deal.II Implementation of a Weak Galerkin Finite Element Solver for Darcy Flow *

Zhuoran Wang[1], Graham Harper[1], Patrick O'Leary[2], Jiangguo Liu[1], and Simon Tavener[1]

[1] Colorado State University, Fort Collins, CO 80523, USA, {wangz,harper,liu,tavener}@math.colostate.edu
[2] Kitware, Inc., Santa Fe, NM 87507, USA, patrick.oleary@kitware.com

Abstract. This paper presents a weak Galerkin (WG) finite element solver for Darcy flow and its implementation on the deal.II platform. The solver works for quadrilateral and hexahedral meshes in a unified way. It approximates pressure by Q-type degree $k(\geq 0)$ polynomials separately in element interiors and on edges/faces. Numerical velocity is obtained in the unmapped Raviart-Thomas space $RT_{[k]}$ via postprocessing based on the novel concepts of discrete weak gradients. The solver is locally mass-conservative and produces continuous normal fluxes. It is implemented in deal.II in the dimension-independent paradigm and allows polynomial degrees up to 5. This is the first attempt of implementing WG on a popular finite element package. Numerical experiments show that our new WG solver performs better than the classical mixed finite element methods.

Keywords: Darcy flow · deal.II · finite element methods · hexahedral meshes · quadrilateral meshes · weak Galerkin

1 Introduction

The Darcy equation, although simple, plays an important role for modeling flow in porous media. The equation usually takes the following form

$$\begin{cases} \nabla \cdot (-\mathbf{K}\nabla p) + c\,p = f, \quad \mathbf{x} \in \Omega, \\ p|_{\Gamma^D} = p_D, \quad ((-\mathbf{K}\nabla p)\cdot \mathbf{n})|_{\Gamma^N} = u_N, \end{cases} \tag{1}$$

where Ω is a 2-dim or 3-dim bounded domain, p is the unknown pressure, $\mathbf{K}$ is a conductivity matrix that is uniformly symmetric positive definite (SPD), c is a known function, f is a known source term, p_D is a Dirichlet boundary condition, u_N is a Neumann boundary condition, and $\mathbf{n}$ is the outward unit normal vector on $\partial\Omega$, which has a nonoverlapping decomposition $\Gamma^D \cup \Gamma^N$.

* G. Harper, J. Liu, and Z. Wang were partially supported by US National Science Foundation under grant DMS-1819252. All authors sincerely thank Prof. Wolfgang Bangerth for providing computing resources and advice for this project.

The elliptic boundary value problem (1) can be solved by many types of finite element methods. But in the context of Darcy flow, *local mass conservation* and *normal flux continuity* are two most important properties to be respected by finite element solvers.

- The continuous Galerkin (CG) methods [5] use the least degrees of freedom but do not possess these two properties and hence cannot be used directly. Several post-processing procedures have been developed [6, 7].
- Discontinuous Galerkin (DG) methods have the desired locality and hence are locally conservative. DG methods gain normal flux continuity after post-processing [4].
- The enhanced Galerkin (EG) methods [18] possess both properties but need to handle some minor issues in implementation.
- The mixed finite element methods (MFEMs) [2, 9] have both properties by design but result in indefinite discrete linear systems, for which hybridization needs to be employed to convert them into definite linear systems.
- The weak Galerkin (WG) methods [13, 19] possess both properties and result in SPD linear systems that are easier to solve [11, 15, 16].

In this paper, we investigate efficient implementation of WG Darcy solvers in `deal.II`, a popular finite element package [3], with the intention to make WG finite element methods practically useful for large-scale scientific computation.

2 A WG Finite Element Solver for Darcy Flow

WG solvers for Darcy flow can be developed for simplicial, quadrilateral or hexahedral, and more general polygonal or polyhedral meshes. These finite element schemes may or may not contain a stabilization term, depending on choices of the approximating polynomials for pressure in element interiors and on edges/faces. Through integration by parts, these polynomial basis functions are used for computation of discrete weak gradients, which are used to approximate the classical gradient in the variational form for the Darcy equation. Discrete weak gradients can be established in a general vector polynomial space [17] or a specific one like the Raviart-Thomas space [11, 16] that has desired approximation properties.

In this paper, we focus on quadrilateral and hexahedral meshes, for which faces are flat or very close to being flat. We use $Q_k(k \geq 0)$-type polynomials in element interiors and on edges/faces for approximating the primal variable pressure. Their discrete weak gradients are established in the local unmapped Raviart-Thomas $RT_{[k]}(k \geq 0)$ spaces, for which we do not use the Piola transformation. We use the same form of polynomials as that for rectangles and bricks in the classical MFEMs [9].

To illustrate these new concepts, let us consider a convex quadrilateral E with center (x_c, y_c). We define the local unmapped Raviart-Thomas space $RT_{[0]}(E)$ as

$$RT_{[0]}(E) = \text{Span}(\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4), \tag{2}$$

where

$$\mathbf{w}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{w}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{w}_3 = \begin{bmatrix} X \\ 0 \end{bmatrix}, \quad \mathbf{w}_4 = \begin{bmatrix} 0 \\ Y \end{bmatrix}, \tag{3}$$

and $X = x - x_c$, $Y = y - y_c$ are the normalized coordinates.

Now we introduce a new concept of 5 discrete weak functions $\phi_i (0 \le i \le 4)$.

- ϕ_0 is for element interior: It takes value 1 in the interior E° but 0 on the boundary E^∂ (all 4 edges);
- $\phi_1, \phi_3, \phi_3, \phi_4$ are for the four sides respectively: $\phi_i (1 \le i \le 4)$ takes value 1 on the i-th edge but 0 on all other three edges and in the interior.

Any such function ϕ has two independent parts: ϕ° is defined in E°, whereas ϕ^∂ is defined on E^∂, together written as $\phi = \{\phi^\circ, \phi^\partial\}$. Its discrete weak gradient $\nabla_{w,d}\phi$ is specified in $RT_{[0]}(E)$ via integration by parts [19]:

$$\int_E (\nabla_{w,d}\phi) \cdot \mathbf{w} = \int_{E^\partial} \phi^\partial (\mathbf{w} \cdot \mathbf{n}) - \int_{E^\circ} \phi^\circ (\nabla \cdot \mathbf{w}), \qquad \forall \mathbf{w} \in RT_{[0]}(E). \tag{4}$$

Implementationwise this attributes to solving a size-4 SPD linear system.

When the quadrilateral is simply a rectangle $E = [x_1, x_2] \times [y_1, y_2]$, we have

$$\begin{cases} \nabla_{w,d}\phi_0 = \quad 0\mathbf{w}_1 \; + \quad 0\mathbf{w}_2 \; + \frac{-12}{(\Delta x)^2}\mathbf{w}_3 \; + \frac{-12}{(\Delta y)^2}\mathbf{w}_4, \\ \nabla_{w,d}\phi_1 = \frac{-1}{\Delta x}\mathbf{w}_1 \; + \quad 0\mathbf{w}_2 \; + \frac{6}{(\Delta x)^2}\mathbf{w}_3 \; + \quad 0\mathbf{w}_4, \\ \nabla_{w,d}\phi_2 = \frac{1}{\Delta x}\mathbf{w}_1 \; + \quad 0\mathbf{w}_2 \; + \frac{6}{(\Delta x)^2}\mathbf{w}_3 \; + \quad 0\mathbf{w}_4, \\ \nabla_{w,d}\phi_3 = \quad 0\mathbf{w}_1 \; + \frac{-1}{\Delta y}\mathbf{w}_2 \; + \quad 0\mathbf{w}_3 \; + \frac{6}{(\Delta y)^2}\mathbf{w}_4, \\ \nabla_{w,d}\phi_4 = \quad 0\mathbf{w}_1 \; + \frac{1}{\Delta y}\mathbf{w}_2 \; + \quad 0\mathbf{w}_3 \; + \frac{6}{(\Delta y)^2}\mathbf{w}_4, \end{cases} \tag{5}$$

where $\Delta x = x_2 - x_1, \Delta y = y_2 - y_1$.

Here are some remarks.

(i) For a quadrilateral, ϕ° or ϕ^∂ each can also be a degree $k \ge 1$ polynomial and the discrete weak gradient $\nabla_{w,d}\phi$ is then established in the local unmapped Raviart-Thomas space $RT_{[k]} (k \ge 1)$.
(ii) For a hexahedron with nonflat faces, we can use the averaged normal vectors in (4). The Jacobian determinant is still used in computation of the integrals.

Let $\mathcal{E}_h$ be a shape-regular quadrilateral mesh. Let Γ_h^D be the set of all edges on the Dirichlet boundary Γ^D and Γ_h^N be the set of all edges on the Neumann boundary Γ^N. Let S_h be the space of discrete shape functions on $\mathcal{E}_h$ that are degree k polynomials in element interiors and also degree k polynomials on edges. Let S_h^0 be the subspace of functions in S_h that vanish on Γ_h^D. For (1), we seek $p_h = \{p_h^\circ, p_h^\partial\} \in S_h$ such that $p_h^\partial|_{\Gamma_h^D} = Q_h^\partial(p_D)$ (the L^2-projection of Dirichlet boundary data into the space of degree k polynomials on Γ_h^D) and

$$\mathcal{A}_h(p_h, q) = \mathcal{F}(q), \qquad \forall q = \{q^\circ, q^\partial\} \in S_h^0, \tag{6}$$

where

$$\mathcal{A}_h(p_h, q) = \sum_{E\in\mathcal{E}_h} \int_E \mathbf{K}\nabla_{w,d} p_h \cdot \nabla_{w,d} q + \sum_{E\in\mathcal{E}_h} \int_E c\, p\, q \tag{7}$$

and

$$\mathcal{F}(q) = \sum_{E\in\mathcal{E}_h} \int_E f q^\circ - \sum_{\gamma\in\Gamma_h^N} \int_\gamma u_N q^\partial. \tag{8}$$

This results in symmetric positive-definite discrete linear systems [16].

Although $\nabla_{w,d} p_h$ is locally in the Raviart-Thomas space, $-\mathbf{K}\nabla_{w,d} p_h$ may not be. A local L^2-projection $\mathbf{Q}_h$ is needed [11, 13, 16] to get this back into the RT space $RT_{[k]}$:

$$\mathbf{u}_h = \mathbf{Q}_h(-\mathbf{K}\nabla_{w,d} p_h). \tag{9}$$

This is the numerical Darcy velocity for subsequent applications, e.g., transport simulations. Clearly, this process is readily parallelizable for large-scale computation. This numerical velocity is locally mass-conservative and the corresponding normal flux is continuous across edges or faces, as proved in [11, 16].

As shown in [16], this Darcy solver is easy to be implemented and results in a symmetric positive-definite system that can be easily solved by a conjugate-gradient type linear solver.

The weak Galerkin methodology has connections to but is also different from the classical mixed finite element methods, especially the hybridized MFEMs [13, 14].

3 `deal.II` Implementation of WG Solver for Darcy Flow

`deal.II` is a popular C++ finite element package for solving partial differential equations [3]. It uses quadrilateral and hexahedral meshes instead of the simplicial (triangular or tetrahedral) meshes. Generally speaking, the former actually involve less degrees of freedom than the latter. The resulting global discrete linear systems may have smaller sizes, although the setup time for the global discrete linear systems might be longer. The setup time is spent on bilinear or trilinear mappings from the reference square/cube to generic quadrilaterals and hexahedra and computation of various integrals.

3.1 Quadrilateral and Hexahedral Meshes

`deal.II` handles meshes by the **GridGenerator** class. All the mesh information, such as the number of active cells, degrees of freedom, are stored in this class. Refining a mesh is a common job in numerical experiments when one needs to compute convergence rates. As previously discussed, for any integer $k \geq 0$, our WG$(Q_k, Q_k; RT_{[k]})$ solver is locally mass-conservative and produces continuous normal fluxes regardless of the quality of the quadrilateral and cuboidal hexahedral meshes. But to obtain the desired order k convergence rate in pressure, velocity, and normal fluxes, we require meshes to be asymptotically parallelogram or parallelopiped [11, 16].

3.2 Finite Element Spaces

The WG($Q_k, Q_k; RT_{[k]}$) solver involves three finite element spaces. The first two polynomial spaces are for the unknown pressure, whereas the third one (Raviart-Thomas space) is used for discrete weak gradients and numerical velocity. For deal.II implementation, the first two are combined as

```
FESystem<dim> fe;
```

The third one is

```
FE_RaviartThomas<dim> fe_rt;
```

where dim could be 2 or 3.

Raviart-Thomas Spaces for Discrete Weak Gradients and Velocity. The weak Galerkin framework allows us to skip the Piola transformation and use the unmapped Raviart-Thomas Spaces on generic quadrilaterals and hexahedra [11, 16]. Note that these spaces use the same polynomials for shape functions as those in the classical Raviart-Thomas spaces for 2-dim or 3-dim rectangles [9]. They are respectively,

$$RT_{[k]}(E) = Q_{k+1,k} \times Q_{k,k+1}, \tag{10}$$

$$RT_{[k]}(E) = Q_{k+1,k,k} \times Q_{k,k+1,k} \times Q_{k,k,k+1}. \tag{11}$$

In deal.II, we use degree for k in equation (10) or (11) and have

```
fe_rt(degree);
```

Later on, for weak Galerkin shape functions for the pressure in element interiors and on interelement edges/faces, we will establish their discrete weak gradients in the above Raviart-Thomas spaces. Furthermore, the numerical velocity will also be computed in these RT spaces through a local L^2-projection.

Two Separate Polynomial Spaces for Pressure. Note that for the WG($Q_k, Q_k; RT_{[k]}$) finite element solver for Darcy flow, the pressure is approximated separately in element interiors by Q_k-type polynomials and on edges/faces by Q_k-type polynomials also. Note that the 2nd group of Q_k-type polynomials are defined locally on each edge/face. For the deal.II implementation, we have

```
fe(FE_DGQ<dim>(degree), 1, FE_FaceQ<dim>(degree), 1),
```

where degree is k, that is, the degree of the polynomials, "1" means these two groups of pressure unknowns are just scalars. Note that

- **FE_DGQ** is a finite element class in deal.II that has no continuity across faces and vertices, i.e., every shape function lives exactly in one cell. So we use it to approximate the pressure in element interiors.
- On the contrary, **FE_FaceQ** is a finite element class that is defined only on edges/faces.

However, these two different finite element spaces are combined into one finite element system, we split these shape functions as

```
const FEValuesExtractors::Scalar interior(0);
const FEValuesExtractors::Scalar face(1);
```

Here "0" corresponds to the 1st finite element class **FE_DGQ** for the interior pressure; "1" corresponds to the 2nd finite element class **FE_FaceQ** for the face pressure. Later on, we will just use **fe_values[interior].value** and **fe_values[face].value** for assembling the element-level matrices.

3.3 Gaussian Quadratures

Finite element computation involves various types of integrals, which are all discretized via quadratures. Gaussian quadratures are common choices. For example, we consider

$$\int_E f \approx \sum_{k=1}^{K} w_k f(x_k, y_k) J_k, \tag{12}$$

where K is the number of quadrature points, (x_k, y_k) is the k-th quadrature point, J_k is the corresponding Jacobian determinant, and w_k is the weight. In `deal.II`, this is handled by the **Quadrature** class. In particular, the Jacobian determinant value and weight for each quadrature point are bundled together as

```
fe_values.JxW(q_k),
```

where q_k is the k-th quadrature point.

3.4 Linear Solvers

`deal.II` provides a variety of linear solvers that are inherited from `PETSc`. The global discrete linear systems obtained from the weak Galerkin finite element discretization of the Darcy equation are symmetric positive-definite. Thus we can choose a conjugate-gradient type linear solver for them.

3.5 Graphics Output

In our WG(Q_k, Q_k) discretization for the Darcy equation, scalar pressures are separately defined in element interiors and on edges/faces of a given mesh. These values are therefore output separately in `deal.II`. The interior pressures are handled by **DataOut**, whereas the face pressures are handled by **DataOutFace**. Specifically,

```
data_out.build_patches(fe.degree);
data_out_face.build_patches(fe.degree);
```

are used to subdivide each cell into smaller patches, which provide better visualization if we use higher degree polynomials. The post-processed data are saved as `vtk` files for later visualization in `VisIt`.

4 Code Excerpts with Comments

This section provides some code excerpts with comments. More details can be found in deal.II tutorial step-61 and subject to minor changes [1].

4.1 Construction of Finite Element Spaces

Note that FE_RaviartThomas is a Raviart-Thomas space for vector-valued functions, FESystem defines the finite element spaces in the interiors and on edges/faces. Shown below is the code for the lowest order WG finite elements.

```
88  FE_RaviartThomas<dim> fe_rt;
89  DoFHandler<dim> dof_handler_rt;
90  FESystem<dim> fe;
91  DoFHandler<dim> dof_handler;
```

```
222 fe_rt (0),
223 dof_handler_rt (triangulation),
224 fe (FE_DGQ<dim>(0), 1, FE_FaceQ<dim>(0), 1),
225 dof_handler (triangulation)
```

4.2 System Setup

The following piece distributes degrees of freedom for finite element spaces.

```
255 dof_handler_rt.distribute_dofs (fe_rt);
256 dof_handler.distribute_dofs (fe);
```

The following piece sets up matrices and vectors in the system.

```
255 DynamicSparsityPattern dsp(dof_handler.n_dofs());
256 DoFTools::make_sparsity_pattern(dof_handler, dsp, constraints);
257 sparsity_pattern.copy_from(dsp);
258 system_matrix.reinit(sparsity_pattern);
259 solution.reinit(dof_handler.n_dofs());
260 system_rhs.reinit(dof_handler.n_dofs());
```

4.3 System Assembly

The following piece uses extractors to extract components of finite element shape functions.

```
const FEValuesExtractors::Vector velocities (0);
const FEValuesExtractors::Scalar interior (0);
const FEValuesExtractors::Scalar face (1);
```

The following pieces calculates the Gram matrix for the RT space.

```
for (unsigned int q = 0; q < n_q_points_rt; ++q) {
  for (unsigned int i = 0; i < dofs_per_cell_rt; ++i) {
    const Tensor<1,dim> phi_i_u =
          fe_values_rt[velocities].value(i,q);
    for (unsigned int j = 0; j < dofs_per_cell_rt; ++j) {
      const Tensor<1,dim>  phi_j_u =
            fe_values_rt[velocities].value (j, q);
            cell_matrix_rt(i,j) += phi_i_u * phi_j_u
                                 * fe_values_rt.JxW(q);
} } }
```

The following piece handles construction of WG local matrices.

```
for (unsigned int q = 0; q < n_q_points_rt; ++q) {
  for (unsigned int i = 0; i<dofs_per_cell; ++i) {
    for (unsigned int j = 0; j<dofs_per_cell; ++j) {
     for (unsigned int k = 0; k<dofs_per_cell_rt; ++k) {
       const Tensor<1,dim> phi_k_u =
             fe_values_rt[velocities].value(k,q);
       for (unsigned int l = 0; l < dofs_per_cell_rt; ++l) {
         const Tensor<1,dim> phi_l_u =
               fe_values_rt[velocities].value(l,q);
               local_matrix(i,j) += coefficient_values[q] *
                   cell_matrix_C[i][k] * cell_matrix_C[j][l] *
                   phi_k_u * phi_l_u * fe_values_rt.JxW(q);
} } } } }
```

The following piece calculates the local right-hand side.

```
for (unsigned int q = 0; q < n_q_points; ++q) {
  for (unsigned int i = 0; i < dofs_per_cell; ++i) {
   cell_rhs(i) += (fe_values[interior].value(i, q) *
     right_hand_side.value(fe_values.quadrature_point(q)) *
     fe_values.JxW(q));
} }
```

The following piece distributes entries of local matrices into the system matrix and also incoporates the local right-hand side into the system right-hand side.

```
cell->get_dof_indices(local_dof_indices);
constraints.distribute_local_to_global(
            local_matrix, cell_rhs, local_dof_indices,
            system_matrix, system_rhs);
```

5 Numerical Experiments

To demonstrate accuracy and robustness of this novel WG solver for Darcy flow, this section presents numerical experiments on three examples, all having $c = 0$ in equation (1).

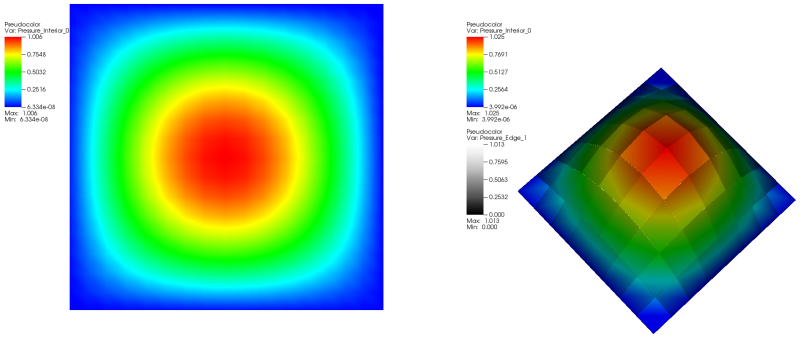
Table 1. Ex.1: Numerical results of WG$(Q_0, Q_0; RT_{[0]})$ solver on rectangular meshes

$1/h$	$\|p - p_h^\circ\|$	Rate	$\|\mathbf{u} - \mathbf{u}_h\|$	Rate	$\|\mathbf{u}\cdot\mathbf{n} - \mathbf{u}_h\cdot\mathbf{n}\|$	Rate
WG$(Q_0, Q_0; RT_{[0]})$						
4	1.5870E-01	—	5.1289E-01	—	7.0500E-01	—
8	7.9980E-02	0.988	2.5309E-01	1.018	3.5523E-01	0.988
16	4.0058E-02	0.997	1.2608E-01	1.005	1.7796E-01	0.997
32	2.0037E-02	0.999	6.2977E-02	1.001	8.9020E-02	0.999
64	1.0020E-02	0.999	3.1481E-02	1.000	4.4516E-02	0.999
128	5.0099E-03	1.000	1.5740E-02	1.000	2.2258E-02	1.000
WG$(Q_1, Q_1; RT_{[1]})$						
4	1.6130E-02	—	5.0989E-02	—	7.1588E-02	—
8	4.0560E-03	1.991	1.2762E-02	1.998	1.8016E-02	1.990
16	1.0155E-03	1.997	3.1915E-03	1.999	4.5113E-03	1.997
32	2.5396E-04	1.999	7.9792E-04	1.999	1.1283E-03	1.999
64	6.3496E-05	1.999	1.9948E-04	1.999	2.8210E-04	1.999
128	1.5874E-05	2.000	4.9871E-05	1.999	7.0528E-05	1.999
WG$(Q_2, Q_2; RT_{[2]})$						
4	1.0719E-03	—	3.3764E-03	—	4.7589E-03	—
8	1.3465E-04	2.992	4.2331E-04	2.995	5.9814E-04	2.992
16	1.6852E-05	2.998	5.2952E-05	2.998	7.4870E-05	2.998
32	2.1072E-06	2.999	6.6203E-06	2.999	9.3620E-06	2.999
64	2.6342E-07	2.999	8.2757E-07	2.999	1.1703E-06	2.999
128	3.2928E-08	2.999	1.0344E-07	2.999	1.46298E-07	2.999

Example 1 (A smooth example for convergence rates). We consider a 2-dim problem with domain $\Omega = (0, 1)^2$, conductivity $\mathbf{K} = \mathbf{I}_2$, and a known analytical solution for the pressure:

$$p(x, y) = \sin(\pi x)\sin(\pi y).$$

The source term is well known to be $f = 2\pi^2 \sin(\pi x)\sin(\pi y)$. A homogeneous Dirichlet boundary condition is posed on the entire boundary.



Element interior pressure for $h = \frac{1}{16}$ Interior/edge pressure 3d view ($h = \frac{1}{8}$)

Fig. 1. Ex.1: Numerical pressure by WG$(Q_1, Q_1; RT_{[1]})$ solver on rectangular meshes.

The new WG$(Q_k, Q_k; RT_{[k]})$ solver is tested on Example 1 for $k = 0, 1, 2$ on a sequence of uniform rectangular meshes. As shown in Table 1, the solver exhibits order k convergence rates for the L^2-norms of the errors in the interior pressure, velocity, and normal flux. Shown in Figure 1 are the profiles of the numerical pressure obtained from applying the WG$(Q_1, Q_1; RT_{[1]})$ solver. In the right panel, the edge pressures are plotted as grey line segments. The graphical results in both panels demonstrate nice monotonicity in the numerical pressure produced by our WG solver.

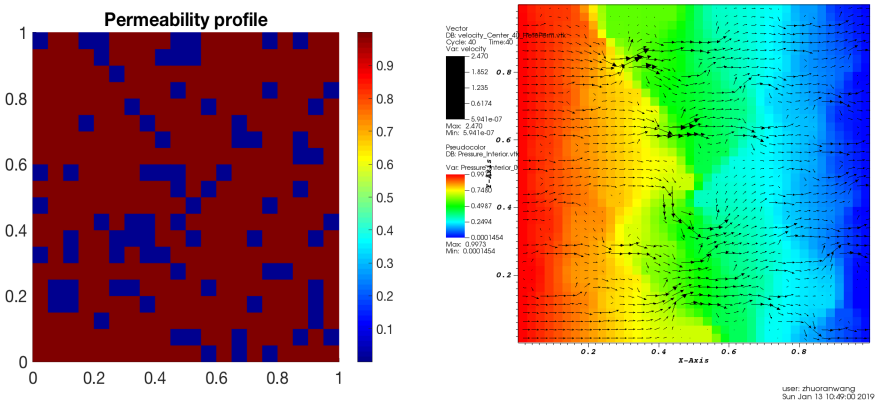


Fig. 2. Example 2 (Heterogeneous permeability): Numerical pressure and velocity for a high-contrast permeability field used in [8].

Table 2. Example 2: Comparison between WG and MFEM solvers

tol $= 10^{-9}$	WG			MFEM		
Mesh	$p_{\min}$	$p$max	Runtime	$p_{\min}$	$p$max	Runtime
20×20	1.21321e-4	0.995113	0.857s	1.21320e-4	0.995113	1.410s
40×40	1.45401e-4	0.997289	6.759s	1.45401e-4	0.997289	13.833s
80×80	8.73042e-5	0.998587	59.070s	8.73043e-5	0.998587	103.141s
160×160	4.59350e-5	0.999281	607.556s	4.59345e-5	0.999281	877.648s

Example 2 (Heterogeneous permeability). The permeability profile in this example is adopted from [8]. We consider a simple Darcy flow problem on the unit square Ω. Dirichelet boundary conditions are posed on the left and right sides: $p = 1$ for $x = 0$; and $p = 0$ for $x = 1$. The other two sides have a homogeneous Neumann (no-flow) boundary condition. The problem was also tested using `Matlab` in [16].

Shown in Figure 2 right panel are the numerical pressure and velocity profiles obtained from apply our WG$(Q_0, Q_0; RT_0)$ solver on a uniform 40×40 rectangular mesh. Clearly, the elementwise numerical pressure stays between 0 and 1, the pressure profile demonstrates monotonicity from left to right, and the velocity profile reflects the low-permeability regions and channels for fast flow.

Of course, the problem can also be solved by `deal.II` built in mixed finite element solvers based on Schur complement. See `deal.II` tutorial step-20. But for the MFEM, Neumann boundary conditions are essential and Dirichlet boundary conditions become natural.

For Example 2, we compare the lowest order WG solver ($k = 0$) with the lowest order MFEM solver on a sequence of rectangular meshes on the same Toshiba laptop. The tolerance for linear solvers is 10^{-9}. Table 2 shows that the WG solver produces very close results with significantly less runtime.

Example 3 (Permeability profile from SPE10 Model 2). This example is adopted from the 10th comparative solution project developed by the Society of Petroleum Engineers (SPE). SPE10 was developed as a benchmark for upscaling methods, but the 2nd dataset of SPE10 is becoming a very popular testcase in the academic community for comparing different numericl methods. The dataset is a 3-dim geo-statistical realization from the Jurassic Upper Brent formations [12]. The model has geometric dimensions 1200 (ft) $\times$ 2200 (ft) $\times$ 170 (ft). The dataset is provided on a $60 \times 220 \times 85$ Cartesian grid, in which each block has a size 20(ft)$\times$10(ft)$\times$10(ft). The top 70 ft or 35 layers represent the shallow-marine Tarbert formation, whereas the bottom 100 ft or 50 layers represent the fluvial Ness formation. The SPE10 model is structurally simple but highly heterogeneous in porosity and permeability. It poses significant challenges to numerical simulators.

The SPE 10 dataset is publicly available at `http://www.spe.org/web/csp/`. The original data assume the z-axis pointing downwards uses a right-hand coordinate system. So a conversion of ordering in blocks is needed for the original

data items. Here we actually use the code in the `Matlab Reservoir Simulation Toolbox` (MRST) [12] to acquire the needed data.

In this paper, we focus on the Darcy flow part. We use the original permeability data and consider a flow problem. Dirichelet boundary conditions are posed on two boundary faces: $p = 1$ for $y = 0$; and $p = 0$ for $y = 1200$. All other four boundary faces have a homogeneous Neumann (no-flow) boundary condition.

We test the new weak Galerkin solver on the coarse, medium, and fine meshes, respectively.

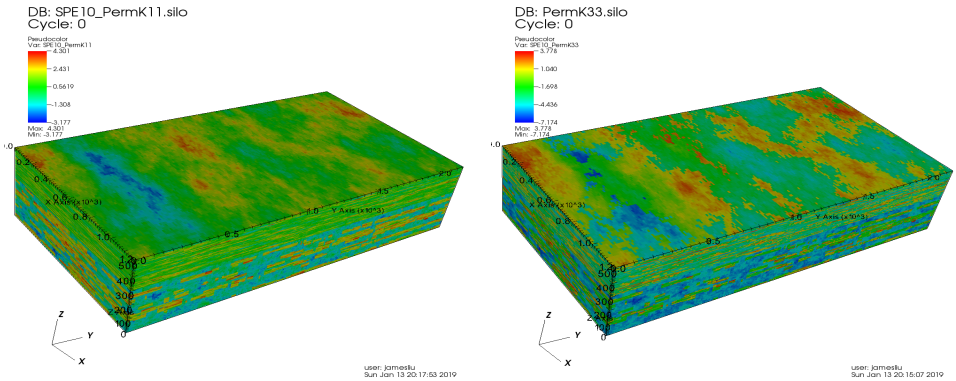
(i) A **coarse mesh** with partitions $12 \times 44 \times 17$. When the WG$(Q_0, Q_0; RT_{[0]})$ solver is used, there are $8,976$ pressure degrees of freedom (DOFs) for the element interiors; $28,408$ pressure DOFs for all the faces, and totally 37384 (about 37K) DOFs. Note that the local $RT_{[0]}$ spaces are used to compute the discrete weak gradients of the pressure basis functions, but they do not constitute any degrees of freedom.

(ii) A **medium mesh** with partitions $30 \times 110 \times 85$. We use WG$(Q_0, Q_0; RT_{[0]})$ again. There are $280,500$ DOFs for the pressure in element interiors, $856,700$ DOFs for all faces, and totally $1,137,200$ (about 1M) DOFs.

(iii) A **fine mesh** with partitions $60 \times 220 \times 85$, which is the same as the original gridblock. Again WG$(Q_0, Q_0; RT_{[0]})$ is used. There are $1,122,000$ interior DOFs; $3,403,000$ face DOFs; and a total $4,525,000$ (about 4M) DOFs.

We tripled the z-axis for a better visualization, i.e., the geometric dimensions are $1200\,(\text{ft}) \times 2200\,(\text{ft}) \times 510\,(\text{ft})$.

As shown in Figure 4, the coarse mesh is too coarse to reveal the geological features of the reservior. The result on the medium mesh is good enough to reflect the channel features of the fluvial Ness formation. The fine mesh result is smoother and exposes further details about the heterogeneity, but it requires



(a) Permeability $K_{11} = K_{22}$ (a) Permeability K_{33}

Fig. 3. Example 3 (SPE10 Model 2): Permeability profiles on log_{10} scales

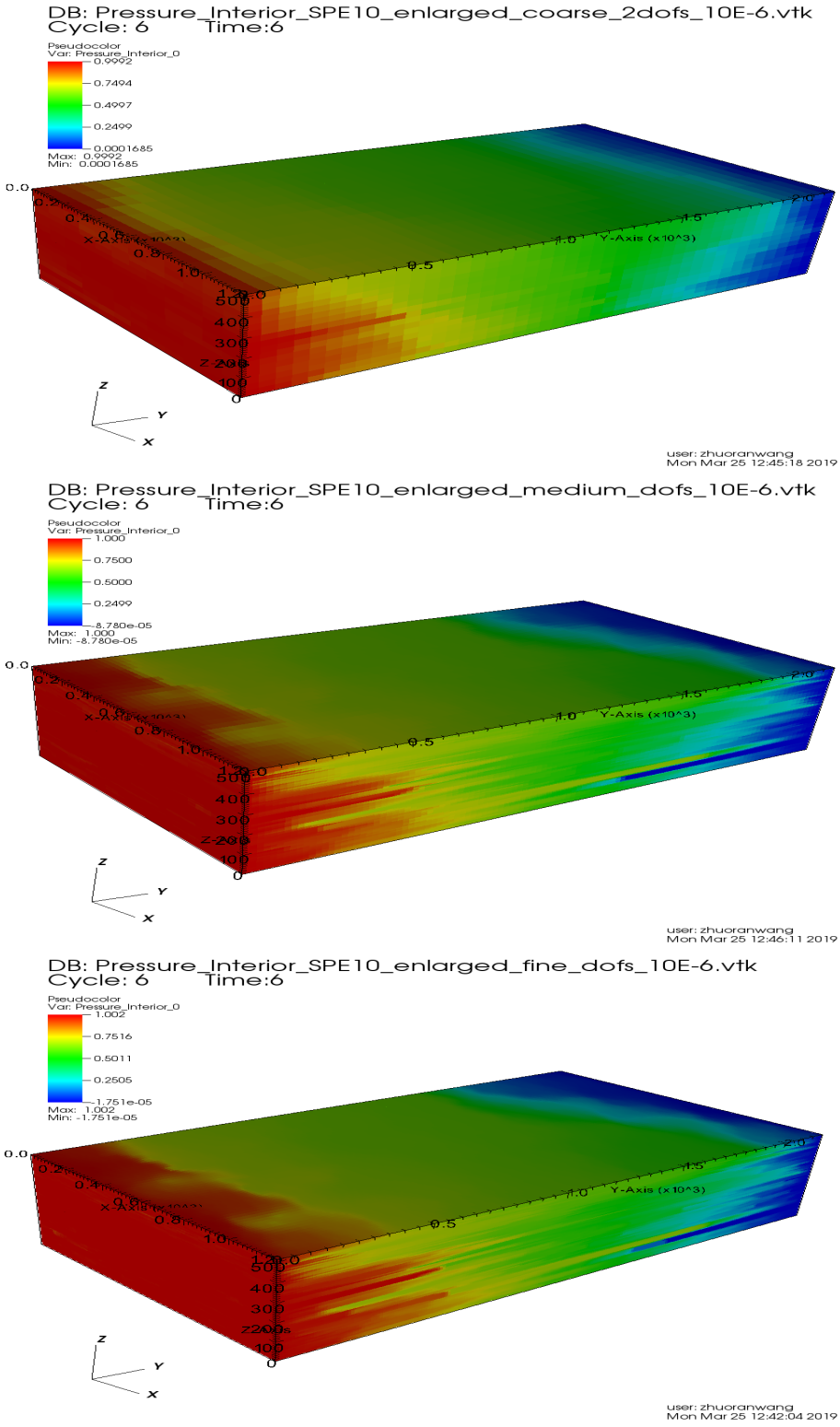


Fig. 4. Example 3 (SPE10): Numerical pressure for coarse, medium, and fine meshes.

expensive computation. Tables 3 and 4 show the running time of MEFM is significantly longer than that of WG.

Table 3. Example 3: SPE10_Darcy by WG($Q_0, Q_0; RT_{[0]}$) on 3 meshes

tol=10^{-6}					
	MaxItrs	#Itrs	$p_{\min}$	$p$max	Runtime
coarse	2*DOFs	40,383	1.684E-4	0.999,151	2m45s
medium	DOFs	207,704	-8.779E-5	1.000,003	1h34m
fine	DOFs	241,492	-1.750E-5	1.002,136	6h42m
tol=10^{-9}					
	MaxItrs	#Itrs	$p_{\min}$	$p$max	Runtime
coarse	6*DOFs	213,316	1.682E-4	0.999,151	5m24s
medium	DOFs	901,327	-8.782E-5	1.000,002	3h20m
fine	DOFs	1,371,887	-1.750E-5	1.000,083	17h38m

Run on Wolfgang Bangerth's UpDown Server at ColoState

Table 4. Example 3: SPE10_Darcy by MFEM($RT_{[0]}, Q_0$) on 3 meshes

tolerance	$p_{\min}$	$p$max	Runtime
coarse mesh			
$10^{-3}\|\|rhs\|\|$	1.693E-4	0.998,574	8m39s
$10^{-6}\|\|rhs\|\|$	1.682E-4	0.999,150	34m02s
$10^{-9}\|\|rhs\|\|$	1.682E-4	0.999,151	1h16m
medium mesh			
$10^{-3}\|\|rhs\|\|$	-8.712E-5	1.001,850	6h04m
$10^{-6}\|\|rhs\|\|$	-8.782E-5	1.000,002	31h25m
$10^{-9}\|\|rhs\|\|$	-8.782E-5	1.000,002	82h58m
fine mesh			
$10^{-3}\|\|rhs\|\|$	-1.745E-5	1.028,902	71h16m
$10^{-6}\|\|rhs\|\|$		DidNotTry	
$10^{-9}\|\|rhs\|\|$		DidNotTry	

Run on Wolfgang Bangerth's UpDown Server at ColoState

6 Concluding Remarks

The novel weak Galerkin finite element methods represent a different type of methodology for solving many real world problems modeled by partial differential equations. There have been efforts on implementing WG FEMs in `Matlab` and `C++`. But the work presented in this paper represents the first attempt for

implementing WG FEMs in a popular finite element package like deal.II. This shall provide open access to the scientific community for examining usefulness of the WG methodology for large-scale scientific computing tasks.

There are many research directions we could go from here.

(i) Parallelization of the WG solver for Darcy flow;
(ii) deal.II implementation for coupled WG Darcy solvers and transport solvers for the full problem of SPE10 or alike;
(iii) deal.II implementation for both 2-dim and 3-dim for the 2-field poroelasticity solver presented in [10];
(iv) Implementation of WGFEMs for triangular and tetrahedral meshes on FEniCS;
(v) add comments on HDG??

Results on such efforts will be reported in our future work.

References

1. https://github.com/dealii/dealii/pull/7746
2. Arbogast, T., Correa, M.: Two families of mixed finite elements on quadrilaterals of minimal dimension. SIAM J. Numer. Anal. 54, 3332–3356 (2016)
3. Bangerth, W., Hartmann, R., Kanschat, G.: deal.II — a general purpose object oriented finite element library. ACM Trans. Math. Softw. 33, 24.1–24.27 (2007)
4. Bastian, P., Riviere, B.: Superconvergence and $h(div)$ projection for discontinuous galerkin methods. Int. J. Numer. Meth. Fluids 42, 1043–1057 (2003)
5. Brenner, S., Scott, L.: The mathematical theory of finite element methods, Texts in Applied Mathematics, vol. 15. Springer-Verlag, New York, third edn. (2008)
6. Bush, L., Ginting, V.: On the application of the continuous Galerkin finite element method for conservation problems. SIAM J. Sci. Comput. 35, A2953–A2975 (2013)
7. Cockburn, B., Gopalakrishnan, J., Wang, H.: Locally conservative fluxes for the continuous Galerkin method. SIAM J. Numer. Anal. 45, 1742–1770 (2007)
8. Durlofsky, L.: Accuracy of mixed and control volume finite element approximations to Darcy velocity and related quantities. Water Resour. Res. 30, 965–973 (1994)
9. F.Brezzi, M.Fortin: Mixed and hybrid finite element methods. Springer-Verlag (1991)
10. Harper, G., Liu, J., Tavener, S., Wang, Z.: A two-field finite element solver for poroelasticity on quadrilateral meshes. Lec. Notes in Comput. Sci. 10862, 76–88 (2018)
11. Harper, G., Liu, J., Zheng, B.: The THex algorithm and a simple Darcy solver on hexahedral meshes. Proc. Comput. Sci. 108C, 1903–1912 (2017)
12. Lie, K.A.: An introduction to reservoir simulation using MATLAB/GNU Octave:. No. ISBN 9781108492430, Cambridge University Press (2019)
13. Lin, G., Liu, J., Mu, L., Ye, X.: Weak galerkin finite element methdos for Darcy flow: Anistropy and heterogeneity. J. Comput. Phys. 276, 422–437 (2014)
14. Lin, G., Liu, J., Sadre-Marandi, F.: A comparative study on the weak Galerkin, discontinuous Galerkin, and mixed finite element methods. J. Comput. Appl. Math. 273, 346–362 (2015)
15. Liu, J., Sadre-Marandi, F., Wang, Z.: Darcylite: A Matlab toolbox for Darcy flow computation. Proc. Comput. Sci. 80, 1301–1312 (2016)

16. Liu, J., Tavener, S., Wang, Z.: The lowest-order weak Galerkin finite element method for the Darcy equation on quadrilateral and hybrid meshes. J. Comput. Phys. 359, 312–330 (2018)
17. Mu, L., Wang, J., Ye, X.: A weak Galerkin finite element method with polynomial reduction. J. Comput. Appl. Math. 285, 45–58 (2015)
18. Sun, S., Liu, J.: A locally conservative finite element method based on piecewise constant enrichment of the continuous Galerkin method. SIAM J. Sci. Comput. 31, 2528–2548 (2009)
19. Wang, J., Ye, X.: A weak Galerkin finite element method for second order elliptic problems. J. Comput. Appl. Math. 241, 103–115 (2013)