How to implement HDG Methods Efficiently

Abdullah Ali Sivas

December 22, 2017

Table of contents

What is already around?

General Scientific Computing Packages

Some tools that are commonly used

General Scientific Computing Packages

Some tools that are commonly used

PETSc: Parallel library of matrix and vector data structures, preconditioners, iterative solvers, nonlinear solvers, ODE solvers, GPU support

General Scientific Computing Packages

Some tools that are commonly used

PETSc: Parallel library of matrix and vector data structures, preconditioners, iterative solvers, nonlinear solvers, ODE solvers, GPU support

SLEPc: Parallel library for the solution of large sparse eigenproblems

General Scientific Computing Packages

Some tools that are commonly used

PETSc: Parallel library of matrix and vector data structures, preconditioners, iterative solvers, nonlinear solvers, ODE solvers, GPU support

SLEPc: Parallel library for the solution of large sparse eigenproblems

(par)METIS: Graph Partitioning and Fill-reducing Matrix Ordering

General Scientific Computing Packages

Some tools that are commonly used

PETSc: Parallel library of matrix and vector data structures, preconditioners, iterative solvers, nonlinear solvers, ODE solvers, GPU support

SLEPc: Parallel library for the solution of large sparse eigenproblems

(par)METIS: Graph Partitioning and Fill-reducing Matrix Ordering

(p)BLAS, (SCA)LAPACK: Linear algebra routines

General Scientific Computing Packages

Some tools that are commonly used

PETSc: Parallel library of matrix and vector data structures, preconditioners, iterative solvers, nonlinear solvers, ODE solvers, GPU support

SLEPc: Parallel library for the solution of large sparse eigenproblems

(par)METIS: Graph Partitioning and Fill-reducing Matrix Ordering

(p)BLAS, (SCA)LAPACK: Linear algebra routines

hypre: Parallel library of high performance preconditioners

General Scientific Computing Packages

Some tools that are commonly used

PETSc: Parallel library of matrix and vector data structures, preconditioners, iterative solvers, nonlinear solvers, ODE solvers, GPU support

SLEPc: Parallel library for the solution of large sparse eigenproblems

(par)METIS: Graph Partitioning and Fill-reducing Matrix Ordering

(p)BLAS, (SCA)LAPACK: Linear algebra routines

hypre: Parallel library of high performance preconditioners

SUNDIALS: Parallel library of ODE solvers

General Scientific Computing Packages

Some tools that are commonly used

PETSc: Parallel library of matrix and vector data structures, preconditioners, iterative solvers, nonlinear solvers, ODE solvers, GPU support

SLEPc: Parallel library for the solution of large sparse eigenproblems

(par)METIS: Graph Partitioning and Fill-reducing Matrix Ordering

(p)BLAS, (SCA)LAPACK: Linear algebra routines

hypre: Parallel library of high performance preconditioners

SUNDIALS: Parallel library of ODE solvers

SuiteSparse: GraphBLAS, LU, Cholesky, QR, Matrix reorderings,...
MATLAB uses many parts of this package.

Finite Element Packages

A NON-comprehensive list -everybody seems to have their packages!

Finite Element Packages

A NON-comprehensive list -everybody seems to have their packages!

Fluidity: Eulerian and Lagrangian description, main focus on fluid dynamics, includes DG. Open source

Finite Element Packages

A NON-comprehensive list -everybody seems to have their packages!

Fluidity: Eulerian and Lagrangian description, main focus on fluid dynamics, includes DG. Open source

OOFEM: Object oriented FEM solver, includes XFEM, open source, has full restart support

Finite Element Packages

A NON-comprehensive list -everybody seems to have their packages!

Fluidity: Eulerian and Lagrangian description, main focus on fluid dynamics, includes DG. Open source

OOFEM: Object oriented FEM solver, includes XFEM, open source, has full restart support

FEniCS: open source, python based, well-developed, easy to use, includes DG and a high performance version

Finite Element Packages

A NON-comprehensive list -everybody seems to have their packages!

Fluidity: Eulerian and Lagrangian description, main focus on fluid dynamics, includes DG. Open source

OOFEM: Object oriented FEM solver, includes XFEM, open source, has full restart support

FEniCS: open source, python based, well-developed, easy to use, includes DG and a high performance version

DEAL.II: open source, well-developed, includes HDG and many others

Finite Element Packages

A NON-comprehensive list -everybody seems to have their packages!

Fluidity: Eulerian and Lagrangian description, main focus on fluid dynamics, includes DG. Open source

OOFEM: Object oriented FEM solver, includes XFEM, open source, has full restart support

FEniCS: open source, python based, well-developed, easy to use, includes DG and a high performance version

DEAL.II: open source, well-developed, includes HDG and many others

MFEM: open source, rather recent, HDG is being developed(by us), modular approach with emphasis on exascale computing

Finite Element Packages

A NON-comprehensive list -everybody seems to have their packages!

Fluidity: Eulerian and Lagrangian description, main focus on fluid dynamics, includes DG. Open source

OOFEM: Object oriented FEM solver, includes XFEM, open source, has full restart support

FEniCS: open source, python based, well-developed, easy to use, includes DG and a high performance version

DEAL.II: open source, well-developed, includes HDG and many others

MFEM: open source, rather recent, HDG is being developed(by us), modular approach with emphasis on exascale computing

FreeFEM++: open source, developed on and off, has own language, has good scalability results, includes RT0 and DG

Few (unavailable) MATLAB based HDG packages

Quick google search reveals a lot of papers with sentences like;

Few (unavailable) MATLAB based HDG packages

Quick google search reveals a lot of papers with sentences like;

"We have developed our MATLAB code on top iFEM package..."

Few (unavailable) MATLAB based HDG packages

Quick google search reveals a lot of papers with sentences like;

"We have developed our MATLAB code on top iFEM package..."

"We used our MATLAB code for these tests..."

Few (unavailable) MATLAB based HDG packages

Quick google search reveals a lot of papers with sentences like;

"We have developed our MATLAB code on top iFEM package..."

"We used our MATLAB code for these tests..."

"We have implemented this method in MATLAB as a proof of concept..."

Few (unavailable) MATLAB based HDG packages

Quick google search reveals a lot of papers with sentences like;

"We have developed our MATLAB code on top iFEM package..."

"We used our MATLAB code for these tests..."

"We have implemented this method in MATLAB as a proof of concept..."

None available publicly. Another common point,

Few (unavailable) MATLAB based HDG packages

Quick google search reveals a lot of papers with sentences like;

"We have developed our MATLAB code on top iFEM package..."

"We used our MATLAB code for these tests..."

"We have implemented this method in MATLAB as a proof of concept..."

None available publicly. Another common point, small problem sizes

Two available MATLAB based HDG packages

Deeper look reveals,

Two available MATLAB based HDG packages

Deeper look reveals,

HDG3D - http://www.math.udel.edu/~fjsayas/HDG3D/

I based my implementation upon this one.

Two available MATLAB based HDG packages

Deeper look reveals,

HDG3D - http://www.math.udel.edu/~fjsayas/HDG3D/

I based my implementation upon this one.

FESTUNG - https://github.com/FESTUNG/project

Only implements advection

Two available MATLAB based HDG packages

Deeper look reveals,

HDG3D - http://www.math.udel.edu/~fjsayas/HDG3D/

I based my implementation upon this one.

FESTUNG - https://github.com/FESTUNG/project

Only implements advection

Both implement LDG-H

What did I do?

Started from HDG3D to avoid

- creating a mesh structure
- writing quadrature rules
- implementing basis functions

Mesh maketh the FEM package

```
T =
    coordinates: [48x3 double]
    elements: [108x4 double]
    dirichlet: [36x3 double]
    neumann: [48x3 double]
    faces: [258x4 double]
    dirfaces: [1x36 double]
    neufaces: [1x48 double]
    facebyele: [108x4 double]
    orientation: [108x4 double]
    perm: [108x4 double]
    volume: [108x1 double]
    area: [258x1 double]
    normals: [108x12 double]
```

Quadratures - Volume Integrals

On the reference element $\hat{K}$,

$$\int_{\hat{K}} \phi = \frac{1}{6} \sum_q \omega_q \phi(p_q)$$

Quadratures - Volume Integrals

On the reference element $\hat{K}$,

$$\int_{\hat{K}} \phi = \frac{1}{6} \sum_{q} \omega_q \phi(p_q)$$

Given tetrahedron (v_1, v_2, v_3, v_4), transformation from the reference element to a general element,

$$F_K(x) = B_K x + v_1$$

Quadratures - Volume Integrals

On the reference element $\hat{K}$,

$$\int_{\hat{K}} \phi = \frac{1}{6} \sum_{q} \omega_q \phi(p_q)$$

Given tetrahedron (v_1, v_2, v_3, v_4), transformation from the reference element to a general element,

$$F_K(x) = B_K x + v_1$$

On general tetrahedra

$$\int_K \phi = |B_K| \int_{\hat{K}} \phi \circ F_K = |K| \sum_q \omega_q \phi(F_K(p_q))$$

Quadratures - Volume Integrals

On the reference element $\hat{K}$,

$$\int_{\hat{K}} \phi = \frac{1}{6} \sum_q \omega_q \phi(p_q)$$

Given tetrahedron (v_1, v_2, v_3, v_4), transformation from the reference element to a general element,

$$F_K(x) = B_K x + v_1$$

On general tetrahedra

$$\int_K \phi = |B_K| \int_{\hat{K}} \phi \circ F_K = |K| \sum_q \omega_q \phi(F_K(p_q))$$

Construct F_K s.t. $|B_K| = 6|K|$.

Quadratures - Volume Integrals

An example,

$$\int_K f\phi = 6|K| \int_{\hat{K}} \phi \circ F_K = |K| \sum_q f(F_K(p_q))\omega_q \phi(F_K(p_q))$$

Rewrite in more MATLAB friendly notation

$$vol^T \odot \left((w \odot P)^T f(X, Y, Z)\right)$$

Quadratures - Volume Integrals

An example,

$$\int_K f\phi = 6|K| \int_{\hat{K}} \phi \circ F_K = |K| \sum_q f(F_K(p_q))\omega_q \phi(F_K(p_q))$$

Rewrite in more MATLAB friendly notation

$$vol^T \odot \left((w \odot P)^T f(X, Y, Z)\right)$$

where $(u^T \odot A)_{ij} = u_j A_{ij}$

Quadratures - Volume Integrals

An example,

$$\int_K f\phi = 6|K| \int_{\hat{K}} \phi \circ F_K = |K| \sum_q f(F_K(p_q))\omega_q \phi(F_K(p_q))$$

Rewrite in more MATLAB friendly notation

$$vol^T \odot \left((w \odot P)^T f(X, Y, Z)\right)$$

where $(u^T \odot A)_{ij} = u_j A_{ij}$ and $(u \odot A)_{ij} = u_i A_{ij}$.

Quadratures - Volume Integrals

$$vol^T \odot \left((w \odot P)^T f(X, Y, Z)\right)$$

```
P=basisf3d(2*xhat-1,2*yhat-1,2*zhat-1,k);
wP=bsxfun(@times,weights,P);

Ints=bsxfun(@times,T.volume',wP'*f(x,y,z));
```

Highlights: `bsxfun`, dot product `*`

Quadratures - Face Integrals

On the reference element $\hat{F}$,

$$\int_{\hat{F}} \phi = \frac{1}{2} \sum_q \omega_q \phi(p_q)$$

Quadratures - Face Integrals

On the reference element $\hat{F}$,

$$\int_{\hat{F}} \phi = \frac{1}{2} \sum_q \omega_q \phi(p_q)$$

Given triangle $e = (w_1, w_2, w_3)$, parametrization from the reference face to a general face,

$$\varphi(s,t) = s(w_2 - w_1) + t(w_3 - w_1) + w_1, \varphi : \hat{F} \to e, (s,t) \in \hat{F}$$

Quadratures - Face Integrals

On the reference element $\hat{F}$,

$$\int_{\hat{F}} \phi = \frac{1}{2} \sum_q \omega_q \phi(p_q)$$

Given triangle $e = (w_1, w_2, w_3)$, parametrization from the reference face to a general face,

$$\varphi(s,t) = s(w_2 - w_1) + t(w_3 - w_1) + w_1, \varphi : \hat{F} \to e, (s,t) \in \hat{F}$$

Note that, $|\partial_s \varphi \times \partial_t \varphi| = 2|e|$. On general triangle

$$\int_e \phi = 2|e| \int_{\hat{K}} \phi \circ \varphi = |e| \sum_q \omega_q \phi(\varphi(p_q)).$$

Quadrature - Face Integrals

An example, $< (h_K)^{-1}\sigma\kappa\lambda, \mu >_e$,

$$\frac{\sigma\kappa}{h_K}\int_e \lambda\mu = 2\sigma\kappa\int_F \lambda\circ\varphi\mu\circ\varphi = \sum_q \omega_q\lambda(\varphi(p_q))\mu(\varphi(p_q))$$

Rewrite in more MATLAB friendly notation, but with a loop over faces,

$$\kappa\sum_{l=1}^{4}\sigma_l^T\otimes\left(\left(\omega\odot P_l\right)^T P_l\right), \quad \sigma_l^T = \text{row}(\sigma, l)$$

Quadrature - Face Integrals

$$\kappa \sum_{l=1}^{4} \sigma_l^T \otimes \left(\left(\omega \odot P_l \right)^T P_l \right), \quad \sigma_l^T = \mathrm{row}(\sigma, l)$$

```
d=basisf2d(2*s-1,2*t-1,k);
dweights=bsxfun(@times,d,weights);
dwd=dweights'*d;
HDGInterface=zeros(4*d2,4*d2,Nelts);
for l=1:4
HDGInterface(block2(l),block2(l),:)=reshape(kron(sigmakappa(l,:),dwd),...
                                                  [d2,d2,Nelts]);
end
```

Highlights: `kron`, `:`

Quadrature - Face Integrals

There are two other face integrals, namely,

$$< \kappa \nabla u \cdot n, \mu >_e - < (h_K)^{-1} \sigma \kappa u, \mu >_e$$

and

$$- < u, \kappa \nabla v \cdot n >_{\partial K} - < \kappa \nabla u \cdot n, v >_{\partial K} + < (h_K)^{-1} \sigma \kappa u, v >_{\partial K} .$$

First one: HDGMixedDiffusion

Second one: HDGDiffusion

Both are similar to above example, just more complicated.

Local Solvers - Equation

Consider reaction-diffusion equation with all Dirichlet b.c.s,

$$\begin{aligned} u - \kappa \Delta u &= f, \quad \text{in } \Omega \\ u &= g_D, \quad \text{on } \partial\Omega \end{aligned}$$

Local Solvers - Weak Form

Find $(u, \lambda) \in V_h \times M_h$ s.t. $\forall (v, \mu) \in V_h \times M_h$,

$$(\kappa \nabla u, \nabla v)_\Omega - < u, \kappa \nabla v \cdot n >_{\partial\Omega} + < \lambda, \kappa \nabla v \cdot n >_{\partial\Omega}$$
$$- < \frac{\alpha}{h_K} \kappa \lambda, v >_{\partial\Omega} + < -\kappa \nabla u \cdot \vec{n} + \frac{\alpha}{h_K} \kappa u, v >_{\partial\Omega} + (u, v)_\Omega = (f, v)_\Omega,$$

and,

$$-\left(< -\kappa \nabla u \cdot \vec{n} + \frac{\alpha}{h_K} \kappa u, \mu >_{\partial\Omega} - < \frac{\alpha}{h_K} \kappa \lambda, \mu >_{\partial\Omega} \right) = 0.$$

Local Solvers - Implementation

```
%  Create the element matrices
[HDGDiffusion,HDGMixedDiff,HDGInterface]=...
           matricesFace(T,sigma,kappa,k,formulas{3});
diffusion=Diffusion(T,k,kappa,formulas{1});
mass=MassMatrix(T,k,formulas{1});
```

Local Solvers - Implementation

```
%  Create the element matrices
[HDGDiffusion,HDGMixedDiff,HDGInterface]=...
          matricesFace(T,sigma,kappa,k,formulas{3});
diffusion=Diffusion(T,k,kappa,formulas{1});
mass=MassMatrix(T,k,formulas{1});
```

`diffusion`, `mass` and `HDGDiffusion` are of size $n_{eu} \times n_{eu} \times n_{el}$,

Local Solvers - Implementation

```
%  Create the element matrices
[HDGDiffusion,HDGMixedDiff,HDGInterface]=...
          matricesFace(T,sigma,kappa,k,formulas{3});
diffusion=Diffusion(T,k,kappa,formulas{1});
mass=MassMatrix(T,k,formulas{1});
```

`diffusion`, `mass` and `HDGDiffusion` are of size $n_{eu} \times n_{eu} \times n_{el}$,

`HDGMixedDiff` is of size $4n_{fu} \times n_{eu} \times n_{el}$

Local Solvers - Implementation

```
%  Create the element matrices
[HDGDiffusion,HDGMixedDiff,HDGInterface]=...
          matricesFace(T,sigma,kappa,k,formulas{3});
diffusion=Diffusion(T,k,kappa,formulas{1});
mass=MassMatrix(T,k,formulas{1});
```

`diffusion`, `mass` and `HDGDiffusion` are of size $n_{eu} \times n_{eu} \times n_{el}$,

`HDGMixedDiff` is of size $4n_{fu} \times n_{eu} \times n_{el}$ and `HDGInterface` is of size

$4n_{fu} \times 4n_{fu} \times n_{el}$.

Local Solvers - Implementation

```
% A = [[ UV LV ]      b = [ f
%      [ UM LM ]],          0 ]
UV = mass + diffusion + HDGDiffusion;
UM = HDGMixedDiff;
LV = permute(UM,[2 1 3]);
LM = HDGInterface;

f=testElem(f,T,k,formulas{1});
```

Local Solvers - Implementation

```
for i=1:Nelts
  A(:,:,i)= LM(:,:,i)-UM(:,:,i)*inv(UV(:,:,i))*LV(:,:,i);
  b(:,i) = -UM(:,:,i)*inv(UV(:,:,i))*f(:,i);
end
```

Local Solvers - Implementation

```
for i=1:Nelts
  A(:,:,i)= LM(:,:,i)-UM(:,:,i)*inv(UV(:,:,i))*LV(:,:,i);
  b(:,i) = -UM(:,:,i)*inv(UV(:,:,i))*f(:,i);
end
```

Loop over elements? Isn't it bad?

Local Solvers - Implementation

```
for i=1:Nelts
  A(:,:,i)= LM(:,:,i)-UM(:,:,i)*inv(UV(:,:,i))*LV(:,:,i);
  b(:,i) = -UM(:,:,i)*inv(UV(:,:,i))*f(:,i);
end
```

Loop over elements? Isn't it bad? Yes.

Local Solvers - Implementation

```
for i=1:Nelts
  A(:,:,i)= LM(:,:,i)-UM(:,:,i)*inv(UV(:,:,i))*LV(:,:,i);
  b(:,i) = -UM(:,:,i)*inv(UV(:,:,i))*f(:,i);
end
```

Loop over elements? Isn't it bad? Yes.

Easily parallelizable. How?

Local Solvers - Implementation

```
for i=1:Nelts
  A(:,:,i)= LM(:,:,i)-UM(:,:,i)*inv(UV(:,:,i))*LV(:,:,i);
  b(:,i) = -UM(:,:,i)*inv(UV(:,:,i))*f(:,i);
end
```

Loop over elements? Isn't it bad? Yes.

Easily parallelizable. How? Write `parfor` instead of `for`.

Assembly

```
%  Assemble the system
A=sparse(R(:),C(:),A(:));
phif=accumarray(RowsRHS,b(:));

uhatD=BC3d(uD,T,k,formulas{3}); % Dirichlet B.C.

%Dirichlet BC
Uhatv=zeros(d2*Nfaces,1);
Uhatv(dirfaces)=uhatD;       %uhat stored as a vector: d2*Nfaces
```

Assembly

```
%  Assemble the system
A=sparse(R(:),C(:),A(:));
phif=accumarray(RowsRHS,b(:));

uhatD=BC3d(uD,T,k,formulas{3}); % Dirichlet B.C.

%Dirichlet BC
Uhatv=zeros(d2*Nfaces,1);
Uhatv(dirfaces)=uhatD;       %uhat stored as a vector: d2*Nfaces
```

Highlights: `sparse` and `accumarray`

Assembly

```
%RHS
rhs=zeros(d2*Nfaces,1);
rhs(free)=phif(free);
rhs=rhs-A(:,dirfaces)*Uhatv(dirfaces);

% Solve the system
Uhatv(free)=A(free,free)\rhs(free);
Uhat=reshape(Uhatv,d2,Nfaces);
```

Assembly

```
% Reconstruct the solution
faces=T.facebyele'; faces=faces(:);
uhhataux=reshape(Uhat(:,faces),[4*d2,Nelts]);
Uh=zeros(d3,Nelts);
for K=1:Nelts
Uh(:,K)=inv(UV(:,:,K))*(f(:,K)-LV(:,:,K)*uhhataux(:,K));
end
```

Loop over elements again, but it is not a problem.

Demonstration

This slide is left blank intentionally.

Thanks for listening!

Any questions?