Title

Agglomeration-based geometric multigrid solvers for compact discontinuous Galerkin discretizations on unstructured meshes

Authors

Pan, Y

Persson, P-O

Publication Date

2022

DOI

10.1016/j.jcp.2021.110775

Agglomeration-Based Geometric Multigrid Solvers for Compact Discontinuous Galerkin Discretizations on Unstructured Meshes

Y. Pan[a,b,1,*], P.-O. Persson[a,b,2]

[a]*Department of Mathematics, University of California, Berkeley, Berkeley, CA 94720, United States*

[b]*Mathematics Group, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, United States*

Abstract

We present a geometric multigrid solver for the Compact Discontinuous Galerkin method through building a hierarchy of coarser meshes using a simple agglomeration method which handles arbitrary element shapes and dimensions. The method is easily extendable to other discontinuous Galerkin discretizations, including the Local DG method and the Interior Penalty method. We demonstrate excellent solver performance for Poisson's equation, provided a flux formulation is used for the operator coarsening and a suitable switch function chosen for the numerical fluxes.

Keywords: discontinuous Galerkin, agglomeration, geometric multigrid

1. Introduction

The discontinuous Galerkin (DG) method with high-order approximations are becoming increasingly popular for the solution of systems of conservation laws, due to their natural ability to stabilize convection-dominated problem on

*Corresponding author

Email addresses: yllpan@berkeley.edu (Y. Pan), persson@berkeley.edu (P.-O. Persson)

[1]Graduate student, Department of Mathematics, University of California, Berkeley

[2]Professor, Department of Mathematics, University of California, Berkeley

arbitrary unstructured meshes with high-order accuracy. The resulting semi-discrete systems are often integrated in time using explicit solvers, however, for many real-world problems it is widely believed that implicit solvers will be required. This poses many challenges, since the Jacobian matrices are expensive to compute and store, and specialized solvers are required to solve the corresponding linear systems that arise.

One of the most important solver techniques employed, at least for elliptic or diffusion-dominated problems, is the multigrid method [3]. The method has been used extensively for DG methods [9, 17, 21, 15, 10, 19], where it can naturally be applied as a p-multigrid solver where the grid hierarchy is formed by varying the polynomial degrees in each element. It can also be used in the more traditional h-multigrid setting, where the hierarchy is based on meshes of varying coarseness, or as a combined hp-multigrid method which combines both these techniques [14]. For fully unstructured meshes, it is in general difficult to coarsen a given mesh in order to produce the mesh hierarchies needed for a full h-multigrid. This is one of the motivations for using so-called Algebraic Multigrid methods [23, 11, 1].

An alternative approach for coarsening an unstructured mesh is agglomeration, that is, merging neighboring elements into larger ones successively. The technique is not widely used for continuous Galerkin finite element methods, because of the difficulties in defining continuous approximation spaces on the resulting polyhedral elements. However, the technique has been used successfully for finite volume methods [12, 5, 20], where it is easier to update the element-averages after coarsening. This is also true for high-order discontinuous Galerkin methods, since they are straight-forward to implement on meshes of arbitrarily shaped elements [8, 7].

In this paper, we propose an agglomeration-based h-multigrid method for

Poisson's equation based on the CDG method [16]. This is a variant of the LDG method [6], with important benefits such as element-wise compact stencils and improved stability properties. However, our method should be straight-forward to use with the LDG method, or any other discretization such as Interior Penalty or the BR2 methods [2].

We perform the element agglomeration with a simple approach which extends to arbitrary elements and dimensions. While the resulting hierarchy might not be optimal for the multigrid performance, our numerical experiments demonstrate that the method is quite insensitive to the shape of the agglomerated elements. We also show the importance of choosing a good switch function for the numerical fluxes in the CDG method.

The paper is organized as follows. In Section 2, we describe the CDG discretization and in particular write it in the so-called flux formulation which is needed for the operator coarsening in the multigrid method. Next, we outline the (heuristic) geometric element agglomeration algorithm in Section 3, and the details of the multigrid method in Section 4. Our numerical results in Section 5 show a number of important properties of our scheme, and demonstrate its performance.

2. Discontinuous Galerkin formulation

2.1. Problem definition

For our notation, quantities that have a spatial dimension, such as the spatial gradient of a function, are bolded whilst scalar functions are not. We consider

here Poisson's equation as our model elliptic problem

$$\begin{aligned} \nabla^2 u &= f \quad && \text{in } \Omega, \\ u &= g_D && \text{on } \Gamma_D, \\ \nabla u \cdot \boldsymbol{n} &= g_N && \text{on } \Gamma_N, \end{aligned} \tag{1}$$

in a domain $\Omega \subset \mathbb{R}^d$, where $d \in \{1, 2, 3\}$ is the dimension of the system. Γ_D, Γ_N respectively denote parts of the boundary $\partial\Omega$ on which Dirichlet and Neumann boundary conditions are imposed, with $\boldsymbol{n}$ denoting the unit outward normal on $\partial\Omega$. Here, $f(\boldsymbol{x})$ is an arbitrary given function in $L^2(\Omega)$ and we further assume that the length of Ω_D is strictly greater than zero.

2.2. DG formulation for elliptic problems

To apply a DG method to the above model problem, we rewrite Equation 1 as a first order system of equations by introducing the variable $\boldsymbol{q} = \nabla u$ and rewriting the Laplacian operator as the divergence of $\boldsymbol{q}$,

$$\begin{aligned} \nabla \cdot \boldsymbol{q} &= f \quad && \text{in } \Omega, \\ \boldsymbol{q} &= \nabla u && \text{in } \Omega, \\ u &= g_D && \text{on } \Gamma_D, \\ \boldsymbol{q} \cdot \boldsymbol{n} &= g_N && \text{on } \Gamma_N. \end{aligned} \tag{2}$$

In this work, we consider discretizations where meshes $\mathcal{T}_h = \{K\}$ of Ω may consist of arbitrarily shaped elements, with the only restriction being elements must not self intersect. We define the broken spaces $V(\mathcal{T}_h)$ and $\Sigma(\mathcal{T}_h)$ as the union of Sobolev spaces $H^1(K)$ and $[H^1(K)]^d$ restricted to each element K. Specifically,

$$V = \{v \in L^2(\Omega) : v|_K \in H^1(K),\ \forall K \in \mathcal{T}_h\} \tag{3}$$

$$\Sigma = \{\boldsymbol{\tau} \in [L^2(\Omega)]^d : \boldsymbol{\tau}|_K \in [H^1(K)]^d,\ \forall K \in \mathcal{T}_h\} \tag{4}$$

We also introduce the finite element spaces $V_h \subset V$ and $\Sigma_h \subset \Sigma$ as

$$V_h = \{v \in L^2(\Omega) : v|_K \in \mathcal{P}_p(K),\ \forall K \in \mathcal{T}_h\} \tag{5}$$

$$\Sigma_h = \{\boldsymbol{\tau} \in [L^2(\Omega)]^d : \boldsymbol{\tau}|_K \in [\mathcal{P}_p(K)]^d,\ \forall K \in \mathcal{T}_h\} \tag{6}$$

where $\mathcal{P}_p(K)$ denotes the space of polynomial functions of order at most $p \geq 1$ on each element K.

We obtain a weak DG formulation by multiplying the system of equations with test functions $v, \boldsymbol{\tau}$ before integrating by parts. From this our formulation can be expressed as finding $\boldsymbol{u}_h \in V_h, \boldsymbol{q}_h \in \Sigma_h$ such that for all $K \in \mathcal{T}_h = \{K\}$, we have

$$\begin{aligned} \int_K (\boldsymbol{q}_h + u_h \nabla) \cdot \boldsymbol{\tau} dx &= \int_{\partial K} \hat{u} \boldsymbol{\tau} \cdot \boldsymbol{n} ds \quad \forall \boldsymbol{\tau} \in [\mathcal{P}_p(K)]^d, \\ \int_K \boldsymbol{q_h} \cdot \nabla v \ dx &= \int_{\partial K} v \hat{\boldsymbol{q}} \cdot \boldsymbol{n} ds + \int_K f v \ dx \quad \forall v \in \mathcal{P}_p(K). \end{aligned} \tag{7}$$

The numerical fluxes $\hat{u}, \hat{\boldsymbol{q}}$ approximate the quantities to u and to $\boldsymbol{q} = \nabla u$ on the boundaries of each element K. For the CDG method, numerical fluxes are expressed as a function of the fields u_h and $\boldsymbol{q}_h$, in addition to the specified boundary conditions on $\partial\Omega$ as follows.

To specify the numerical fluxes, we define a switch function $S_K^{K'} \in \{-1, 1\}$ on each internal boundary separating element K from its neighbour K', which satisfies the property $S_K^{K'} = -S_{K'}^K$. One example is the natural switch function, where given any enumeration of the elements $\{\mathcal{N}(K)\}$, for any two elements

K, K', the switch $S_K^{K'} > 0$ if $\mathcal{N}(K) > \mathcal{N}(K')$. Given a switch function, the numerical fluxes are defined as:

- In Equation 7, $\hat{u}$ is defined by standard upwinding based on the switch function

$$\hat{u} = \begin{cases} u_h & \text{if } S_K^{K'} > 0 \\ u_h' & \text{if } S_K^{K'} < 0 \end{cases} \tag{8}$$

  where u_h' is the numerical solution to u in Equation 7 on the neighbouring element K' on boundary ∂K.

- On every inter-element boundary f separating two elements $K, \tilde{K}$, where $S_K^{\tilde{K}} < 0$, define a "boundary gradient" $\boldsymbol{q}_h^f$ using a slight modification of Equation 7

$$\int_K (\boldsymbol{q}_h^f + u_h \nabla) \cdot \boldsymbol{\tau} dx = \int_{\partial K \setminus f} u_h \boldsymbol{\tau} \cdot \boldsymbol{n} ds + \int_f \tilde{u}_h \boldsymbol{\tau} \cdot \boldsymbol{n} ds \tag{9}$$

  where tilde on $\tilde{u}_h, \tilde{\boldsymbol{q}}_h$ denotes numerical solutions to the respective fields defined on $\tilde{K}$. The flux $\hat{\boldsymbol{q}}$ on f is then defined simply by restricting $\boldsymbol{q}_h^f$ to the boundary f.

- On a boundary f of element K that coincides with $\partial\Omega$, we similarly define a "boundary gradient"

$$\int_K (\boldsymbol{q}_h^f + u_h \nabla) \cdot \boldsymbol{\tau} dx = \int_{\partial K} u \boldsymbol{\tau} \cdot \boldsymbol{n} ds \tag{10}$$

  The numerical fluxes are defined using the defined "boundary gradient" in addition to the specified boundary conditions,

$$\begin{aligned} \hat{\boldsymbol{q}} &= \boldsymbol{q}_h^f - C_D(u_h - g_D)\boldsymbol{n}, & \hat{u} &= g_D \quad \text{on } \partial\Omega_D \\ \hat{\boldsymbol{q}} &= g_N \boldsymbol{n}, & \hat{u} &= u_h \quad \text{on } \partial\Omega_N \end{aligned} \tag{11}$$

where the parameter $C_D > 0$ is included for additional stabilisation. For our applications, we choose $C_D = \gamma / h_{avg}$, where $\gamma > 0$ is a constant, and h_{avg} is mean height of elements K on the boundary $\partial\Omega_D$. The choice of γ and its effect on multigrid convergence is discussed in Section 5.1.

We briefly note the similarity of the CDG method to the LDG method, with the only distinction being in the definition of fluxes $\hat{\boldsymbol{q}}$. For a more detailed treatment on the CDG method and its properties we turn the reader to [16].

2.3. Discrete formulation

Discretising Equation 7 we obtain a linear system

$$\begin{aligned} M\boldsymbol{q}_h + Gu_h &= \boldsymbol{r} \\ \tilde{D}\boldsymbol{q}_h + N\hat{\boldsymbol{q}} &= f \end{aligned} \tag{12}$$

where M denotes the system mass matrix, G the discrete gradient operator, and $\boldsymbol{r}$ the Dirichlet vector, defined as

$$\begin{aligned} G(u_h) &= \sum_K \left(\int_K u_h \nabla \cdot \boldsymbol{\tau} dx - \int_{\partial K \setminus \partial\Omega_D} \hat{u}\boldsymbol{\tau} \cdot \boldsymbol{n} ds \right) \\ \boldsymbol{r} &= \int_{\partial\Omega_D} g_D \boldsymbol{\tau} \cdot \boldsymbol{n} ds \end{aligned} \tag{13}$$

The operators $\tilde{D}$, N, and the vector f are defined as

$$\begin{aligned} \tilde{D}(\boldsymbol{q}_h) &= \sum_K \int_K \boldsymbol{q}_h \cdot \nabla v dx \\ N(\hat{\boldsymbol{q}}) &= -\sum_K \int_{\partial K \setminus \partial\Omega_N} v\hat{\boldsymbol{q}} \cdot \boldsymbol{n} ds \\ f &= \sum_K \int_K f v dx - \int_{\partial\Omega_N} v g_N ds - \int_{\partial\Omega_D} v C_D g_D ds \end{aligned} \tag{14}$$

Following [16], it is possible to write $\hat{\boldsymbol{q}}$ in terms of the variables u_h and $\boldsymbol{q}_h$ as a consequence of Equation 11. Similarly, following Equation 13, it is possible

to write the variable $\boldsymbol{q}_h$ simply as a variable of the unknown u_h. All together this allows us to write the Poisson system discretely as a single linear system

$$Au_h = b \tag{15}$$

where the matrix A is compact, meaning a block (i, j) has non-zero entries if and only if elements K_i, K_j are immediate neighbours. This is known as the primal form of the CDG method.

For construction of a multigrid solver however, following [10], direct coarsening of the primal operator A can lead to decreased performance of the solver. A flux form of the CDG system can instead be defined as follows

$$\begin{bmatrix} M & G \\ D & C \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_h \\ u_h \end{bmatrix} = \begin{bmatrix} \boldsymbol{r} \\ s \end{bmatrix} \tag{16}$$

where $D = -G^T$ is the discrete divergence operator, and $A = C - DM^{-1}G$, $s = f - DM^{-1}\boldsymbol{r}$. Unlike the matrix A, the matrix C is not compact in that it may contain non-zero entries in a block (i, j) where elements K_i, K_j are not immediate neighbours.

The CDG method is closely related to the LDG method in that the only difference in the flux formulations of the two lie in the bottom right hand entry of the flux operator; for LDG the C matrix is equal to the zero matrix. The C matrix in the CDG method cancels out the non-compact entries from the term $DM^{-1}G$, rendering the resulting matrix A to be compact. This therefore implies that in general it is unnecessary to store the non-compact entries of the matrix C as they may be implicitly inferred.

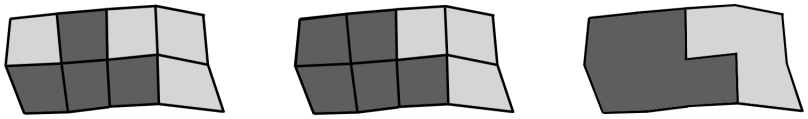


Figure 1: Example mesh partitions. The left partition is invalid as one subset is not connected. The middle partition is valid and agglomerated into the polygonal mesh on the right.

3. Mesh hierarchy

For h-multilevel solvers, a hierarchy of successively coarser mesh discretizations are constructed on which the matrix equation $Ax = b$ is solved approximately on each level of the hierarchy. While for structured meshes coarsening algorithms such as quadtree/octree are widely used to construct mesh hierarchies, it is in general difficult to construct such hierarchies for unstructured meshes.

3.1. Element agglomeration

In this work, we focus on mesh hierarchy construction via agglomeration. A valid mesh agglomerate is defined in this work as a partition of the set of elements $\mathcal{T}_h = \{K\}$ such that the union of elements within each subset of the partition form a connected domain. This choice of hierarchy constructed is based on the observation that in general for a DG formulation, the lack of C^0 continuity required in standard continuous finite elements allows for the easy definition of modal basis functions on arbitrarily shaped polyhedra.

To define a mesh hierarchy, elements within each partition are agglomerated to form a single polygonal element, which are then all collected as the set of elements for the next level of the mesh hierarchy. This process can be performed recursively until the final level of the hierarchy contains only a single polygonal element defined by the boundaries of the computational domain Ω.

Algorithm 1: Recursive mesh agglomeration

Input mesh with elements E_0

Hierarchy storage $E = \{E_0\}$

$E_{level} = E_0$

while $length(E_{level}) > 1$ **do**

$E_{level+1}$ = Find set partition of E_{level}

Append $E_{level+1}$ to E

$E_{level} = E_{level+1}$

end while

return E

The problem of finding mesh partitions is well studied in literature, including popular domain decomposition methods in the software package METIS [13]. In this work however we use a simple greedy heuristic to demonstrate the generality of the method for mesh partitions of arbitrary shape and quality.

3.2. Greedy agglomeration

We describe a mesh agglomeration algorithm through use of a simple greedy heuristic, outlined in Algorithm 2. To construct a new mesh at a lower level, we assign to each element of the input mesh an integer weight corresponding to the number of neighbour elements in the mesh not yet processed. Elements are loaded into a priority queue and processed in ascending order according to the integer weights. To process an element, we identify the vertex of the element adjacent to the most unprocessed elements left in the priority queue, breaking ties at random. All the unprocessed elements touching the identified vertex are marked as processed, and agglomerated into a subset of the mesh partition, the union of which serves as an polyhedral element in the new mesh, termed a block. The priority queue is updated to reflect the removal the corresponding elements, and the algorithm repeated until no elements are left remaining in the

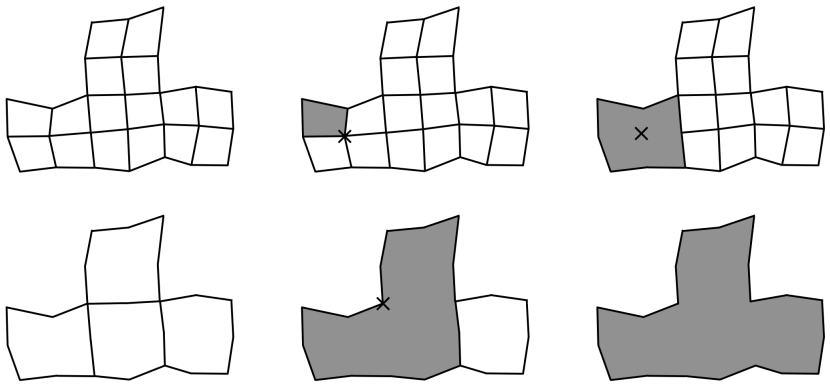


Figure 2: Schematic of agglomeration algorithm. Top row from left to right: Input mesh at level 0 of hierarchy, element with fewest number of neighbours and its vertex touching most unprocessed neighbours are chosen, all unprocessed elements touching chosen vertex are grouped to form 8 sided polygonal element for next level in hierarchy. Bottom row from left to right: Mesh at level 1 or hierarchy, all elements touching vertex are merged to form 16 sided polygonal element leaving the unshaded element with 0 unprocessed neighbours, unprocessed element with 0 unprocessed neighbour elements is merged into neighbouring agglomerate polygon.

priority queue. This process is shown in the first row of Figure 2.

In the case where an unprocessed element neighbours fewer than two unprocessed elements left in the priority queue, we instead append the element to the smallest adjacent block, breaking ties at random. The priority queue is then updated to reflect the successful processing of the element. This special case is shown in the bottom row of Figure 2.

Due to the use of a priority queue, which is inserted into and the minimum extracted from n times respectively, where n denotes the number of vertices in the mesh, the overall computational cost of the algorithm scales as $O(n \log n)$. The memory cost of the algorithm however scales only as $O(n)$, as only a single integer indicating the number of unprocessed neighbours to each vertex is stored in the priority queue.

3.3. Basis functions and quadrature

To define basis functions on the generated polyhedral blocks at each level

Algorithm 2: Greedy mesh agglomeration

Input mesh M with vertices $\{v_i\}$ and elements $\{e_i\}$
Create priority queue $P = \{(e_i, N(e_i)), \forall e_i \in M\}$, $N(e_i) = \#$neighbours of e_i
Create empty array N to store new elements
while $length(P) > 0$ **do**
  Pop e_{min} with smallest $N(e_{min})$ from P
  if $N(e_{min}) \geq 2$ **then**
    Find vertex v_{max} of e_{min} adjacent to the most elements in P
    Create set $E = \{e_j \in P, e_j \text{ adjacent to } v_{max}\} \cup \{e_{min}\}$
    for e_j in E **do**
      Remove e_j from P
      Update all neighbours e_k of e_j in P, $N(e_k) = N(e_k) - 1$
    end for
    Combine all elements in E to form new element, append to N
  else
    Create $N_{adj} = \{$ elements $n_j \in N$ with subelement e_k adjacent to $e_{min}\}$
    Find n_{min}, element in N with fewest subelements e_k
    Append e_{min} to n_{min}
    Update all neighbours e_k of e_{min} in P, $N(e_k) = N(e_k) - 1$
  end if
end while
return N

in the hierarchy, we adopt a modal basis set of polynomials on each block due to the difficulty of assigning nodal basis functions on arbitrary polyhedra. For instance the linear set of basis functions of this form would be simply $1, x, y$ in two dimensions. To numerically integrate on each of the blocks, we use the fact that each of the blocks are constructed by taking a union of a subset of elements $\{K_{n_1}, ..., K_{n_j}\}$ from the input mesh. This allows quadrature on polygonal elements to be computed by summing contributions from each sub-element of the block, which can be calculated using preexisting quadrature defined on the input mesh. Thus no additional computational expense due to quadrature is required at each coarser level of the mesh hierarchy.

3.4. Solution transfer

We define restriction and interpolation operators to transfer residuals and states between neighbouring levels in the mesh hierarchy. For the purposes of our preconditioning strategy, we focus only on residual restriction and the case of state prolongation but do not consider the case of state restriction.

The prolongation operator from level $l+1$ to level l acts as

$$L_{l+1}^{l} v_h^{l+1} = v_h^l. \tag{17}$$

As the basis functions for each polygonal block are chosen to be the same modal polynomials at each level, the operator can be chosen to be simple injection [4]. Following [18] this has an equivalent variational formulation, which can be defined using an L^2 projection.

The restriction operator is defined as the adjoint of the prolongation operator L_{l+1}^{l}

$$(R_l^{l+1} u_h^l, v_h^{l+1})_{l+1} = (u_h^l, L_{l+1}^{l} v_h^{l+1})_l \tag{18}$$

for all u_h^l, v_h^{l+1} piecewise polynomial functions defined on levels $l, l+1$ respectively. Equivalently using the L^2 weak formulation the restriction operator can be written as

$$M^{l+1} R_l^{l+1} = (M^l L_{l+1}^{l})^T \tag{19}$$

where M^l, M^{l+1} denote mass matrices for the corresponding superscript levels.

3.5. Operator coarsening

Coarsening a general operator A_l defined on level l to level $l+1$ is performed using the well known RAT method [22]. Specifically, to apply an operator A_l to a vector v_{l+1} on level $l+1$ of the mesh hierarchy: (1) the vector is interpolated

onto level l using the interpolation operator L_{l+1}^{l}, (2) the operator A_l is applied to the interpolated vector, (3) the resulting vector is restricted back to level $l+1$ using the restriction operator R_{l}^{l+1}. This procedure is equivalent to writing a coarsened operator on mesh hierarchy level $l+1$ as

$$A_{l+1} = R_{l}^{l+1} A_l L_{l+1}^{l} \tag{20}$$

4. Multigrid preconditioning

Our plan is to utilise a multi-level h-multigrid solver as a right preconditioner for an iterative Krylov solver to solve the system in Equation 15. We use a right preconditioner instead of left since its residual is identical to the true residual. While the system matrix for Poisson's problem is symmetric positive definite, allowing for use of the conjugate gradient method, we instead opt for the GMRES algorithm as it is extendable to other problems. Furthermore, in our numerical experiments we find that convergence is generally obtained in well under 50 iterations, enabling us to consider convergence behaviour without any effects from restarts.

4.1. Flux coarsening/Primal coarsening

Following the discussion in [10], direct coarsening of the operator obtained from the primal formulation of the CDG system results in a decline in multigrid performance. Instead, each operator in the flux formulation should be individually coarsened and the Schur complement taken at each level to reform the coarse primal formulation. Coarsening of the flux formulation operator from

level l to level $l+1$ can be written as

$$\begin{bmatrix} M_{l+1} & G_{l+1} \\ D_{l+1} & C_{l+1} \end{bmatrix} = \begin{bmatrix} R_l^{l+1} & 0 \\ 0 & R_l^{l+1} \end{bmatrix} \begin{bmatrix} M_l & G_l \\ D_l & C_l \end{bmatrix} \begin{bmatrix} L_{l+1}^l & 0 \\ 0 & L_{l+1}^l \end{bmatrix} \tag{21}$$

$$A_{l+1} = C_{l+1} - D_{l+1} M_{l+1}^{-1} G_{l+1}$$

We verify the decline in multigrid performance from directly coarsening the primal operator in Section 5.1, as opposed to coarsening using the flux formulation.

4.2. CDG switch functions

While the CDG method has been shown to be stable and retains compactness in the primal form irrespective of the choice of switch function, it can however affect the sparsity of the matrix C in Equation 16 of the flux formulation and which can in turn affect the performance of multigrid flux operator coarsening. In particular, for each element K_n separated from an element K_i by a single element K_j, the (K_i, K_n) block of the C matrix is nonzero if the two conditions are satisfied:

1. the switch on the edge separating K_i, K_j is $S_{K_i}^{K_j} = 1$,
2. the switch on the edge separating K_j, K_n is $S_{K_j}^{K_n} = -1$.

This implies for optimal coarsening of the operator C, for each partition of elements $\mathcal{T}_h$, all subsets of the partition must be closed under second neighbours that satisfy the above two properties. This is however in general impossible to satisfy for an arbitrary input mesh unless the partition consists only of one subset equal to the entire mesh.

In practice, a consistent switch function may be used to minimise the number of second neighbour interactions in C not accounted for in the operator coarsening step. A consistent switch function is one where for each element K

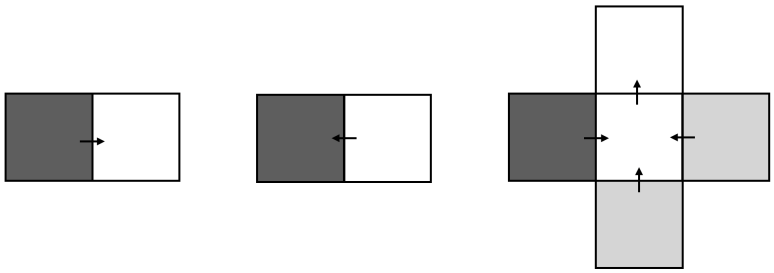


Figure 3: Effect of CDG switch function on sparsity of C in Equation 16. On the left, arrow pointing from dark element K_d to blank element K_w implies switch function $S_{K_d}^{K_w} = 1$. Middle figure shows arrow pointing from blank element K_w to shaded element Kd, denoting that $S_{K_d}^{K_w} = -1$. On the right, shaded element K_d interacts with second neighbour elements shown in light grey K_{g_1}, K_{g_2} as a result of the given switch, implying that blocks $(K_d, K_{g_1}), (K_d, K_{g_2})$ of the matrix C to be nonzero.

with the set of neighbours $\{K_i\}$

$$|\sum_{K_i} S_K^{K_i}| < |\{K_i\}| \tag{22}$$

That is, there must be at least one inter-element boundary separating elements K, K' where the switch $S_K^{K'} = -1$, and another where the switch $S_K^{K'} = 1$. The effect on performance of the multigrid preconditioner due to choice of switch function is demonstrated in Section 5.3.

4.3. Multigrid V-cycle

For the h-multigrid solver, we use a single V-cycle wherein a hierarchy of meshes constructed via agglomeration is traversed using the L^2 projection operators outlined in Section 3.4. At each level, various iterations of a smoother are applied, except at the coarsest level where, the problem is solved directly.

Algorithm 3: Multigrid V-cycle

Input matrix $A^{(0)}$, vector $b^{(0)}$, set $x^{(0)} = \mathbf{0}$

Input mesh hierarchy $E = \{E_0, E_1, ..., E_n\}$

for $k = 1 : n - 1$ **do**

  Project to current level $b^k = R^k_{k-1}(b^{k-1} - A^{k-1}x^{k-1})$

  Construct smoother $\tilde{A}_k$ from A_k

  for $i = n_{pre}$ **do**

    Apply smoother, $x^k = x^k + \alpha\tilde{A}_k^{-1}(b^k - A^k x^k)$

  end for

end for

Solve $A^n x^n = b^n$ directly

for $k = n - 1 : 1$ **do**

  Project to current level $x^k = x^k + L^k_{k+1}x^{k+1}$

  Construct smoother $\tilde{A}_k$ from A_k

  for $i = n_{pre}$ **do**

    Apply smoother, $x^k = x^k + \alpha\tilde{A}_k^{-1}(b^k - A^k x^k)$

  end for

end for

return $x^{(0)}$

Commonly used smoother include block Jacobi, block Gauss-Seidel, or incomplete LU factorisations. In this work we focus on block Jacobi smoothers with a damping factor $\alpha = \frac{2}{3}$, as they are simple to parallelise for large systems.

4.4. hp-multigrid

In this manuscript, we consider explicitly only the case of linear basis functions in the Discontinuous Galerkin discretisation. For problems with higher polynomial degree basis functions, we would first employ standard p-multigrid [9, 17] on the fine mesh to project down to $p = 1$ basis functions, and then

followed by our agglomeration method. For simplicity and to only highlight the h-multigrid procedure, we only consider $p = 1$ in our examples.

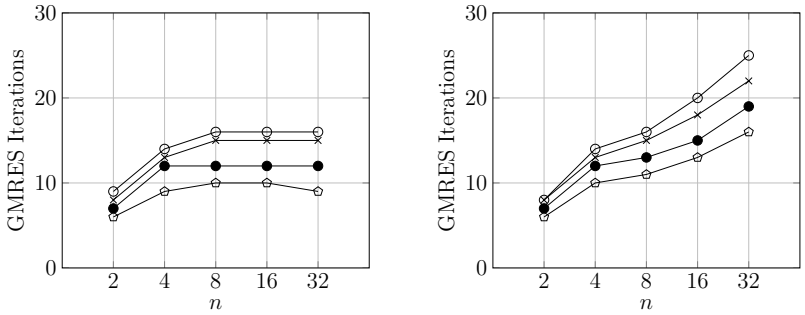
5. Numerical results

In this section we present numerical results to evaluate the performance of the multigrid preconditioner. Unless otherwise stated, a consistent switch function is used for flux definition in the CDG discretization. Our initial solution vector is always set as the zero vector, and we iterate until we reach a tolerance of 10^{-8} in the relative norm $|Au_h - b|/|b|$. In all examples, we agglomerate elements in the mesh until the lowest level in the h-multigrid hierarchy consists of only one element using the greedy algorithm described above.

Following the discussion in Section 4.4, for all the following examples, we consider only linear order basis functions for both basis functions defined on the initial mesh and on all subsequent meshes in the h-multigrid hierarchy. Within the multigrid V-cycle we choose the number of pre-smoothing steps $n_{pre} = 0$, and the number of post-smoothing steps $n_{post} = 3$. We choose a GMRES restart parameter of 50, as in all the following examples we converge in fewer iterations and so convergence is not impacted by any restarts. We also do not consider true computational time in this study, and report only the number of iterations required for convergence.

5.1. Flux vs primal coarsening and choice of Dirichlet parameter

We start by solving Poisson's problem on the domain $\Omega = [0, 1]^2$ using a uniform square $n \times n$ mesh. We impose Neumann conditions on the vertical boundaries at $x = 0, 1$, in addition to Dirichlet boundary conditions on the horizontal boundaries at $y = 0, 1$. We build the h-multigrid hierarchy using Algorithm 2, which is shown in the top row of Figure 5.



(a) Flux coarsening (b) Primal coarsening

Figure 4: h-multigrid convergence for flux vs. primal coarsening on a square $n \times n$ mesh. Different plot markers indicate varying Dirichlet parameters C_D: —○—, —×—, —●—, —⬠— denote values of $C_D = 10^1/h_{avg},\ 10^2/h_{avg},\ 10^3/h_{avg},\ 10^4/h_{avg}$, respectively.

Figure 4 shows the number of iterations to convergence for the square mesh under h-refinement, in addition to varying values of the Dirichlet penalty parameter C_D. A deterioration in performance under primal coarsening as $h_{avg} \to 0$ is seen as in [10], whereas performance under flux coarsening does not suffer similar problems

Increased performance is also gained through using larger values of C_D, a result the choice of block Jacobi as the smoother in the h-multigrid solver. As only the magnitude of values in the blocks on the diagonal of matrix C in Equation 16 scale with the value of C_D, an increase in C_D implies an increase to the values in the blocks on the diagonal of A in Equation 15 relative to values in blocks off the diagonal of A. Based on these observations, going forward for the remainder of our tests, we focus on flux coarsening using a value of $C_D = 10^4/h_{avg}$.

5.1.1. Hierarchy element shapes

We investigate the effect of irregular element shapes in the h-multigrid hierarchy on performance by considering once more a domain $\Omega = [0, 1]^2$, with Neumann conditions on the boundaries at $x = 0, 1$, and Dirichlet conditions

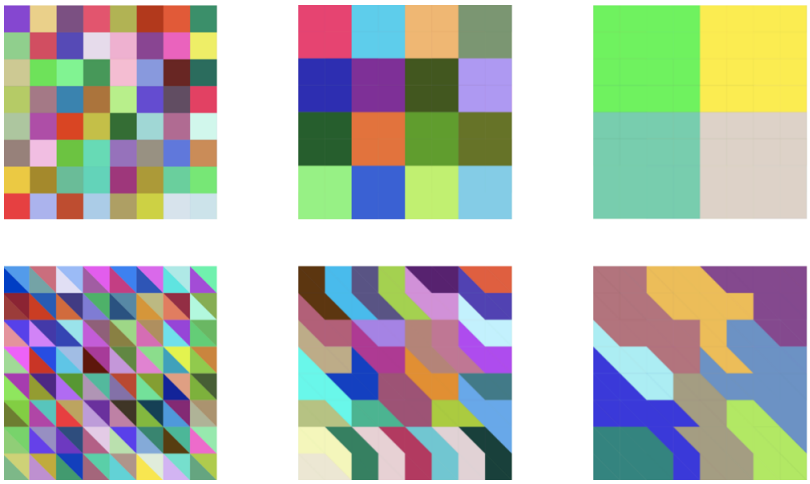


Figure 5: Regular and irregular mesh hierarchies. Mesh hierarchies shown from left to right correspond to level = 0,1,2 of a $8x8$ square mesh respectively.

on the boundaries at $y = 0, 1$. We compare the number of iterations to convergence using two different mesh hierarchies as shown in Figure 5, one with regular quadrilateral shaped elements at each level in the h-multigrid and the other with highly irregularly shaped elements that are in general non-convex.

The plot in Figure 5.1.1 show the number of iterations to convergence using the regular and irregular shaped elements respectively. While a decrease in performance is observed in using irregularly shaped elements in the h-multigrid hierarchy, the decrease is independent of element size h_{avg}.

5.2. NACA airfoil

To investigate the effect of non-uniform element sizes and also of meshes which are not simply connected, we consider the example of Poisson's problem on a rectangular domain around a NACA airfoil. Figure 7 shows the h-multigrid hierarchy of the coarsest input mesh consisting of 605 elements, shown in the top left of the figure. Dirichlet conditions are applied on the boundary at the airfoil and at the two horizontal boundaries, while Neumann conditions are applied at the two vertical boundaries.

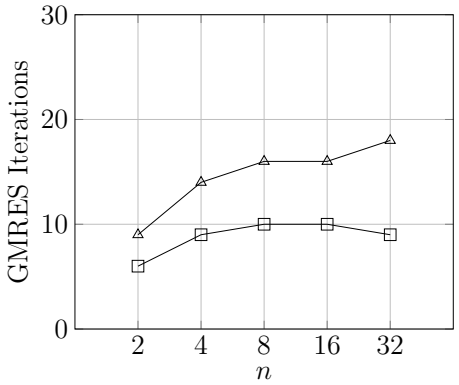


Figure 6: Comparison of multigrid performance with varying mesh hierarchy element shapes on a square $n \times n$ mesh. —□— denotes the number of iterations on a regular mesh hierarchy, —△— denotes the number of iterations on a non-regular mesh hierarchy.

Figure 7: Multigrid hierarchy for airfoil mesh.

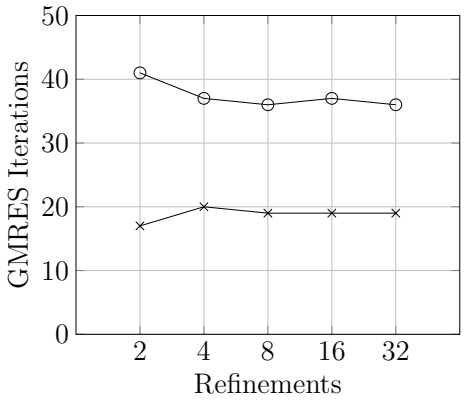


Figure 8: GMRES convergence for Poisson's problem on airfoil mesh. —×— shows the number of iterations using a consistent switch function, —○— shows the number of iterations using a natural switch function.

5.2.1. Switch function

Figure 8 shows the number of iterations to convergence for the airfoil problem using a consistent switch function, and a natural switch function based on random element enumeration. The performance of the multigrid preconditioner is shown to clearly deteriorate with a poorly chosen switch function. Using a consistent switch function, the presence of non simply connected elements in the mesh hierarchy does not seem to have a large effect on the performance of the multigrid preconditioner.

5.3. Convection-Diffusion

Finally, we consider the more general example of a convection-diffusion equation on the airfoil mesh:

$$\beta \boldsymbol{v} \cdot \nabla u + \Delta u = f. \tag{23}$$

We employ zero Dirichlet boundary conditions everywhere, we set $f = 1$, and the velocity field $\boldsymbol{v} = (1, 0)$. The resulting convergence in the GMRES iterations is shown in Figure 9, for a range of values of β under refinement.

We see that the performance of the preconditioner is largely unaffected for

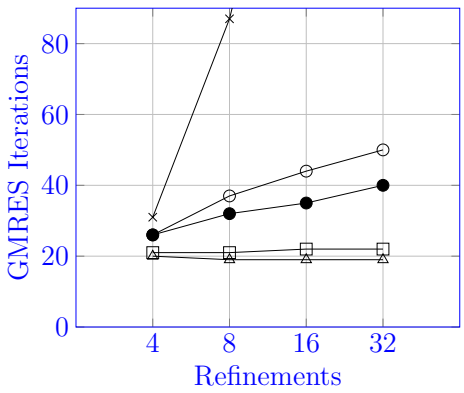


Figure 9: GMRES convergence for convection-diffusion equation on airfoil mesh. The number of iterations are shown for $\beta = 100$ by —×—, $\beta = 10$ by —○—, $\beta = 1$ by —●—, $\beta = 0.1$ by —□— and $\beta = 0$ by —△—.

small values of β, but quickly deteriorates with higher magnitudes of β. This is expected as it changes the structure of the problem. We note however that this can be fixed by using other existing smoothers for convection such as line-based solvers [9], or ILU/Gauss-Seidel with good element ordering [17].

6. Conclusions

We have developed an algorithm for constructing suitable mesh hierarchies for the geometric multigrid method via use of simple element agglomeration. The merged elements will in general be polyhedral, which are easily supported using a discontinuous Galerkin discretization. While the method should perform well with any choice of numerical fluxes, we have used the Compact DG method and showed that in this case a consistent switch function gives better multigrid performance. The resulting solver gives excellent performance for Poisson's equation on fully unstructured meshes, as well as for convection-diffusion with moderate magnitudes of the convective component. Future work include extension to other equations, parallelization, and numerical examples in 3D.

Acknowledgments

This work was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny V. Dobrev, Y. Dudouit, A. Fisher, Tz. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, and S. Zampini. MFEM: A modular finite element library. *Computers & Mathematics with Applications*, 2020.

[2] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779 (electronic), 2001/02.

[3] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31(138):333–390, 1977.

[4] Susanne C. Brenner and L. Ridgway Scott. *The mathematical theory of finite element methods*, volume 15 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 1994.

[5] Tony F. Chan, Jinchao Xu, and Ludmil Zikatanov. An agglomeration multigrid method for unstructured grids. In *Domain decomposition methods, 10 (Boulder, CO, 1997)*, volume 218 of *Contemp. Math.*, pages 67–81. Amer. Math. Soc., Providence, RI, 1998.

[6] B. Cockburn and C.-W. Shu. The local discontinuous Galerkin method for time-dependent convection-diffusion systems. *SIAM J. Numer. Anal.*, 35(6):2440–2463 (electronic), 1998.

[7] Steven Dargaville, Andrew G Buchan, Richard P Smedley-Stevenson, Paul N Smith, and Christopher C Pain. A comparison of element agglomeration algorithms for unstructured geometric multigrid. *arXiv preprint arXiv:2005.09104*, 2020.

[8] Sven-Erik Ekström and Martin Berggren. Agglomeration multigrid for the vertex-centered dual discontinuous Galerkin method. In *ADIGMA-A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications*, pages 301–308. Springer, 2010.

[9] K.J. Fidkowski, T.A. Oliver, J. Lu, and D.L. Darmofal. p-multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 207(1):92–113, 2005.

[10] Daniel Fortunato, Chris H. Rycroft, and Robert Saye. Efficient operator-coarsening multigrid schemes for local discontinuous Galerkin methods. *SIAM J. Sci. Comput.*, 41(6):A3913–A3937, 2019.

[11] Van Emden Henson and Ulrike Meier Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. volume 41, pages 155–177. 2002. Developments and trends in iterative methods for large systems of equations—in memoriam Rüdiger Weiss (Lausanne, 2000).

[12] Jim E. Jones and Panayot S. Vassilevski. AMGe based on element agglomeration. *SIAM J. Sci. Comput.*, 23(1):109–133, 2001.

[13] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1999.

[14] C. Nastase and D. Mavriplis. A parallel hp-multigrid solver for three-dimensional discontinuous Galerkin discretizations of the Euler equations.

In *45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada*, 2007. AIAA-2007-512.

[15] Will Pazner. Efficient Low-Order Refined Preconditioners for High-Order Matrix-Free Continuous and Discontinuous Galerkin Methods. *SIAM J. Sci. Comput.*, 42(5):A3055–A3083, 2020.

[16] J. Peraire and P.-O. Persson. The compact discontinuous Galerkin (CDG) method for elliptic problems. *SIAM J. Sci. Comput.*, 30(4):1806–1824, 2008.

[17] P.-O. Persson and J. Peraire. Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier-Stokes equations. *SIAM J. Sci. Comput.*, 30(6):2709–2733, 2008.

[18] Rahul S. Sampath and George Biros. A parallel geometric multigrid method for finite elements on octree meshes. *SIAM J. Sci. Comput.*, 32(3):1361–1392, 2010.

[19] Robert I. Saye. Efficient multigrid solution of elliptic interface problems using viscosity-upwinded local discontinuous Galerkin methods. *Commun. Appl. Math. Comput. Sci.*, 14(2):247–283, 2019.

[20] D. Strauss and J.L.F. Azevedo. On the development of an agglomeration multigrid solver for turbulent flows. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 25(4):315–324, 2003.

[21] Hari Sundar, Georg Stadler, and George Biros. Comparison of multigrid algorithms for high-order continuous finite element discretizations. *Numer. Linear Algebra Appl.*, 22(4):664–680, 2015.

[22] Jinchao Xu. Iterative methods by space decomposition and subspace correction. *SIAM Rev.*, 34(4):581–613, 1992.

[23] Jinchao Xu and Ludmil Zikatanov. Algebraic multigrid methods. *Acta Numer.*, 26:591–721, 2017.