problem in Section 2.2. The spaces of the test and the trial functions for velocity $\boldsymbol{u}_h(\boldsymbol{x},t) \in \mathcal{V}_h^u$ and pressure $p_h(\boldsymbol{x},t) \in \mathcal{V}_h^p$ are given as:

$$\mathcal{V}_h^u = \left\{ \boldsymbol{u}_h \in \left[L_2(\Omega)\right]^d \,:\, \boldsymbol{u}_h|_{\Omega^e} \in \left[\mathcal{Q}_{k_u}(\Omega^e)\right]^d \,\forall \Omega_e \in \mathcal{T}_h \right\}, \tag{70a}$$

$$\mathcal{V}_h^p = \left\{ p_h \in L_2(\Omega) \,:\, p_h|_{\Omega^e} \in \mathcal{Q}_{k_p}(\Omega^e) \,\forall \Omega_e \in \mathcal{T}_h \right\}. \tag{70b}$$

We use mixed-order polynomials of degree $(k_u, k_p) = (k, k-1)$ for velocity and pressure. For the fully discretized form of Equation (69), the reader is referred to [24, 26]. Here, we shall only present the discretized form of the pressure

Poisson equation, which is relevant for testing of the developed hybrid multigrid solver:

$$l_h^e\left(q_h,\, p_h^{n+1}\right) = -\frac{\gamma_0}{\Delta t} d_h^e\left(q_h,\, \hat{\boldsymbol{u}}_h\right), \tag{71a}$$

with l_h^e being the (negative) Laplace operator (see also Section 2.2) and d_h^e being the velocity divergence term:

$$l_h^e(q_h, p_h) = \left(\nabla q_h, \nabla p_h\right)_{\Omega_e} - \left(\nabla q_h, 1/2[\![p_h]\!]\right)_{\partial\Omega_e} - \left(q_h, \{\{\nabla p_h\}\}\cdot\boldsymbol{n}\right)_{\partial\Omega_e} - \left(q_h, \tau[\![p_h]\!]\cdot\boldsymbol{n}\right)_{\partial\Omega_e}, \tag{71b}$$

$$d_h^e(q_h, \boldsymbol{u}_h) = -\left(\nabla q_h, \boldsymbol{u}_h\right)_{\Omega_e} + \left(q_h, \{\{\boldsymbol{u}_h\}\}\cdot\boldsymbol{n}\right)_{\partial\Omega_e}. \tag{71c}$$

7.2 Numerical results for the FDA benchmark nozzle problem

Problem description

In the following, we consider the FDA benchmark nozzle problem [67]. This problem involves a flow through a nozzle, which is a cylindrical pipe with gradual and sudden changes in the cross-section area, for different Reynolds numbers ($500 \leq \mathrm{Re_{th}} \leq 6500$), covering laminar, transitional, and turbulent flows. Fehn et al. [28] have recently investigated the FDA benchmark nozzle problem extensively, using the high-order discontinuous Galerkin method for $\mathrm{Re_{th}} \in \{500,\, 2000,\, 3500,\, 5000,\, 6500\}$. Our analysis is restricted to $\mathrm{Re_{th}} = 6500$, but represents the nozzle - as per the aforementioned publication - by a coarse grid consisting of 440 cells, which are uniformly refined l times. The total number of degrees of freedom, both of velocity and pressure unknowns, is the result of:

$$N_{DoFs}(l,\, k) = 440 \cdot 2^{3\cdot l} \cdot \left(3\cdot(k_u+1)^3 + (k_p+1)^3\right) = 440 \cdot 2^{3\cdot l} \cdot \left(3\cdot(k+1)^3 + k^3\right). \tag{72}$$

The number of relevant degrees of freedom for the pressure Poisson problem is:

$$N_{DoFs}^p(l, k) = 440 \cdot 2^{3\cdot l} \cdot k^3. \tag{73}$$

Our examination is restricted to configurations $\{(l, k_u)\,|\, l \in \mathbb{N}_0 \,\wedge\, k \in [3, 11] \,\wedge\, n_{l,k} \leq 29e6\}$, *i.e.* to a moderate number of refinement levels and to moderate high-order degrees. This application case is interesting because 440 coarse cells[25] are too much for purely matrix-free h-multigrid solvers for high-order DG, in the event that not enough coarse-grid levels can be created. The coarse PCG, simply preconditioned by the diagonal of the matrix, might need too much time to find an appropriate approximation of the solution for the system of equations of size $n_{0,k}^p = 440 \cdot k^3$. This would lead to the predominance of the coarse-grid problem (see also Section 5.5) and consequently to a disproportional predominance of the pressure Poisson problem in the overall algorithm of the dual-splitting projection scheme.

For a detailed description of the setup of the FDA benchmark nozzle problem, the reader is referred to [28]. In that publication, an incremental pressure-correction scheme in rotational formulation was used. It treats the viscous term and the convective term in the momentum equation implicitly in time. By contrast, we use the high-order dual-splitting projection scheme (see Section 7.1) and treat the convective term explicitly in time. We compute the time step, using the CFL criterion [26, 27]:

$$\Delta t = \frac{\mathrm{Cr}}{k_u^{1.5}} \cdot \frac{h_{\min}}{||\boldsymbol{u}||_{\max}}, \tag{74}$$

where Cr is fixed to 0.15, $||\boldsymbol{u}||_{\max} = 2\overline{u}_{th}$, and $h_{\min}$ is the minimum distance between two vertices of the mesh. We decided to forego the use of a precurser, instead we applied a parabolic inflow profile.

In order to solve the convective step (Equation 69a), the inverse mass matrix is applied to the right-hand side in a highly efficient matrix-free way [55]. For solving the projection step (Equation 69c) and the viscous step (Equation 69d), PCG is used with an absolute solver tolerance of 10^{-12} and a relative solver tolerance of 10^{-3}. It is preconditioned by the inverse mass matrix in both cases. FGMRES is used for solving the pressure Poisson problem (Equation 69b) with the same tolerance as above. It is preconditioned by a single V-cycle of the presented hybrid multigrid solver. The levels are created either via h-coarsening (h-multigrid) or via p-coarsening (p-multigrid) or via consecutive h- and p- coarsening (hp-multigrid). The hybrid multigrid solver uses a Chebyshev smoother on all levels, and the coarse problem is solved with PCG preconditioned by point Jacobi in the case of h-multigrid and by a single AMG V-cycle in the case of p- and hp-multigrid (see also Subsection 3.3).

The results are obtained on the Linux Cluster using 28 processes on a single node (see also Section 4.1). The simulations were run for 30 time steps.

[25] 440 cells are equivalent to a 3D hypercube refined 2-3 times in each direction: $2^{3\cdot 2} = 64 < 440 < 512 = 2^{3\cdot 3}$.

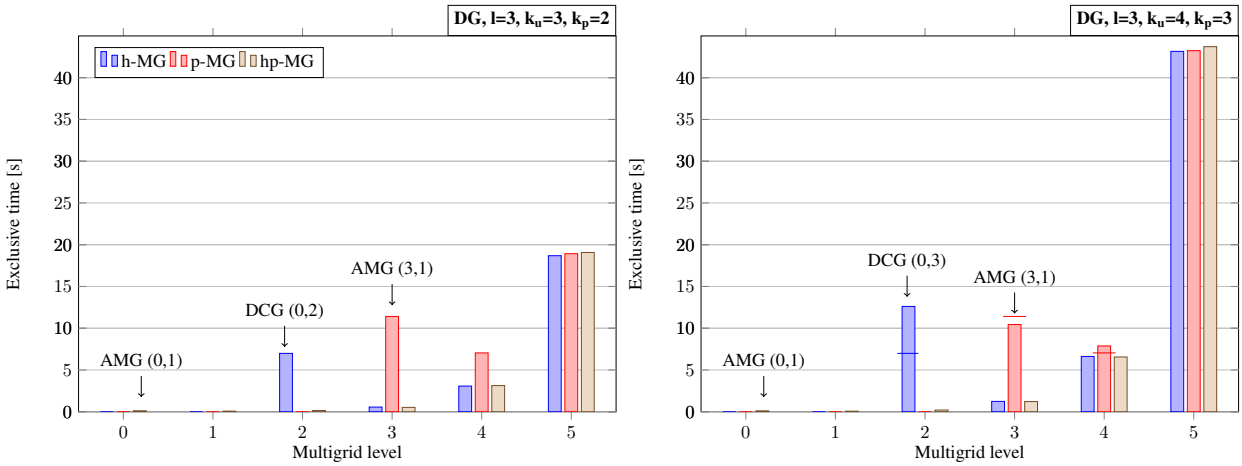

Figure 28: Exclusive time spent on each multigrid level of h-, p-, and hp-multigrid for two configurations of the spatial discretization the FDA benchmark nozzle problem. DCG stands for conjugate gradient solver preconditioned by the diagonal of the system matrix.

Results

Tables 13-18 summarize the results of the performance analysis conducted. As a starting point, let us take a look at the performance of h-multigrid for solving the FDA benchmark nozzle problem. As an assessment criterion, the fraction of time spent on solving the pressure Poisson problem and the projection step is used in Table 13. It can be clearly seen that with a decreasing number of levels l and an increasing polynomial degree k, the fraction of time spent on solving the pressure Poisson problem increases significantly. For refinements $l \leq 1$, more than 90% of time is spent on solving this problem. As discussed above, this is expected since the size of the coarse-grid problem increases making the solution of the coarse-grid problem disproportionally expensive. For $l = 0$, actually no multigrid preconditioning is/can be performed at all. Only a preconditioning of PCG by the diagonal of the system matrix is possible, leading to a non-optimal algorithm.

Table 14 shows the speedup of solving the pressure Poisson problem if h-MG is replaced by p-MG with AMG. For almost all configurations, a significant speedup can be observed. For configurations that are relevant in practice ($1 \leq l \leq 2$, $3 \leq k \leq 7$), a speedup of 2-10 is reached. For $l = 0$, speedup is even more than 65. For $l = 3$, the speedup is less pronounced because the coarse-grid problem that is constructed via a sequence of three h-coarsening steps is significantly smaller than the fine-grid problem. For the configuration ($l = 3$, $k_u = 3$), pure h-multigrid is even faster than p-MG with AMG.

Since this is the only configuration in which a slowdown of approx. 10% rather than a speedup can be observed, let us take a closer look at it to understand why p-MG is slower in this case. Figure 28 breaks down the time spent on each multigrid level and shows the exclusive time spent on each level both for p-MG and for h-MG for ($l = 3$, $k_u = 3$).

H-MG consists of one level per refinement, *i.e.* four multigrid levels. A disproportional amount of time (23%) is spent on solving the coarse-grid problem. In the case of p-MG, there are only three multigrid levels. On the finest level, approximately the same absolute time is spent by p-MG as by h-MG. However, on the coarser levels significantly more absolute time is spent by p-MG, making it slower for this configuration. The slowness of the coarser levels is understandable since, on the one hand, p-coarsening reduces the order only by one from $k_p = 2$ to $k_p = 1$, and it is not able to halve the degrees of freedom in each direction. On the other hand, AMG has to solve an equation system with approximately 18 times more unknowns. In the light of the sheer size difference between the coarse-grid equation system in the case of h-MG and of p-MG with AMG, it is surprising that AMG needs only approximately twice as much time for solving the coarse-grid problem.

The observation that, in the case of p-MG for the given configuration, a significant proportion of preconditioning time is spent on the (coarse) AMG level because the number of unknowns has not been sufficiently reduced due to the lack of possibility to create additional levels via p-coarsening, motivated us to decrease the number of unknowns even more by creating new matrix-free coarse levels in a sequence of h- and p-coarsening steps. The time spent on each level of

hp-MG using AMG as a coarse-grid preconditioner[26] is shown additionally in Figure 28. The timings for the three finest levels are practically identical to the timings in the case of h-MG. Processing the new matrix-free coarse levels and the algebraic coarse problem together in a negligible amount of time (0.5% of the whole multigrid preconditioning time) produces the speedup of 1.26.

Figure 28 also shows the timings of the configuration $l = 3$ and $k_p = 3$ in which only the polynomial degree has been increased by one. In this case, we can observe again a speedup of more than 20% in favor of hp-MG against h-MG. What is different from the previous configuration, is that p-MG is faster (by 3%) than h-MG. This is due, on the one hand, to p-coarsening, which in this configuration is more efficient in decreasing the number of unknowns, and, on the other hand, to the time spent on the algebraic coarse-grid problem staying approximately the same, while the time to solution of the coarse-grid problem in the case of h-MG has doubled in comparison to the results obtained for $k_p = 2$.

Table 15 shows the speedup in solving the pressure Poisson problem if h-MG is replaced by hp-MG with AMG. The speedup is comparable to the speedup if h-MG is replaced by p-MG (see Table 14), with hp-MG being faster (up to a factor of two) for moderate high-order elements and few refinement levels.

Until now, we have only discussed the speedup of solving the Poisson problem by using p-/hp-multigrid. However, as Table 13 indicates, the time reduction of the immensely costly solution process of the Poisson problem also leads to the speedup of the implementation of the whole dual-splitting projection scheme[27]. Table 16 shows the maximal speedup of the overall algorithm reached with either p- or hp-multigrid. For moderate degrees and refinement levels, speedup ranges between 1.66-7.01. The maximal observed speedup is approximately 42. Table 17 shows the percentual time spent on solving the Poisson problem and the projection step. It is clear that solving the Poisson problem is still the most expensive part of the overall algorithm (40-70%). However, it is only three times more expensive than performing the projection. Finally, Table 18 shows the maximal average throughput achieved for the solution of a single Poisson problem: for relevant configurations, throughput greater than 5MDoF/s can be observed.

The promising observation obtained by this performance analysis that hp-multigrid with AMG has given the best results for moderate high-order elements and few refinement levels, should be taken as a reason to investigate hp-multigrid in more detail in the future.

[26] In fact, the equation system on the coarsest level is so small that preconditioning with the diagonal of the system matrix only would probably suffice.

[27] Please note that the speedup of the pressure Poisson solver by using the developed hybrid multigrid solver also leads to a speedup of the incremental pressure-correction scheme, as used by Fehn et al. [28], because this scheme consists of the solution of the pressure Poisson problem in a substep, as does the dual-splitting projection scheme. However, the overall speedup of the incremental pressure-correction scheme will not be as high because the fraction of time spent on the pressure Poisson solver is not that significant in this case.

Table 13: Fraction of time spent on solving the pressure Poisson problem and the projection step (h-MG)

l	k_u								
	3	4	5	6	7	8	9	10	11
0	96/2	97/1	98/1	98/1	99/1	99/1	99/0	99/1	99/0
1	94/3	93/3	93/3	93/3	94/3	94/3	95/2	96/2	97/1
2	73/13	67/14	70/13	75/10	80/9	-	-	-	-
3	59/19	54/22	-	-	-	-	-	-	-

Table 14: Speedup of solving the pressure Poisson problem with p-MG (with AMG) instead of h-MG

l	k_u								
	3	4	5	6	7	8	9	10	11
0	6.28	12.64	19.13	26.32	34.44	40.53	49.79	56.25	65.68
1	4.68	5.64	7.96	9.40	11.34	12.27	14.82	16.77	20.25
2	2.28	2.21	2.09	2.28	2.45	-	-	-	-
3	**0.83**	1.03	-	-	-	-	-	-	-

Table 15: Speedup of solving the pressure Poisson problem with hp-MG (with AMG) instead of h-MG

l	k_u								
	3	4	5	6	7	8	9	10	11
0	6.43	13.79	20.76	26.84	35.50	40.62	50.08	55.94	66.39
1	9.48	10.04	11.47	12.52	13.50	13.36	15.31	17.68	20.30
2	3.00	2.51	2.31	2.52	2.72	-	-	-	-
3	1.20	1.26	-	-	-	-	-	-	-

Table 16: Maximal speedup of solving the overall dual-splitting projection scheme

l	k_u								
	3	4	5	6	7	8	9	10	11
0	5.38	10.06	14.10	18.49	23.99	25.77	31.95	34.23	41.74
1	6.18	6.06	6.67	7.01	7.68	7.76	9.09	10.54	12.64
2	1.95	1.69	1.66	1.84	2.02	-	-	-	-
3	1.11	1.34	-	-	-	-	-	-	-

Table 17: Fraction of time spent on solving the pressure Poisson problem and the projection step for the best MG configuration

l	k_u								
	3	4	5	6	7	8	9	10	11
0	81/9	71/12	66/14	68/15	67/15	63/17	63/16	60/17	62/17
1	61/19	56/21	54/21	52/21	53/21	55/20	57/20	57/20	60/17
2	47/24	45/24	50/22	55/19	59/19	-	-	-	-
3	55/20	58/20	-	-	-	-	-	-	-

Table 18: Maximal average throughput achieved for solving a single pressure Poisson problem [MDoF/s]

l	k_u								
	3	4	5	6	7	8	9	10	11
0	1.07	2.69	3.74	4.11	4.52	5.08	4.72	4.75	4.29
1	5.32	7.57	8.44	8.33	7.42	6.60	6.14	5.73	4.90
2	7.41	8.72	7.97	7.13	6.17	-	-	-	-
3	5.30	5.70	-	-	-	-	-	-	-

8 Conclusions & outlook

In the course of this Master's thesis, an existing efficient matrix-free h-multigrid solver [59] for high-order continuous and discontinuous Galerkin methods was developed further and extended into a hybrid multigrid solver with the aim of solving high-order DG problems with complex, non-trivial geometries efficiently on modern CPU hardware. The basis of the efficiency of the developed hybrid multigrid solver is threefold. Firstly, the solver geometrically constructs as many multigrid levels as possible via h- and p-coarsening such that the given operator can be rediscretized on every level and the size of the coarse-grid problem is reduced as much as possible. Secondly, the rediscretization of the operator on every multigrid level enables the evaluation of almost all multigrid components in a highly efficient matrix-free way, especially for high orders. Thirdly, an efficient algebraic multigrid solver is applied to solve the relatively small coarse-grid problem.

The application of almost all multigrid components was implemented in a matrix-free way, based on sum factorization. It includes the application of discrete operators and of intergrid transfer operators as well as the computation of the diagonal of the system matrix and its matrix representation via columnwise reconstruction. The remarkable implementation efficiency was demonstrated in the comparison of experimental measurements with theoretical expectations, conducted for CG/DG, 2D/3D, Cartesian/curved mesh.

In order to demonstrate the overall efficiency of the developed hybrid multigrid solver, it was applied to the Poisson problem, to the convection–diffusion equation, and to the unsteady incompressible Navier–Stokes equations discretized in time with the dual-splitting projection scheme. As a performance reference, we used the original h-multigrid solver that we know to be efficient for simple coarse grids.

Experiments on the Poisson problem on a uniformly refined hypercube were performed for CG and DG in order to identify the optimal configuration of the hybrid multigrid solver used as a pure p-multigrid solver applying AMG on the coarsest level. The solver converged for all considered refinement levels and polynomial orders. The difference in the iteration numbers between h-MG and p-MG was small for both CG and DG. For low-order problems ($k < 5$), h-MG was faster; whereas p-MG required fewer iterations for high-order problems due to its better order independence, raising its throughput. CG outperformed DG by a factor of two in all cases, and p-MG seemed to work generally better for CG than for DG.

We found that the best performing p-multigrid configuration for DG halves the polynomial order on every level and uses embedded AMG V-cycles on the auxiliary linear CG space. This p-multigrid configuration exhibits a weak increase in iteration numbers for higher orders and an order-independent throughput (max. 10MDoF/s for 2D and 4.5MDoF/s for 3D for DG on a single compute node) for the considered order range ($k \leq 9$). An alternative p-coarsening approach frequently used in the literature [12, 13, 90] goes from high-order space directly to first-order auxiliary space. The integration of this approach into the presented hybrid multigrid solver showed that the number of iterations in the case of this approach depends strongly on the order, directly resulting in the increase in the time to solution. Computations of this approach were up to two times slower for high order than those of the approach we identified as optimal.

We presented strong-scaling results for 2D/3D, h-MG/p-MG, and CG/DG, obtained on a maximum of 1680 CPUs. We observed ideal strong scaling of both p- and h-multigrid solvers until a low threshold. Communication latency of AMG became disproportionally dominant in the case of p-MG after that threshold.

Based on the gathered data, we constructed a model for the strong-scaling limit of the developed hybrid multigrid solver, quantifying the cost of each multigrid level. According to this model, the costs of each matrix-free and of each matrix-based level are the same, with matrix-based levels being slightly less expensive than matrix-free levels. Nevertheless, the need to perform multiple embedded V-cycles on the cheaper matrix-based levels results in a poor strong-scaling limit when using AMG as a preconditioner on the coarsest level. Based on the derived strong-scaling limit model, we were able to show the benefit of only performing a single coarse AMG V-cycle at the cost of performing more iterations to improve the strong-scaling limit. In addition, the redistribution and repartitioning of the coarse grid might be beneficial in improving the strong-scaling limit.

In order to quantify the efficiency of the developed hybrid multigrid solver in the context of convection-dominated problems, the boundary-layer problem was considered. For solving 2D versions of this problem, we observed a maximal throughput of 4.3MDoF/s. The throughput was independent of the element Péclet number (as was the iteration count) up to an order-independent threshold, the critical element Péclet number $Pe_e = 1$; whereas for $Pe_e > 1$, a drastic increase in iteration numbers led to a rapid drop in throughput. Possible improvements in performance, via the merging of operator applications, promise a speedup by a factor of up to 2. In future research, the influence of matrix-free implementations of both alternative smoothers and anisotropic multgrid transfer operators on the value of the critical element Péclet number ought to be investigated.

Table 19: Performance comparison of the one-step and the two-step hybrid multigrid algorithm (p-MG+AMG) applied to the 2D DG Poisson problem on non-Cartesian mesh as described in Section 5 (degrees of freedom <4000×4000)

	k	3	4	5	6	7	8	9	10	11	12	13
CG+DG	cycles CG [-]	5	4	5	5	5	4	5	5	5	5	6
	cycles DG [-]	1	1	1	1	1	1	1	1	1	1	1
	fraction of time CG [%]	69.0	66.7	73.2	75.0	74.0	70.3	75.2	76.0	76.1	76.9	80.0
	fraction of time DG [%]	27.7	29.9	23.8	22.3	23.0	26.4	21.8	21.1	21.0	20.3	17.6
	throughput [MDoF/s]	19.2	18.5	16.2	15.1	19.1	19.5	16.4	15.5	15.6	14.7	13.0
DG	cycles [-]	7	6	7	7	8	7	9	8	10	10	10
	throughput [MDoF/s]	9.8	10.1	9.7	9.8	10.7	10.8	8.6	9.3	7.7	7.4	7.6
	speedup [-]	2.0	1.8	1.7	1.5	1.8	1.8	1.9	1.7	2.0	2.0	1.7

Numerical results are also presented for solving the incompressible Navier–Stokes equations, considering the FDA benchmark nozzle problem, which we discretized using a non-trivial coarse-grid mesh consisting of 440 cells. As a temporal discretization, we applied the dual-splitting projection scheme, which required solving the pressure Poisson equation. It was solved using the developed hybrid multigrid solver. Computations were performed using p-multigrid with AMG and h-multigrid. For almost all configurations of refinement levels and degrees, a significant speedup – for moderate refinement levels and degrees a speedup of 2-10 – could be observed in solving the pressure Poisson problem with p-MG.

Using the presented hybrid multigrid solver as a hp-multigrid solver led to a comparable speedup, with hp-MG being faster (by up to a factor of two) for moderate high-order elements and few refinement levels than p-multigrid. The promising observation that hp-multigrid with AMG produced the best results for these configurations, should be taken as a reason to exploit the structure of the given discretization in order to geometrically reduce the size of the coarse-grid problem as much as possible. We believe it is always possible to explicitly create 3-4 multigrid levels (1-2 refinements, 2-3 p-levels) such that the developed hybrid multigrid solver remains applicable also to even more complex geometries.

The time spent on solving the pressure Poisson problem and therefore its fraction of time in the dual-splitting projection scheme was reduced considerably for non-trivial coarse-grid meshes by using the hybrid multigrid solver presented here. The benefits of also using this solver in solving the projection and viscous substeps in the dual-splitting projection scheme should be investigated. A prerequisite for such an investigation would be the extension of the hybrid multigrid solver to vectorial quantities.

Currently, the hybrid multigrid implementation is limited to uniformly refined meshes. Its extension to (dynamically) adaptive meshes with hanging nodes is straightforward and should be done in the near future.

As a final remark, it should be noted that the analyzed Poisson problem can be solved twice as fast with CG discretization than with DG discretization because of the need to perform significantly fewer iterations in the CG case. Based on this observation, one can derive an alternative algorithm for the auxiliary space idea to solve high-order DG problems. Instead of using CG only on the coarse-grid space, one could first solve the high-order DG problem rediscretized with continuous elements and then use that solution as the initial solution for finding the actual solution on the DG space. Such a two-step algorithm can be summarized as follows:

1. construct the right-hand side of the CG problem by restricting the provided right-hand side $\boldsymbol{b}_{DG}$, solve the given problem on the finest mesh with continuous elements and prolongate the result back to the high-order DG space:

$$\hat{\boldsymbol{x}}_{DG} \leftarrow \boldsymbol{P}_c \boldsymbol{A}_{CG}^{-1} \boldsymbol{R}_c \boldsymbol{b}_{DG} \qquad \text{with} \qquad \boldsymbol{R}_c = \boldsymbol{P}_c^T, \tag{75a}$$

2. use $\hat{\boldsymbol{x}}_{DG}$ as initial guess for solving the actual DG problem:

$$\boldsymbol{x}_{DG} \leftarrow \boldsymbol{A}_{DG}^{-1} \boldsymbol{b}_{DG} \qquad \text{with} \qquad \boldsymbol{x}_{DG,0} = \hat{\boldsymbol{x}}_{DG}, \tag{75b}$$

or alternatively:

$$\boldsymbol{x}_{DG} - \hat{\boldsymbol{x}}_{DG} \leftarrow \boldsymbol{A}_{DG}^{-1} \left(\boldsymbol{b}_{DG} - \boldsymbol{A}_{DG} \hat{\boldsymbol{x}}_{DG} \right) \qquad \text{with} \qquad \boldsymbol{x}_{DG,0} - \hat{\boldsymbol{x}}_{DG} = \boldsymbol{0}. \tag{75c}$$

Preliminary results of tests using this two-step algorithm are promising and show that it is sufficient to perform only one DG V-cycle after solving the given problem with CG. Table 19 revisits the example of the 2D DG Poisson problem from

Section 5 and presents the results for this problem solved in the standard, one-step way – as it was done throughout this Master's thesis – as well as in the alternative, two-step way proposed here. It is clearly shown that the additional DG cycle of this two-step approach only introduces an overhead of 20-40% to the CG solution, leading to a speedup of up to two compared to the one-step approach. Variations of the two-step approach might improve the performance of the developed hybrid multigrid solver even more and should be investigated more profoundly in the future, along with its possible limitations.

It should be noted that the two-step algorithm uses all multigrid components developed, analyzed, and presented in this Master's thesis only in a slightly different order. All findings of this thesis therefore remain valid irrespective of whether a one-step or a two-step hybrid multigrid solver is applied.

References

[1] G Alzetta, D Arndt, W Bangerth, V Boddu, B Brands, D Davydov, R Gassmoeller, T Heister, L Heltai, K Kormann, M Kronbichler, M Maier, J.-P. Pelteret, B Turcksin, and D Wells. The deal.II Library, Version 9.0. *Journal of Numerical Mathematics*, accepted 2018.

[2] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D Marini. Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems. *SIAM Journal on Numerical Analysis*, 39(5):1749–1779, 2002.

[3] Harold Atkins and Brian Helenbrook. Numerical Evaluation of P-Multigrid Method for the Solution of Discontinuous Galerkin Discretizations of Diffusive Equations. In *17th AIAA Computational Fluid Dynamics Conference*, pages 1–11, Toronto, Ontario Canada, 2005. 17th AIAA Computational Fluid Dynamics Conference; 6-9 Jun. 2005, American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-053-6. doi: 10.2514/6.2005-5110.

[4] Michael Bader. *Space-Filling Curves : An Introduction with Applications in Scientific Computing*, volume 9 of *Texts in Computational Science and Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, Springer edition, 2013. ISBN 978-3-642-31045-4. doi: 10.1007/978-3-642-31046-1.

[5] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries. In Erlend Arge, editor, *Modern Software Tools for Scientific Computing*, pages 163–202. Birkhäuser, Boston, 1997. ISBN 978-1-4612-1986-6 (online); 978-1-4612-7368-4 (print). doi: https://doi.org/10.1007/978-1-4612-1986-6_8.

[6] Satish Balay, Shrirang Abhyankar, Mark F. Adams, and Jed Brown. PETSc Users Manual Revision 3.10, Mathematics and Computer Science Division. Technical report, Argonne National Laboratory (ANL), Argonne, IL (United States), jun 2018. URL `https://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf`.

[7] Wolfgang Bangerth, Carsten Burstedde, Timo Heister, and Martin Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transactions on Mathematical Software*, 38(2): 1–28, dec 2011. ISSN 00983500. doi: 10.1145/2049673.2049678.

[8] F Bassi and S Rebay. A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier-Stokes Equations. *Journal of Computational Physics*, 131(2):267–279, 1997.

[9] F. Bassi, A. Ghidoni, S. Rebay, and P. Tesini. High-order accurate p-multigrid discontinuous Galerkin solution of the Euler equations. *International Journal for Numerical Methods in Fluids*, 60(8):847–865, jul 2009. ISSN 02712091. doi: 10.1002/fld.1917.

[10] Francesco Bassi and Stefano Rebay. Numerical Solution of the Euler Equations with a Multiorder Discontinuous Finite Element Method. In S.W. Armfield, editor, *Computational Fluid Dynamics 2002*, pages 199–204. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-642-59334-5 (online); 978-3-642-63938-8 (print). doi: 10.1007/978-3-642-59334-5_27.

[11] Francesco Bassi, Stefano Rebay, G. Mariotti, S. Pedinotti, and M. Savini. A high order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows. In R. Decuypere and G. Dibelius, editors, *2nd European Conference on Turbomachinery - Fluid Dynamics and Thermodynamics : proceedings*, pages 99–108. Technologische Instituut, Antwerpen, 1997. ISBN 90-5204-032-X.

[12] Peter Bastian, Markus Blatt, and Robert Scheichl. Algebraic multigrid for discontinuous Galerkin discretizations of heterogeneous elliptic problems. *Numerical Linear Algebra with Applications*, 19(2):367–388, 2012. ISSN 10705325. doi: 10.1002/nla.1816.

[13] Peter Bastian, Eike Hermann Müller, Steffen Müthing, and Marian Piatkowski. Matrix-free multigrid block-preconditioners for higher order Discontinuous Galerkin discretisations. *arXiv:1805.11930*, 2018.

[14] Dietrich Braess. *Finite Elemente*. Springer, Berlin, Heidelberg, 2013. ISBN 978-3-642-34796-2. doi: 10.1007/978-3-642-34797-9.

[15] F. Brezzi, M. Manzini, D. Marini, P. Pietra, and A. Russo. Discontinuous finite elements for diffusion problems, 1999. URL `http://arturo.imati.cnr.it/brezzi/papers/lombardo.pdf`.

[16] Jed Brown. Efficient nonlinear solvers for nodal high-order finite elements in 3D. *Journal of Scientific Computing*, 45(1-3):48–63, 2010. ISSN 08857474. doi: 10.1007/s10915-010-9396-8.

[17] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. p4est : Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, jan 2011. ISSN 1064-8275. doi: 10.1137/100791634.

[18] Bernardo Cockburn and Chi-Wang Shu. The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems. *SIAM Journal on Numerical Analysis*, 35(6):2440–2463, 1998.

[19] David Darmofal and Krzysztof Fidkowski. Development of a Higher-Order Solver for Aerodynamic Applications. In *42nd AIAA Aerospace Sciences Meeting and Exhibit*, number January, pages 1–12, Reston, Virigina, jan 2004. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-078-9. doi: 10.2514/6.2004-436.

[20] Veselin A. Dobrev, Raytcho D. Lazarov, Panayot S. Vassilevski, and Ludmil T. Zikatanov. Two-level preconditioning of discontinuous Galerkin approximations of second-order elliptic equations. *Numerical Linear Algebra with Applications*, 13(9):753–770, nov 2006. ISSN 10705325. doi: 10.1002/nla.504.

[21] Veselin A. Dobrev, Raytcho D. Lazarov, and Ludmil T. Zikatanov. Preconditioning of Symmetric Interior Penalty Discontinuous Galerkin FEM for Elliptic Problems. In Ulrich Langer, editor, *Domain Decomposition Methods in Science and Engineering XVII. Lecture Notes in Computational Science and Engineering*, volume 60, pages 33–44. Springer, Berlin, Heidelberg, 2008. ISBN 9783540751984. doi: 10.1007/978-3-540-75199-1_3.

[22] Jim Douglas and Todd Dupont. Interior Penalty Procedures for Elliptic and Parabolic Galerkin Methods. In R. Glowinski, editor, *Computing Methods in Applied Sciences. Lecture Notes in Physics*, number 58, pages 207–216. Springer, Berlin, Heidelberg, 1976. ISBN 978-3-540-08003-9. doi: 10.1007/BFb0120591.

[23] Niklas Fehn. *A Discontinuous Galerkin Approach for the Unsteady Incompressible Navier–Stokes Equations*. Masterarbeit, Technische Universität München, 2015.

[24] Niklas Fehn, Wolfgang A. Wall, and Martin Kronbichler. On the stability of projection methods for the incompressible Navier–Stokes equations based on high-order discontinuous Galerkin discretizations. *Journal of Computational Physics*, 351(15 December):392–421, dec 2017. ISSN 00219991. doi: 10.1016/j.jcp.2017.09.031.

[25] Niklas Fehn, Wolfgang A Wall, and Martin Kronbichler. Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. *International Journal for Numerical Methods in Fluids*, 88(1):32–54, sep 2018. ISSN 02712091. doi: org.eaccess.ub.tum.de/10.1002/fld.4511.

[26] Niklas Fehn, Wolfgang A Wall, and Martin Kronbichler. Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows. *Journal of Computational Physics*, 372(1 November):667–693, nov 2018. ISSN 00219991. doi: 10.1016/j.jcp.2018.06.037.

[27] Niklas Fehn, Wolfgang A. Wall, and Martin Kronbichler. A matrix-free high-order discontinuous Galerkin compressible Navier-Stokes solver: A performance comparison of compressible and incompressible formulations for turbulent incompressible flows. *arXiv:1806.03095*, jun 2018. doi: 10.1002/ĭňĆd.4683.

[28] Niklas Fehn, Wolfgang A Wall, and Martin Kronbichler. Modern discontinuous Galerkin methods for the simulation of transitional and turbulent flows in biomedical engineering : A comprehensive LES study of the FDA benchmark nozzle model. *Manuskript*, 2018.

[29] Krzysztof J. Fidkowski, Todd A. Oliver, James Lu, and David L. Darmofal. p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 207(1):92–113, 2005. ISSN 00219991. doi: 10.1016/j.jcp.2005.01.005.

[30] Paul F. Fischer and James W. Lottes. Hybrid Schwarz-Multigrid Methods for the Spectral Element Method: Extensions to Navier-Stokes. In Ralf Kornhuber, Ronald W. Hoppe, Jacques Périaux, Olivier Pironneau, Olof Widlund, and Jinchao Xu, editors, *Domain Decomposition Methods in Science and Engineering*, pages 35–49. Springer-Verlag, Berlin/Heidelberg, 2005. ISBN 978-3-540-22523-2. doi: 10.1007/3-540-26825-1_3.

[31] Michael W Gee, Christopher M Siefert, Jonathan J Hu, Ray S Tuminaro, and Marzio G Sala. ML 5.0 Smoothed Aggregation User's Guide. Technical report, 2006. URL `https://trilinos.org/oldsite/packages/ml/mlguide5.pdf`.

[32] A. Ghidoni, A. Colombo, F. Bassi, and S. Rebay. Efficient p -multigrid discontinuous Galerkin solver for complex viscous flows on stretched grids. *International Journal for Numerical Methods in Fluids*, 75(2):134–154, may 2014. ISSN 02712091. doi: 10.1002/fld.3888.

[33] Amir Gholami, Dhairya Malhotra, Hari Sundar, and George Biros. FFT, FMM, or Multigrid? A comparative Study of State-Of-the-Art Poisson Solvers for Uniform and Nonuniform Grids in the Unit Cube. *SIAM Journal on Scientific Computing*, 38(3):C280–C306, jan 2016. ISSN 1064-8275. doi: 10.1137/15M1010798.

[34] P. M. Gresho and R. L. Sani. *Incompressible Flow and the Finite Element Method, Volume 1: Advection - Diffusion*. Wiley, Chichester, 2000. ISBN 978-0-471-49249-8.

[35] X.-Z Guo and I.N. Katz. Performance Enhancement of the Multi-p Preconditioner. *Computers & Mathematics with Applications*, 36(4):1–8, 1998.

[36] Xian-Zhong Guo and I. N. Katz. A Parallel Multi-p Method. *Computers & Mathematics with Applications*, 39 (9-10):115–123, 2000.

[37] Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*, volume 20102232 of *Chapman & Hall/CRC Computational Science*. CRC Press, Boca Raton, jul 2010. ISBN 978-1-4398-1192-4. doi: 10.1201/EBK1439811924.

[38] R.E. Heath, I.M. Gamba, P.J. Morrison, and C. Michler. A discontinuous Galerkin method for the Vlasov–Poisson system. *Journal of Computational Physics*, 231(4):1140–1174, feb 2012. ISSN 00219991. doi: 10.1016/j.jcp.2011.09.020.

[39] Brian Helenbrook, Dimitri Mavriplis, and Harold Atkins. Analysis of "p"-Multigrid for Continuous and Discontinuous Finite Element Discretizations. In *16th AIAA Computational Fluid Dynamics Conference*, Orlando, Florida, jun 2003. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-086-4. doi: 10.2514/6.2003-3989.

[40] Brian T Helenbrook and H L Atkins. Application of p-Multigrid to Discontinuous Galerkin Formulations of the Poisson Equation. *AIAA Journal*, 44(3):566–575, 2006. doi: 10.2514/1.15497.

[41] Brian T Helenbrook and H L Atkins. Solving Discontinuous Galerkin Formulations of Poisson's Equation using Geometric and p Multigrid. *AIAA Journal*, 46(4):894–902, apr 2008. ISSN 0001-1452. doi: 10.2514/1.31163.

[42] Brian T Helenbrook and Brendan S Mascarenhas. Analysis of Implicit Time-Advancing p-Multigrid Schemes for Discontinuous Galerkin Discretizations of the Euler Equations. In *46th AIAA Fluid Dynamics Conference*, Washington, D.C., 2016. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-436-7. doi: 10.2514/6.2016-3494.

[43] John L. Hennessy and David A. Patterson. *Computer Architecture : A Quantitative Approach*. Elsevier, Morgan Kaufmann, Amsterdam, 5. ed. edition, 2012. ISBN 9780123838728. doi: 10.1.1.115.1881.

[44] Van Emden Henson and Ulrike Meier Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, 2002. ISSN 01689274. doi: 10.1016/S0168-9274(01)00115-5.

[45] Michael A Heroux, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M Willenbring, Alan Williams, Kendall S. Stanley, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, and Roger P. Pawlowski. An overview of the Trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, sep 2005. ISSN 00983500. doi: 10.1145/1089014.1089021.

[46] Koen Hillewaert. *Development of the Discontinuous Galerkin Method for High-resolution, Large Scale CFD and Acoustics in Industrial Geometries*. Presses universitaires de Louvain, Louvain, 2013. ISBN 9782875581198.

[47] Koen Hillewaert, P Wesseling, E Oñate, J Périaux, Jean-François Remacle, Nicolas Cheveaugeon, Paul-Emile Bernard, and Philippe Geuzaine. Analysis of a hybrid p-multigrid method for the discontinuous Galerkin discretisation of the Euler equations. In Pieter Wesseling, editor, *Proceedings of the European Conference on Computational Fluid Dynamics*, Egmond aan Zee, Netherlands, 2006. ECCOMAS CFD 2006. doi: 90-9020970-0.

[48] Ning Hu and Norman Katz. Multi-P Methods: Iterative Algorithms for the P-Version of the Finite Element Analysis. *SIAM Journal on Scientific Computing*, 16(6):1308–1332, 1995.

[49] Ning Hu, Xian-Zhong Guo, and I Norman Katz. Multi-p Preconditioners. *SIAM Journal on Scientific Computing*, 18(6):1676–1697, 1997.

[50] Immo Huismann, Jörg Stiller, and Jochen Fröhlich. Scaling to the stars - a linearly scaling elliptic solver for p-multigrid. *arXiv:1808.03595*, 2018.

[51] Zhenhua Jiang, Chao Yan, Jian Yu, and Wu Yuan. Practical aspects of p-multigrid discontinuous Galerkin solver for steady and unsteady RANS simulations. *International Journal for Numerical Methods in Fluids*, 78(11): 670–690, 2015. ISSN 10970363. doi: 10.1002/fld.4035.

[52] George Em Karniadakis, Moshe Israeli, and Steven A Orszag. High-order splitting methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 97(2):414–443, dec 1991. ISSN 00219991. doi: 10.1016/0021-9991(91)90007-8.

[53] Benjamin Krank, Niklas Fehn, Wolfgang A Wall, and Martin Kronbichler. A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow. *Journal of Computational Physics*, 348(1 November):634–659, nov 2017. ISSN 00219991. doi: 10.1016/j.jcp.2017.07.039.

[54] Rolf Krause and Patrick Zulian. A Parallel Approach to the Variational Transfer of Discrete Fields between Arbitrarily Distributed Unstructured Finite Element Meshes. *SIAM Journal on Scientific Computing*, 38(3): C307–C333, jan 2016. ISSN 1064-8275. doi: 10.1137/15M1008361.

[55] M. Kronbichler, S. Schoeder, C. Müller, and W. A. Wall. Comparison of implicit and explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation. *International Journal for Numerical Methods in Engineering*, 106(9):712–739, jun 2016. ISSN 00295981. doi: 10.1002/nme.5137.

[56] Martin Kronbichler and Momme Allalen. Efficient high-order discontinuous Galerkin finite elements with matrix-free implementations. *Manuskript*, 2018.

[57] Martin Kronbichler and Katharina Kormann. A generic interface for parallel cell-based finite element operator application. *Computers and Fluids*, 63:135–147, 2012. ISSN 00457930. doi: 10.1016/j.compfluid.2012.04.012.

[58] Martin Kronbichler and Katharina Kormann. Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *arXiv:1711.03590*, 2017.

[59] Martin Kronbichler and Wolfgang A Wall. A Performance Comparison of Continuous and Discontinuous Galerkin Methods with Fast Multigrid Solvers. *SIAM Journal on Scientific Computing*, 40(5):A3423–A3448, jan 2018. ISSN 1064-8275. doi: 10.1137/16M110455X.

[60] Qi Li, Kazumasa Ito, Zhishen Wu, Christopher S. Lowry, and Steven P. Loheide. COMSOL multiphysics: A novel approach to ground water modeling. *Ground Water*, 47(4):480–487, 2009. ISSN 0017467X. doi: 10.1111/j.1745-6584.2009.00584.x.

[61] C. Liang, R. Kannan, and Z.J. Wang. A p-multigrid spectral difference method with explicit and implicit smoothers on unstructured triangular grids. *Computers & Fluids*, 38(2):254–265, feb 2009. ISSN 00457930. doi: 10.1016/j.compfluid.2008.02.004.

[62] James W Lottes and Paul F Fischer. Hybrid Multigrid/Schwarz Algorithms for the Spectral Element Method. *Journal of Scientific Computing*, 24(1):45–78, jul 2005. ISSN 0885-7474. doi: 10.1007/s10915-004-4787-3.

[63] Cao Lu, Xiangmin Jiao, and Nikolaos Missirlis. A Hybrid Geometric+Algebraic Multigrid Method with Semi-Iterative Smoothers. *Numerical Linear Algebra with Applications*, 21(2):221–238, 2014. ISSN 10991506. doi: 10.1002/nla.1925.

[64] Hong Luo, Joseph D Baum, and Rainald Löhner. A fast, p-Multigrid Discontinuous Galerkin Method for Compressible Flows at All Speeds. Technical report, 44th AIAA Aerospace Sciences Meeting and Exhibit, 9–12, January 2006, Reno, Nevada, 2006.

[65] Hong Luo, Joseph D. Baum, and Rainald Löhner. A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids. *Journal of Computational Physics*, 211(2):767–783, 2006. ISSN 00219991. doi: 10.1016/j.jcp.2005.06.019.

[66] Yvon Maday and Rafael Munoz. Spectral element multigrid. II. Theoretical justification. *Journal of Scientific Computing*, 3(4):323–353, dec 1988. ISSN 0885-7474. doi: 10.1007/BF01065177.

[67] Richard A. Malinauskas, Prasanna Hariharan, Steven W. Day, Luke H. Herbertson, Martin Buesen, Ulrich Steinseifer, Kenneth I. Aycock, Bryan C. Good, Steven Deutsch, Keefe B. Manning, and Brent A. Craven. FDA Benchmark Medical Device Flow Models for CFD Validation. *ASAIO Journal*, 63(2):150–160, 2017. ISSN 1058-2916. doi: 10.1097/MAT.0000000000000499.

[68] Brendan S. Mascarenhas, Brian T. Helenbrook, and Harold L. Atkins. Application of p-Multigrid to Discontinuous Galerkin Formulations of the Euler Equations. *AIAA Journal*, 47(5):1200–1208, may 2009. ISSN 0001-1452. doi: 10.2514/1.39765.

[69] Brendan S. Mascarenhas, Brian T. Helenbrook, and Harold L. Atkins. Coupling p-multigrid to geometric multigrid for discontinuous Galerkin formulations of the convection-diffusion equation. *Journal of Computational Physics*, 229(10):3664–3674, 2010. ISSN 10902716. doi: 10.1016/j.jcp.2010.01.020.

[70] John D. McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers, 2018. URL `https://www.cs.virginia.edu/stream/`.

[71] J.M. Melenk, K Gerdes, and Ch. Schwab. Fully discrete hp-finite elements: fast quadrature. *Computer Methods in Applied Mechanics and Engineering*, 190(32-33):4339–4364, may 2001. ISSN 00457825. doi: 10.1016/S0045-7825(00)00322-4.

[72] William F. Mitchell. The hp-multigrid method applied to hp-adaptive refinement of triangular grids. *Numerical Linear Algebra with Applications*, 17(2-3):211–228, 2010. ISSN 10705325. doi: 10.1002/nla.700.

[73] Steffen Müthing, Marian Piatkowski, and Peter Bastian. High-performance Implementation of Matrix-free High-order Discontinuous Galerkin Methods. *arXiv:1711.10885*, nov 2017.

[74] Artem Napov and Yvan Notay. An Algebraic Multigrid Method with Guaranteed Convergence Rate. *SIAM Journal on Scientific Computing*, 34(2):A1079–A1109, 2012. ISSN 1064-8275. doi: 10.1137/100818509.

[75] Cristian R. Nastase and Dimitri J. Mavriplis. High-order discontinuous Galerkin methods using an hp-multigrid approach. *Journal of Computational Physics*, 213(1):330–357, 2006. ISSN 00219991. doi: 10.1016/j.jcp.2005.08.022.

[76] Yvan Notay. An aggregation-based algebraic multigrid method. *Electronic Transactions on Numerical Analysis*, 37:123–146, 2010. ISSN 1068-9613.

[77] Yvan Notay. Aggregation-Based Algebraic Multigrid for Convection-Diffusion Equations. *SIAM Journal on Scientific Computing*, 34(4):A2288–A2316, 2012. ISSN 1064-8275. doi: 10.1137/110835347.

[78] Yvan Notay. User ' s Guide to AGMG. Technical report, 2018. URL `http://agmg.eu/agmg{_}userguide.pdf`.

[79] B. O'Malley, J. Kópházi, R. P. Smedley-Stevenson, and M. D. Eaton. P-multigrid expansion of hybrid multilevel solvers for discontinuous Galerkin finite element discrete ordinate (DG-FEM-SN) diffusion synthetic acceleration (DSA) of radiation transport algorithms. *Progress in Nuclear Energy*, 98:177–186, 2017. ISSN 01491970. doi: 10.1016/j.pnucene.2017.03.014.

[80] B. O'Malley, J. Kópházi, R. P. Smedley-Stevenson, and M. D. Eaton. Hybrid Multi-level solvers for discontinuous Galerkin finite element discrete ordinate diffusion synthetic acceleration of radiation transport algorithms. *Annals of Nuclear Energy*, 102(April):134–147, 2017. ISSN 0306-4549. doi: doi.org/10.1016/j.anucene.2016.11.048.

[81] Steven A. Orszag. Spectral methods for problems in complex geometries. *Journal of Computational Physics*, 37 (1):70–92, aug 1980. ISSN 00219991. doi: 10.1016/0021-9991(80)90005-4.

[82] Will Pazner and Per-Olof Persson. Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods. *Journal of Computational Physics*, 354(1 February):344–369, feb 2018. ISSN 00219991. doi: 10.1016/j.jcp.2017.10.030.

[83] Sachin Premasuthan, Chunlei Liang, Antony Jameson, and Zhi Wang. A p-Multigrid Spectral Difference Method For Viscous Compressible Flow Using 2D Quadrilateral Meshes. In *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, Orlando, Florida, 2009. American Institute of Aeronautics and Astronautics. ISBN 978-1-60086-973-0. doi: 10.2514/6.2009-950.

[84] Glyn Owen Rees. *Efficient " Black-Box " Multigrid Solvers for Convection-Dominated Problems*. Promotionsarbeit, University of Manchester, 2011.

[85] Thomas Roehl, Jan Treibig, Georg Hager, and Gerhard Wellein. Overhead Analysis of Performance Counter Measurements. In *2014 43rd International Conference on Parallel Processing Workshops*, volume 2015-May, pages 176–185, Minneapolis, Minnesota, USA, sep 2014. IEEE. ISBN 978-1-4799-5615-9. doi: 10.1109/ICPPW.2014.34.

[86] Einar M. Rønquist and Anthony T. Patera. Spectral element multigrid. I. Formulation and numerical results. *Journal of Scientific Computing*, 2(4):389–406, 1987. ISSN 08857474. doi: 10.1007/BF01061297.

[87] Johann Rudi, Omar Ghattas, A. Cristiano I. Malossi, Tobin Isaac, Georg Stadler, Michael Gurnis, Peter W. J. Staar, Yves Ineichen, Costas Bekas, and Alessandro Curioni. An extreme-scale implicit solver for complex PDEs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*, pages 1–12, New York, USA, 2015. ACM Press. ISBN 9781450337236. doi: 10.1145/2807591.2807675.

[88] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edition, 2003.

[89] Khosro Shahbazi, Dimitri J. Mavriplis, and Nicholas K. Burgess. Multigrid algorithms for high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 228(21): 7917–7940, 2009. ISSN 00219991. doi: 10.1016/j.jcp.2009.07.013.

[90] C. Siefert, R. Tuminaro, A. Gerstenberger, G. Scovazzi, and S. S. Collis. Algebraic multigrid techniques for discontinuous Galerkin methods with varying polynomial order. *Computational Geosciences*, 18(5):597–612, oct 2014. ISSN 1420-0597. doi: 10.1007/s10596-014-9419-x.

[91] Jörg Stiller. Robust multigrid for high-order discontinuous Galerkin methods: A fast Poisson solver suitable for high-aspect ratio Cartesian grids. *arXiv:1603.02524*, mar 2016. doi: 10.1016/j.jcp.2016.09.041.

[92] Jörg Stiller. Robust Multigrid for Cartesian Interior Penalty DG Formulations of the Poisson Equation in 3D. In M. Bittencourt, editor, *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2016. Lecture Notes in Computational Science and Engineering*, volume 119, pages 189–201. Springer, Cham, 2017. ISBN 978-3-319-65869-8 (print); 978-3-319-65870-4 (online). doi: 10.1007/978-3-319-65870-4_12.

[93] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1-2): 281–309, 2001. ISSN 03770427. doi: 10.1016/S0377-0427(00)00516-1.

[94] Hari Sundar, George Biros, Carsten Burstedde, Johann Rudi, Omar Ghattas, and Georg Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Salt Lake City, UT, 2012. International Conference for High Performance Computing, Networking, Storage and Analysis, SC. ISBN 9781467308069. doi: 10.1109/SC.2012.91.

[95] Hari Sundar, Georg Stadler, and George Biros. Comparison of multigrid algorithms for high-order continuous finite element discretizations. *Numerical Linear Algebra with Applications*, 22(4):664–680, 2015. ISSN 10991506. doi: 10.1002/nla.1979.

[96] Jan Treibig, Georg Hager, and Gerhard Wellein. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In Wang-Chien Lee, editor, *2010 39th International Conference on Parallel Processing Workshops*, pages 207–216, Piscataway, NJ, sep 2010. IEEE. ISBN 978-1-4244-7918-4. doi: 10.1109/ICPPW.2010.38.

[97] P. van Slingerland and C. Vuik. Scalable two-level preconditioning and deflation based on a piecewise constant subspace for (SIP)DG systems for diffusion problems. *Journal of Computational and Applied Mathematics*, 275: 61–78, feb 2015. ISSN 03770427. doi: 10.1016/j.cam.2014.06.028.

[98] Richard S. Varga. *Matrix Iterative Analysis*, volume 27 of *Springer Series in Computational Mathematics*. Springer, Berlin, Heidelberg, 2000. ISBN 978-3-642-05154-8. doi: 10.1007/978-3-642-05156-2.

[99] Samuel Williams, Andrew Waterman, and David Patterson. Roofline : An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65, apr 2009. ISSN 00010782. doi: 10.1145/1498765.1498785.

Appendices

A Thread and cache topology

The following informations of the CPUs used for all simulations have been extracted with the program `likwid-topology` from the LIKWID suite [85, 96].

```
CPU name:       Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz
CPU type:       Intel Xeon Haswell EN/EP/EX processor
CPU stepping:   2
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                2
Cores per socket:       14
Threads per core:       1
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
3               0               3               0               *
4               0               4               0               *
5               0               5               0               *
6               0               6               0               *
7               0               7               0               *
8               0               8               0               *
9               0               9               0               *
10              0               10              0               *
11              0               11              0               *
12              0               12              0               *
13              0               13              0               *
14              0               0               1               *
15              0               1               1               *
16              0               2               1               *
17              0               3               1               *
18              0               4               1               *
19              0               5               1               *
20              0               6               1               *
21              0               7               1               *
22              0               8               1               *
23              0               9               1               *
24              0               10              1               *
25              0               11              1               *
26              0               12              1               *
27              0               13              1               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 1 2 3 4 5 6 7 8 9 10 11 12 13 )
Socket 1:               ( 14 15 16 17 18 19 20 21 22 23 24 25 26 27 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 )
                        ( 8 ) ( 9 ) ( 10 ) ( 11 ) ( 12 ) ( 13 ) ( 14 ) ( 15 )
                        ( 16 ) ( 17 ) ( 18 ) ( 19 ) ( 20 ) ( 21 ) ( 22 ) ( 23 )
                        ( 24 ) ( 25 ) ( 26 ) ( 27 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   256 kB
Cache groups:           ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 )
                        ( 8 ) ( 9 ) ( 10 ) ( 11 ) ( 12 ) ( 13 ) ( 14 ) ( 15 )
                        ( 16 ) ( 17 ) ( 18 ) ( 19 ) ( 20 ) ( 21 ) ( 22 ) ( 23 )
                        ( 24 ) ( 25 ) ( 26 ) ( 27 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   18 MB
Cache groups:           ( 0 1 2 3 4 5 6 ) ( 7 8 9 10 11 12 13 )
                        ( 14 15 16 17 18 19 20 ) ( 21 22 23 24 25 26 27 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           4
```

```
Domain:                 0
Processors:             (  0  1  2  3  4  5  6 )
Distances:              10 11 21 21
Free memory:            13221.3 MB
Total memory:           15677.5 MB
--------------------------------------------------------------------------------
Domain:                 1
Processors:             (  7  8  9 10 11 12 13 )
Distances:              11 10 21 21
Free memory:            14434.3 MB
Total memory:           16384 MB
--------------------------------------------------------------------------------
Domain:                 2
Processors:             ( 14 15 16 17 18 19 20 )
Distances:              21 21 10 11
Free memory:            15005.6 MB
Total memory:           16384 MB
--------------------------------------------------------------------------------
Domain:                 3
Processors:             ( 21 22 23 24 25 26 27 )
Distances:              21 21 11 10
Free memory:            15739.1 MB
Total memory:           16384 MB
--------------------------------------------------------------------------------
```