Implementation of Hybridizable Discontinuous Galerkin Methods via the HDG Branch

Tamás Horváth
Oakland University

3^{rd} MFEM Community Workshop
10. 26. 2023.

Contents

Collaborators

Thank you to

Sander Rhebergen (University of Waterloo, Canada)

Abdullah Ali Sivas (University of Waterloo, Canada)

Tan Bui-Thanh (University of Texas at Austin)

Jau-Uei Chen (University of Texas at Austin)

Natasha S. Sharma (University of Texas at El Paso)

Giselle Sosa Jones (University of Waterloo, Canada $\rightarrow$ Oakland University)

DIFFUSION PROBLEM*

Let $\Omega \subset \mathbb{R}^d$, $d \geq 1$ open domain, $f \in L^2(\Omega)$

$$
\begin{aligned}
-\nabla \cdot (\nu \nabla u) &= f && \text{in } \Omega\,, \\
u &= g_D && \text{on } \Gamma\,.
\end{aligned}
$$

Rewrite to a first order system

$$
\begin{aligned}
\mathbf{q} + \nu \nabla u &= 0 && \text{in } \Omega\,, \\
\nabla \cdot \mathbf{q} &= f && \text{in } \Omega\,, \\
u &= g_D && \text{on } \Gamma\,.
\end{aligned}
$$

* Nguyen, N.C., Peraire, J. and Cockburn, B., 2009. An implicit high-order hybridizable discontinuous Galerkin method for linear convection–diffusion equations. Journal of Computational Physics, 228(9), pp 3232-3254

HDG discretization

$$\mathbf{V}_h = \{\mathbf{v}_h \in [L^2(\Omega)]^d : \mathbf{v}|_{K_i} \in [P^p]^d,\ \forall K_i \in \mathcal{T}_h\}$$
$$W_h = \{v_h \in L^2(\Omega) : v|_{K_i} \in P^p,\ \forall K_i \in \mathcal{T}_h\}$$
$$M_h = \{\lambda_h \in L^2(\mathcal{E}_h) : \lambda_h|_e \in P^p(e),\ \forall e \in \mathcal{E}_h\}$$

$$\begin{aligned} -(\mathbf{q}_h, \mathbf{v}_h)_{\mathcal{T}_h} + (u_h, \nu\nabla\cdot\mathbf{v}_h)_{\mathcal{T}_h} - \langle \lambda_h, \nu\mathbf{v}_h\cdot\mathbf{n}\rangle_{\partial\mathcal{T}_h} &= 0 \\ (\nabla\cdot\mathbf{q}_h, w_h)_{\mathcal{T}_h} + \langle \tau u_h, w_h\rangle_{\partial\mathcal{T}_h} - \langle \tau\lambda_h, w_h\rangle_{\partial\mathcal{T}_h} &= (f, w)_{\mathcal{T}_h}\,, \\ -\langle [\![\mathbf{q}_h\cdot\mathbf{n}]\!], \mu_h\rangle_{\mathcal{E}_h} - \langle [\![\tau u_h]\!], \mu_h\rangle_{\mathcal{E}_h} + \langle [\![\tau\lambda_h]\!], \mu_h\rangle_{\mathcal{E}_h} &= 0 \end{aligned}$$

where

$$(u, v)_{\mathcal{T}_h} = \sum_{K\in\mathcal{T}_h}(u, v)_K, \qquad \langle u, v\rangle_{\partial\mathcal{T}_h} = \sum_{K\in\mathcal{T}_h}\langle u, v\rangle_{\partial K}, \qquad \langle \lambda, \mu\rangle_{\mathcal{E}_h} = \sum_{f\in\mathcal{F}_h}\langle \lambda, \mu\rangle_f,$$

ADVECTION PROBLEM AND HDG DISCRETIZATION†

$$\begin{aligned} \beta u + \nabla \cdot \mathbf{c} u &= f \qquad \text{in } \Omega\,, \\ u &= g \qquad \text{on } \Gamma_-\,, \end{aligned}$$

$$\begin{aligned} (\beta u_h, v_h)_{\mathcal{T}_h} - (\mathbf{c} u_h, \nabla v_h)_{\mathcal{T}_h} + \langle \mathbf{n} \cdot \mathbf{c} u_h, v_h \rangle_{\partial \mathcal{T}_{h+}} + \langle \mathbf{n} \cdot \mathbf{c} \lambda_h, v_h \rangle_{\partial \mathcal{T}_{h-}} &= (f, v_h)_{\mathcal{T}_h} \\ \sum_K \langle \mathbf{n} \cdot \mathbf{c} u_h, \mu_h \rangle_{\partial K_+} + \sum_K \langle \mathbf{n} \cdot \mathbf{c} \lambda_h, \mu_h \rangle_{\partial K_-} - \langle \mathbf{c} \cdot \mathbf{n} \lambda_h, \mu_h \rangle_{\Gamma_+} &= \langle g, \mu_h \rangle_{\Gamma_-} \end{aligned}$$

† Wells, G.N., 2011. Analysis of an interface stabilized finite element method: the advection-diffusion-reaction equation. SIAM Journal on Numerical Analysis, 49(1), pp.87-109.

Matrix form & Schur complement

Using U and Λ for the volume and skeletal coefficient vectors

Block system

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} U \\ \Lambda \end{bmatrix} = \begin{bmatrix} F \\ G \end{bmatrix}$$

Matrix form & Schur complement

Using U and Λ for the volume and skeletal coefficient vectors

Block system

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} U \\ \Lambda \end{bmatrix} = \begin{bmatrix} F \\ G \end{bmatrix}$$

A is block diagonal

$$U = A^{-1}(F - B\Lambda)$$
$$CA^{-1}(F - B\Lambda) + D\Lambda = G$$

Matrix form & Schur complement

Using U and Λ for the volume and skeletal coefficient vectors

Block system

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} U \\ \Lambda \end{bmatrix} = \begin{bmatrix} F \\ G \end{bmatrix}$$

A is block diagonal

$$U = A^{-1}(F - B\Lambda)$$
$$CA^{-1}(F - B\Lambda) + D\Lambda = G$$

Final problem

$$(D - CA^{-1}B)\Lambda = G - CA^{-1}F$$
$$U = A^{-1}(F - B\Lambda)$$

Matrix form & Schur complement

Using U and Λ for the volume and skeletal coefficient vectors

Block system

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} U \\ \Lambda \end{bmatrix} = \begin{bmatrix} F \\ G \end{bmatrix}$$

A is block diagonal

$$U = A^{-1}(F - B\Lambda)$$
$$CA^{-1}(F - B\Lambda) + D\Lambda = G$$

Final problem

$$(D - CA^{-1}B)\Lambda = G - CA^{-1}F$$
$$U = A^{-1}(F - B\Lambda)$$

A, B, C, D can be block matrices

DG Assembly

Algorithm 1 DG Assembly loop

```
loop Over all elements
    Calculate the volume integrals
end loop
loop Over all faces
    Calculate the face integrals, add contributions to two neighboring elements
end loop
```

Looping over faces only once

HDG ASSEMBLY

Algorithm 2 HDG Assembly

```
loop Over all elements
    Calculate the volume integrals
    loop Over all faces of the element
        Calculate the face integrals for A, B, C, D, add contributions to local matrices
    end loop
    Invert A locally
    Boundary elimination
    Calculate Schur complement locally (D − CA^{-1}B)
end loop
```

Looping over interior faces twice - once for both neighboring elements

HDG Reconstruction

Algorithm 3 HDG Reconstruction

```
loop Over all elements
    Calculate the volume integrals
    loop Over all faces of the element
        Calculate the face integrals for A and B only, add contributions to local matrices
    end loop
    Invert A locally
    Reconstruct U = A^{-1}(F − BΛ) locally
end loop
```

Looping over interior faces twice - once for both neighboring elements

A, B can be stored during the Assembly process to save time (storage vs time)

Calculating the volume integrals

Volume integrators: One shot integrator, calculates all terms in the local A matrix

Calculating the volume integrals

Volume integrators: One shot integrator, calculates all terms in the local A matrix

For the diffusion problem the local matrix A is a block matrix (coefficients of q_h and u_h)

Calculating the volume integrals

Volume integrators: One shot integrator, calculates all terms in the local A matrix

For the diffusion problem the local matrix A is a block matrix (coefficients of q_h and u_h)

Most of the integrators in A are standard MFEM integrators

Calculating the volume integrals

Volume integrators: One shot integrator, calculates all terms in the local *A* matrix

For the diffusion problem the local matrix *A* is a block matrix (coefficients of q_h and u_h)

Most of the integrators in *A* are standard MFEM integrators

We use problem specific integrators, abandoning the modular approach

Calculating the volume integrals

Volume integrators: One shot integrator, calculates all terms in the local A matrix

For the diffusion problem the local matrix A is a block matrix (coefficients of q_h and u_h)

Most of the integrators in A are standard MFEM integrators

We use problem specific integrators, abandoning the modular approach

Could be done in a modular fashion, but needs bookkeeping (to which submatrix to assemble to)

Volume integrator example

$$
\begin{aligned}
-(\mathbf{q}_h, \mathbf{v}_h)_{\mathcal{T}_h} + (u_h, \nu \nabla \cdot \mathbf{v}_h)_{\mathcal{T}_h} - \langle \lambda_h, \nu \mathbf{v}_h \cdot \mathbf{n} \rangle_{\partial \mathcal{T}_h} &= 0 \\
(\nabla \cdot \mathbf{q}_h, w_h)_{\mathcal{T}_h} + \langle \tau u_h, w_h \rangle_{\partial \mathcal{T}_h} - \langle \tau \lambda_h, w_h \rangle_{\partial \mathcal{T}_h} &= (f, w)_{\mathcal{T}_h}, \\
-\langle [\![\mathbf{q}_h \cdot \mathbf{n}]\!], \mu_h \rangle_{\mathcal{E}_h} - \langle [\![\tau u_h]\!], \mu_h \rangle_{\mathcal{E}_h} + \langle [\![\tau \lambda_h]\!], \mu_h \rangle_{\mathcal{E}_h} &= 0
\end{aligned}
$$

Algorithm 4 HDG Diffusion Integrator

loop Over all integration point
    Calculate the shape values, dshape values
    Calculate a MasssVectorIntegrator for $-(\mathbf{q}_h, \mathbf{v}_h)$
    Calculate a VectorDivergenceIntegrator for $(\nabla \cdot \mathbf{q}_h, w_h)$
end loop
Add the local matrices to A_{11}, A_{12}, A_{21}

Interface integrator

Interface integrators: One shot integrator, calculates all terms in the local A, B, C, D matrices

INTERFACE INTEGRATOR

Interface integrators: One shot integrator, calculates all terms in the local A, B, C, D matrices

The skeletal unknown related integrators are not standard in MFEM

INTERFACE INTEGRATOR

Interface integrators: One shot integrator, calculates all terms in the local A, B, C, D matrices

The skeletal unknown related integrators are not standard in MFEM

Normal in MFEM is calculated such that it is outward normal for Elem1

Interface integrator

Interface integrators: One shot integrator, calculates all terms in the local A, B, C, D matrices

The skeletal unknown related integrators are not standard in MFEM

Normal in MFEM is calculated such that it is outward normal for Elem1

If we need the integrals including Elem2 unknowns: multiply with -1

INTERFACE INTEGRATOR

Interface integrators: One shot integrator, calculates all terms in the local A, B, C, D matrices

The skeletal unknown related integrators are not standard in MFEM

Normal in MFEM is calculated such that it is outward normal for Elem1

If we need the integrals including Elem2 unknowns: multiply with -1

We need to keep a boolean called "elem1or2"

INTERFACE INTEGRATOR

Interface integrators: One shot integrator, calculates all terms in the local A, B, C, D matrices

The skeletal unknown related integrators are not standard in MFEM

Normal in MFEM is calculated such that it is outward normal for Elem1

If we need the integrals including Elem2 unknowns: multiply with -1

We need to keep a boolean called "elem1or2"

We use problem specific integrators, abandoning the modular approach

Interface integrator

Interface integrators: One shot integrator, calculates all terms in the local A, B, C, D matrices

The skeletal unknown related integrators are not standard in MFEM

Normal in MFEM is calculated such that it is outward normal for Elem1

If we need the integrals including Elem2 unknowns: multiply with -1

We need to keep a boolean called "elem1or2"

We use problem specific integrators, abandoning the modular approach

Could be done in a modular fashion, but needs bookkeeping (to which submatrix to assemble to)

INTERFACE INTEGRATOR EXAMPLE

$$
\begin{aligned}
-(\mathbf{q}_h, \mathbf{v}_h)_{\mathcal{T}_h} + (u_h, \nu \nabla \cdot \mathbf{v}_h)_{\mathcal{T}_h} - \langle \lambda_h, \nu \mathbf{v}_h \cdot \mathbf{n} \rangle_{\partial \mathcal{T}_h} &= 0 \\
(\nabla \cdot \mathbf{q}_h, w_h)_{\mathcal{T}_h} + \langle \tau u_h, w_h \rangle_{\partial \mathcal{T}_h} - \langle \tau \lambda_h, w_h \rangle_{\partial \mathcal{T}_h} &= (f, w)_{\mathcal{T}_h}, \\
-\langle [\![\mathbf{q}_h \cdot \mathbf{n}]\!], \mu_h \rangle_{\mathcal{E}_h} - \langle [\![\tau u_h]\!], \mu_h \rangle_{\mathcal{E}_h} + \langle [\![\tau \lambda_h]\!], \mu_h \rangle_{\mathcal{E}_h} &= 0
\end{aligned}
$$

Algorithm 5 HDG Diffusion Integrator

```
loop Over all integration point
    Calculate the shape values, dshape values, shape_facet values
    Calculate a "MassInterfaceIntegrator" for ⟨τu_h, w_h⟩
    Calculate a "MassSkeletonIntegrator" for ⟨[[τλ_h]], μ_h⟩
    Calculate a "NormalSkeletonTraceJumpIntegrator" for ⟨[[q_h · n]], μ_h⟩
    Calculate a "MassMixedSkeletonIntegrator" for ⟨[[τu_h]], μ_h⟩
end loop
Add the local matrices to A_22, B_11, B_21, C_11, C_12, D
```

Unified HDG BilinearForm

One bilinear form for the advection and the diffusion problems

Unified HDG BilinearForm

One bilinear form for the advection and the diffusion problems

Arrays of FES and GridFunction

Unified HDG BilinearForm

One bilinear form for the advection and the diffusion problems

Arrays of FES and GridFunction

To make the code more user friendly: overload the functions

```
HDGBilinearForm(Array<FiniteElementSpace*> &_volume_fes,
                Array<FiniteElementSpace*> &_skeletal_fes,
                bool _parallel = false);

// Diffusion test case without FES arrays
HDGBilinearForm(FiniteElementSpace *_fes1,
                FiniteElementSpace *_fes2,
                FiniteElementSpace *_fes3,
                bool _parallel = false);
```

Unified HDG BilinearForm

One bilinear form for the advection and the diffusion problems

Arrays of FES and GridFunction

To make the code more user friendly: overload the functions

```
HDGBilinearForm(Array<FiniteElementSpace*> &_volume_fes,
                Array<FiniteElementSpace*> &_skeletal_fes,
                bool _parallel = false);

// Diffusion test case without FES arrays
HDGBilinearForm(FiniteElementSpace *_fes1,
                FiniteElementSpace *_fes2,
                FiniteElementSpace *_fes3,
                bool _parallel = false);
```

One single AssembleReconstruct function

Unified HDG BilinearForm

One bilinear form for the advection and the diffusion problems

Arrays of FES and GridFunction

To make the code more user friendly: overload the functions

```
HDGBilinearForm(Array<FiniteElementSpace*> &_volume_fes,
                Array<FiniteElementSpace*> &_skeletal_fes,
                bool _parallel = false);

// Diffusion test case without FES arrays
HDGBilinearForm(FiniteElementSpace *_fes1,
                FiniteElementSpace *_fes2,
                FiniteElementSpace *_fes3,
                bool _parallel = false);
```

One single AssembleReconstruct function

Turned into a miniapp

AssembleReconstruct loop

About 50% of the Assemble and Reconstruct loops were the same

ASSEMBLERECONSTRUCT LOOP

About 50% of the Assemble and Reconstruct loops were the same

Algorithm 7 HDG AssembleReconstruct

```
loop Over all elements
    Calculate the volume integrals
    loop Over all faces of the element
        Calculate the face integrals for A, B and maybe C and D. Add contributions to local matrices
    end loop
    Invert A locally
    if assembly then
        Boundary elimination
        Calculate Schur complement locally (D − CA^{-1}B)
    else
        Reconstruct U = A^{-1}(F − BΛ) locally
    end if
end loop
```

Warning - boundary faces

Accessing a boundary face vs accessing the same face as interior face may give a different DoF order

Differences between GetFaceVDofs and GetBdrElementVDofs Issue 2514
`https://github.com/mfem/mfem/issues/2514`

WARNING - BOUNDARY FACES

Accessing a boundary face vs accessing the same face as interior face may give a different DoF order

Differences between GetFaceVDofs and GetBdrElementVDofs Issue 2514
`https://github.com/mfem/mfem/issues/2514`

We do not have a separate interior and boundary face integrator

WARNING - BOUNDARY FACES

Accessing a boundary face vs accessing the same face as interior face may give a different DoF order

Differences between GetFaceVDofs and GetBdrElementVDofs Issue 2514
`https://github.com/mfem/mfem/issues/2514`

We do not have a separate interior and boundary face integrator

Need to check if element boundary is an interior face or a boundary face

Warning - boundary faces

Accessing a boundary face vs accessing the same face as interior face may give a different DoF order

Differences between GetFaceVDofs and GetBdrElementVDofs Issue 2514
`https://github.com/mfem/mfem/issues/2514`

We do not have a separate interior and boundary face integrator

Need to check if element boundary is an interior face or a boundary face

Projection of a function to the facet terms is different for interior and boundary faces

What have we used it for?

(Space-time) (Navier–)Stokes: 2 volume FES + 2 skeletal FES

What have we used it for?

(Space-time) (Navier–)Stokes: 2 volume FES + 2 skeletal FES

Magnetohydrodynamics: 6 volume FES + 4 skeletal FES

What have we used it for?

(Space-time) (Navier–)Stokes: 2 volume FES + 2 skeletal FES

Magnetohydrodynamics: 6 volume FES + 4 skeletal FES

Phase-field crystal: 7 volume FES + 3 skeletal FES

What have we used it for?

(Space-time) (Navier–)Stokes: 2 volume FES + 2 skeletal FES

Magnetohydrodynamics: 6 volume FES + 4 skeletal FES

Phase-field crystal: 7 volume FES + 3 skeletal FES

Not all of the spaces are the same (different order, continuous/discontinuous facet spaces)

WHAT HAVE WE USED IT FOR?

(Space-time) (Navier–)Stokes: 2 volume FES + 2 skeletal FES

Magnetohydrodynamics: 6 volume FES + 4 skeletal FES

Phase-field crystal: 7 volume FES + 3 skeletal FES

Not all of the spaces are the same (different order, continuous/discontinuous facet spaces)

The elimination and the integrators had to be modified for the particular cases

What have we used it for?

(Space-time) (Navier–)Stokes: 2 volume FES + 2 skeletal FES

Magnetohydrodynamics: 6 volume FES + 4 skeletal FES

Phase-field crystal: 7 volume FES + 3 skeletal FES

Not all of the spaces are the same (different order, continuous/discontinuous facet spaces)

The elimination and the integrators had to be modified for the particular cases

Fast calculation of A inverse in some of the cases

Future of the branch

Planned updates

Add the possibility of saving the local A and B matrices to accelerate reconstruction

Fix a PETSc issue (might be unrelated to the branch)

Future of the branch

Planned updates

Add the possibility of saving the local A and B matrices to accelerate reconstruction

Fix a PETSc issue (might be unrelated to the branch)

Not in the works (anyone interested?)

More modular implementation of the integrator

- Pros: more MFEM style, reusable integrators
- Cons: hard bookkeeping (which submatrix to add to which block)

Future of the branch

Planned updates

Add the possibility of saving the local A and B matrices to accelerate reconstruction

Fix a PETSc issue (might be unrelated to the branch)

Not in the works (anyone interested?)

More modular implementation of the integrator
- Pros: more MFEM style, reusable integrators
- Cons: hard bookkeeping (which submatrix to add to which block)

Nonconforming meshes
- Pros: it would be nice
- Cons: we never needed it, so we ignored it

Thank you!