Ruprecht-Karls-Universität

Heidelberg

Heidelberg Graduate School of

Mathematical and Computational Methods for the

Sciences

Master thesis

in the program of : "Scientific Computing"

submitted by: Saurabh Mehta

March 12, 2018

Matrix Free approach for Raviart-Thomas anisotropic Tensor Product Finite Elements

This Master thesis has been carried out by Saurabh Mehta

at the

Ruprecht-Karls-Universität Heidelberg

under the supervision of

Prof. Dr. Guido Kanschat

and

Dr. Daniel Arndt

Contents

Chapter 1
Purpose and Significance

1.1 Introduction

In this thesis, we develop techniques for fast, efficient cell based operator evaluation of Raviart-Thomas finite elements and thereby extend the existing implementation of matrix free framework in deal.ii ([10], [4]). The extensions allow the framework to also support anisotropic vector-valued finite elements in addition to existing isotropic Lagrangian finite elements.

Tests on finite element operators in divergence conforming spaces for Mixed diffusion equation and Stokes equations show that the method is sufficiently accurate (relative error 10e-16) and for RT_2 is already twice as fast as corresponding sparse matrix based solution.

1.2 Motivation

A typical finite element solver can be divided in at least 3 steps ([6], chapter 4.3):

1. Preprocessing (mesh generation, boundary conditions, initial data)
2. Processing (Picard/Newton iteration, matrix assembly, solving the linear system,etc)
3. Postprocessing (graphical output, computing values of interest (drag, lift, energy, etc)

In particular, in a typical finite element program matrix generation and solving the linear system are two separate steps. Traditional approaches in simulation using finite element method try to achieve computational efficiency with sparse system matrix and sophisticated solver techniques.

Looking at current trends, the modern processor architectures are hit by three walls ([13], chapter 1.3):

1. Power wall: Unacceptable growth in power usage with clock rate
2. Instruction-level parallelism (ILP) wall: Limits to available low-level parallelism
3. Memory wall: A growing discrepancy of processor speeds relative to memory speeds

Therefore, new hardware favor large parallel computation with reduced memory footprint. For computationally intensive applications like finite element methods, the implementations should focus on explicit parallelism and improved memory efficiency.

It has been shown that the Sparse Matrix-vector multiplication are fundamentally limited by memory bandwidth rather than compute bandwidth ([16]). However, if we consider iterative solvers, we realize that we actually don't need the whole matrix but only a matrix-vector product. This operation can in fact be done on-the-fly. By an optimal balance between memory access and at the cost of extra computations, once the problem size is sufficiently big then generating the matrix-vector on-the-fly from the weak form wins over the classical approach ([12]).

Further, the finite elements we deal with often have tensor product structure at least for quadrilaterals on the reference element. For example, the degrees of freedom for a Lagrangian nodal finite element or Raviart-Thomas element are the tensor product of the 1-D degrees of freedom in each direction. By using quadrature rules which are themselves tensor product of 1-D rules, then the evaluation of a finite element function on quadrature points can benefit from sum-factorization (3.4) in terms of computational complexity.

The ideas of on-the-fly matrix-vector product and utilizing tensor product structure have been implemented in deal.ii as matrix free framework ([10]) and presented in dealii step 37

The current implementation of matrix free framework supports Lagrangian finite elements which are isotropic in nature. This means that we are limited to having the same ansatz space in each coordinate direction.

On the other hand, the divergence-conforming Raviart-Thomas elements use a polynomial space of anisotropic form (2.5) $Q_{k+1,k,k} \times Q_{k,k+1,k} \times Q_{k,k,k+1}$ and can not be evaluated in the current framework of matrix free. Hence, we are currently limited for using the high performance implementation. In this thesis, we develop techniques to lift this restriction.

1.3 Outline

In 2, we define finite elements and provide those concepts which we will use in our study.
in 3, we develop mathematical foundations for cell based operator evaluation (matrix free) approach. We list down those operators which we will consider in our study.
In 4, we provide overview of existing implementation of matrix free framework and talk about the extensions we did in accordance with 3.
in 5, we analyze results, followed by conclusion in 6.

Throughput this thesis, the term **"matrix free"** is used to refer to cell based finite element operator approach which is also the framework implemented under the same name in deal.ii. **"MatrixFree"** instead refers to a `C++` class for data cache used in this framework.

Chapter 2
Mathematical Preliminaries

The following definitions mainly follow [2] and [6]

2.1 Finite Elements

Definition 1.1.1 A finite Element consists of a triplet $\{K, \mathcal{P}, \Sigma\}$ where:

- K is a compact, connected, Lipschitz subset of $\mathbb{R}^d$ with non-empty interior
- $\mathcal{P}$ is vector space of functions $p : K \to \mathbb{R}^m$ for some positive integer m (typically m = 1 or d)
- Σ is a set of n_{sh} unisolvent linear functionals $\sigma_1, ..., \sigma_{n_{sh}}$ acting on the elements of $\mathcal{P}$, and such that the following linear mapping is bijective:

$$\mathcal{P} \ni p \mapsto (\sigma_1(p), ...\sigma_{n_{sh}}(p)) \in \mathbb{R}^{n_{sh}} \tag{2.1}$$

Remark 1.1.2

- The vector space $\mathcal{P}$ is a space of ansatz functions on K
- The linear forms in Σ are also referred to as **node functionals**
- As a consequence of bijectivity of mapping (2.1) , there exists a basis $\{\mathbf{\Phi}_1, ..., \mathbf{\Phi}_{n_{sh}}\}$ satisfying:

$$\sigma_i(\mathbf{\Phi}_j) = \delta_{ij} \tag{2.2}$$

Such a basis is commonly known as local shape functions or just **shape functions**

2.2 Domain and mesh

Following [7] we define domain and mesh as follows.

Definition 1.2.1 (**Domain**)

- In dimension 1, a domain is an open, bounded interval
- In dimension ≥ 2, a domain is an open, bounded, connected set in $\mathbb{R}^d$ with Lipschitz continuous boundary $\partial\Omega$.

This implies that for every point $x \in \partial\Omega$, there is a neighborhood $\mathcal{U}_x$ with a Lipschitz-continuous mapping $\Pi_x : \mathcal{U}_x \to \mathbb{R}^d$ such that

$$\Pi_x(\mathcal{U}_x \subset \partial\Omega) \subset \{x \in \mathbb{R}^d | x_0 = 0\} \quad \text{and}$$
$$\Pi_x(\mathcal{U}_x \subset \Omega) \subset \{x \in \mathbb{R}^d | x > 0\}$$

Definition 1.2.1 (**Mesh**) A mesh is defined as a subdivision of a domain Ω into a finite number N_{el} of non-overlapping cells which are triangles or quadrilaterals in two dimensions and tetrahedron or hexahedron in three dimensions.

- A mesh $\{K_m\}_{1 \leq m \leq N_{el}}$ is denoted by $\mathcal{T}_h$. The subscript h refers to level of refinement of the mesh
- The **diameter** of a cell Γ is denoted by h_Γ, and the mesh **size** is $h = \max(h_\Gamma)$

Remark 1.2.2 A mesh is:

- **conforming** if a face/an edge of a cell is either on the boundary or a face/an edge of another cell
- **Uniform** if all cells are congruent
- Consider family of meshes $\mathcal{T}_h$ for $h \to 0$. Such a family is **quasi-uniform**, if there is a constant c such that

$$\forall h, \quad \forall T \in \mathcal{T}_h, \quad h_T \geq c.h \quad , c > 0$$

In practice, a mesh is generated from a *reference cell*, $\hat{K}$ and a set of geometric transformations mapping $\hat{K}$ to *real cell*, the actual mesh cells.

Definition 1.2.3 (**Reference cell**) The triplet $\{\hat{K}, \hat{\mathcal{P}}, \hat{\Sigma}\}$ is called the reference Finite Element. In particular, the basis functions $\{\hat{\mathbf{\Phi}}_1, ..., \hat{\mathbf{\Phi}}_{n_{sh}}\}$ are called **reference shape functions**

- For simplicity, we assume that all mesh cells are generated using the same geometric reference finite element

Definition 1.2.4 (**Affine meshes**) For $K \in \mathcal{T}_h$, denote by $T_k : \hat{K} \to K$ the corresponding transformation. If the transformations $\{T_m\}_{1 \leq m \leq N_{el}}$ are affine, the mesh is said to be affine.

In this thesis, we only consider:

- domains with polygonal boundary
- cartesian mesh cells which are quadrilaterals or hexahedron in two and three dimensions respectively
- quasi-uniform families of conforming mesh, with affine transformation (bilinear mapping)

Reasons for this choice will be made explicit in 3.2.4

2.3 Polynomial spaces

Definition 1.3.1 (**Polynomial space**) Consider a domain $K \in \mathbb{R}^d$ We denote by $\mathbb{P}_k(K)$ the set of all polynomials on K of degree at most k

Definition 1.3.2 (**Tensor product polynomial space**) The space of (isotropic) tensor product polynomials $\mathbb{Q}_k(K)$ is the product of polynomials of degree at most equal to k in each single variable

$$\begin{aligned} \mathbb{Q}_k(K) &= \mathbb{P}_k(\mathbb{R}^1) \otimes ... \otimes \mathbb{P}_k(\mathbb{R}^1) \\ q(x_1, ..., x_d) &= \prod_{i=1}^{d} p_i(x_i) \quad p_i \in \mathbb{P}_k(\mathbb{R}^1) \end{aligned} \tag{2.3}$$

Definition 1.3.2 (**Anisotropic Tensor product polynomials**) Similarly, the space of anisotropic tensor product polynomials is defined as

$$\begin{aligned} \mathbb{Q}_{k_1,..k_d}(K) &= \mathbb{P}_{k_1}(\mathbb{R}^1) \otimes ... \otimes \mathbb{P}_{k_d}(\mathbb{R}^1) \\ q(x_1, ..., x_d) &= \prod_{i=1}^{d} p_i(x_i) \quad p_i \in \mathbb{P}_{k_i}(\mathbb{R}^1) \end{aligned} \tag{2.4}$$

2.4 Parametric Finite Elements

In practice, the finite elements are defined on a reference cell and mapped to the real cell by a suitable transformation. We assume an isoparametric mapping.

Let us assume that for all $K \in \mathcal{T}_h$ there exists

- a Banach space $V(K)$ of functions $v : K \to \mathbb{R}^m$, such that $P \subset V(K)$

- a suitable mapping

$$\Pi_K : V(K) \to V(\hat{K})$$

then, a set of finite elements in $\mathcal{T}_h$ can be defined as as follows:

Definition 1.4.1 For $K \in \mathcal{T}_h$, the triplet $\{K, \mathcal{P}, \Sigma\}$ defined by:

$$\begin{aligned} K &= T_k(\hat{K}) \\ P_K &= \{\Pi_K^{-1}(\hat{p}); \hat{p} \in \hat{P}\} \\ \Sigma_K &= \{\sigma_{K,i}(p) = \hat{\sigma}_i(\Pi_K(p)), \quad \forall p \in P_K\} \end{aligned}$$

is a finite element. The local shape functions are $\mathbf{\Phi}_{K,i} = \Pi_K^{-1}(\hat{\mathbf{\Phi}}_i)$

Choice of mapping Denote by $J(\hat{x})$ and $|J(\hat{x})|$, the jacobi matrix of transformation from reference to real cell, and the jacobian determinant respectively. $\mathbf{x} \in K$ and $\hat{\mathbf{x}} \in \hat{K}$ such that $\mathbf{x} = \Pi(\hat{\mathbf{x}})$

- For Lagrange nodal Finite Elements, linear bijective mapping is used

$$\mathbf{\Phi}(\mathbf{x}) = \hat{\mathbf{\Phi}}(\hat{\mathbf{x}}) \qquad \nabla\mathbf{\Phi}(\mathbf{x}) = \nabla\hat{\mathbf{\Phi}}(\hat{\mathbf{x}}) J(\hat{\mathbf{x}})^{-T} \tag{2.5}$$

- For Raviart-Thomas Finite Elements, **Piola transform** or contravariant transformation is used

$$\mathbf{\Phi}(\mathbf{x}) = \frac{1}{|J(\hat{\mathbf{x}})|} J(\hat{\mathbf{x}}) \hat{\mathbf{\Phi}}(\hat{\mathbf{x}}) \qquad \nabla\mathbf{\Phi}(\mathbf{x}) = \frac{1}{|J(\hat{\mathbf{x}})|} J(\hat{\mathbf{x}}) [\hat{\nabla}\hat{\mathbf{\Phi}}(\hat{\mathbf{x}})] J^{-1}(\hat{\mathbf{x}}) \tag{2.6}$$

2.5 Raviart-Thomas Finite Element

Definition 1.5.1 Define the function spaces

$$\begin{aligned} H^{div}(\Omega) &= \{\mathbf{v} \in L^2(\Omega; \mathbb{R}^d) | \nabla \cdot \mathbf{v} \in L^2\} \\ H^{div}_{\Gamma_N}(\Omega) &= \{\mathbf{v} \in H^{div}(\Omega) | \mathbf{v} \cdot \mathbf{n} = \mathbf{v}^N \cdot \mathbf{n} \quad on \quad \Gamma_N \subset \partial\Omega\} \\ H^{div}_0(\Omega) &= \{\mathbf{v} \in H^{div}(\Omega) | \mathbf{v} \cdot \mathbf{n} = 0 \quad on \quad \partial\Omega\} \end{aligned} \tag{2.7}$$

Definition 1.5.2 The **Raviart-Thomas element** of degree $l \geq 0$ on the reference cell $\hat{K} = [-1, 1]^d$ consists of the polynomial space

$$RT_l(\hat{K}) = \mathbb{Q}_l^d(\hat{K}) + \mathbf{x}\mathbb{Q}_l(\hat{K}) \tag{2.8}$$

Chapter 3
Mixed Finite Element Operators and Cell-based evaluation scheme (Matrix Free)

3.1 Saddle point problems

3.1.1 Operator notation

Definition 2.1.1 Following [8], [5], the **abstract saddle point problem** in weak form reads:
Find a pair $(\mathbf{u}, p) \in V \times Q$ such that

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) &= f(\mathbf{v}) \quad \forall \mathbf{v} \in V, \\ b(\mathbf{u}, q) - c(p, q) &= g(q) \quad \forall q \in Q \end{aligned} \tag{3.1}$$

Definition 2.1.2 Operator Notation : The bilinear forms in Definition 2.1.1 can also be written in operator notation as:

$$\begin{aligned} \mathcal{A}\mathbf{u} + \mathcal{B}^T p &= f \quad \text{in } V^*, \\ \mathcal{B}\mathbf{u} - \mathcal{C}p &= g \quad \text{in } Q^* \end{aligned} \tag{3.2}$$

where the corresponding operators map to dual space:

$$\begin{aligned} \mathcal{A} &: V \to V^* \\ \mathcal{B} &: V \to Q^* \\ \mathcal{B}^T &: Q \to V^* \\ \mathcal{C} &: Q \to Q^* \end{aligned}$$

3.1.2 Reference problems

Since we focus exclusively on Raviart Thomas polynomial spaces, (2.8), we select two reference mixed finite element problems to develop the further analysis.

Definition 2.2.1 The **mixed diffusion problem** : Several different transport phenomena with conservation properties can be modeled as a diffusion problem. Notable examples are Darcy's porous media flow, heat conduction etc.
Mathematically, the weak form is set up in dual mixed formulation ([8]) which reads as:
Find $(\mathbf{u},p) \in V \times Q$ such that $\forall \mathbf{v} \in V$ and $\forall q \in Q$ holds such that

$$\begin{aligned}(K^{-1}\mathbf{u},\mathbf{v}) - (p,\nabla\cdot\mathbf{v}) &= \langle p^D, \mathbf{v}\cdot\mathbf{n}\rangle_{\Gamma_D} \\ (\nabla\cdot\mathbf{u},q) &= (f,q)\end{aligned} \tag{3.3}$$

- The spaces are

$$V = H^{div}_{\Gamma_N}(\Omega) \qquad Q = L^2(\Omega)$$

- Γ_D = Dirichlet boundary, Γ_N = Neumann boundary and $\Gamma_D \cap \Gamma_N = \partial\Omega$
- $p = p^D$ on Γ_D, $u = u^N$ on Γ_N
- K denotes the permeability tensor (Note: this is unrelated to mesh cell K)

Remark 2.2.2 For **discretization** on mesh $\mathcal{T}_h$, we choose conforming subspaces:

$$\begin{aligned}\mathbf{V}_h &= \{\mathbf{v} \in H^{div}_{\Gamma_N}(\Omega) | \forall K \in \mathcal{T}_h : \mathbf{v}|_K \in RT_l\} \\ Q_h &= \{q \in L^2(\Omega) | \forall K \in \mathcal{T}_h : q|_K \in Q_l\}\end{aligned} \tag{3.4}$$

Definition 2.2.3 The **simplified Stokes equations** : The weak form with Dirichlet boundary reads :
Find $(\mathbf{u},p) \in V \times Q$ such that $\forall \mathbf{v} \in V$ and $\forall q \in Q$ holds

$$\begin{aligned}\nu(\nabla\mathbf{u},\nabla\mathbf{v}) - (\nabla\cdot\mathbf{v},p) &= (f,\mathbf{v}) \quad \forall \mathbf{v} \in V \\ -(\nabla\cdot\mathbf{u},q) &= 0 \quad \forall q \in Q\end{aligned} \tag{3.5}$$

The spaces are:

$$V = H^1_0(\Omega,\mathbb{R}^d) \qquad Q = L^2_0(\Omega)$$

Remark 2.2.4 For **discretization** on mesh $\mathcal{T}_h$, we are again interested in $H(div)$ conforming spaces:

$$\begin{aligned}\mathbf{V}_h &= \{\mathbf{v} \in H_0^{div}(\Omega) | \forall K \in \mathcal{T}_h : \mathbf{v}|_K \in RT_l\} \\ Q_h &= \{q \in L_0^2(\Omega) | \forall K \in \mathcal{T}_h : q|_K \in Q_l\}\end{aligned} \tag{3.6}$$

However, this will only ensure continuity of the normal component of the functions across interfaces between cells. For the solution $\mathbf{u} \in H_0^1(\Omega, \mathbb{R}^d)$, we also need continuity of tangential components. This can be achieved by **Discontinuous Galerkin discretization**.
Without going into details, we refer to ([9], chapter 2.1) and highlight that such a discretization will lead to elliptic bilinear form as:

$$a_h(\mathbf{u}, \mathbf{v}) = (\nabla \mathbf{u}, \nabla \mathbf{v}) + \text{interior penalty terms}$$

In this thesis, we limit our work to evaluation of $(\nabla \mathbf{u}, \nabla \mathbf{v})$ and leave the rest for future work.
Further on, any reference to operator $\mathcal{A}$ in the context of Stokes equations will mean evaluation of bilinear form $(\nabla \mathbf{u}, \nabla \mathbf{v})$ only.

3.2 Cell-based Operator Evaluation

The general approach of cell-based operator evaluation follows from [10]
Consider for example, the mixed diffusion problem (3.3) for which we are interested in solution on discrete space (3.3) using finite element Galerkin approximation. As a standard approach, we will first assemble the full system matrix ($\mathcal{S}$). For iterative solvers, we will multiply the system matrix with intermediate vectors ($\mathbf{x} = [\mathbf{u}; p]$). Assuming that the degrees of freedom are ordered for velocity followed by pressure, the system matrix in our case will look like:

$$\mathcal{S} = \begin{bmatrix} \mathcal{A} & \mathcal{B}^T \\ \mathcal{B} & 0 \end{bmatrix}$$

$$\mathcal{S}\mathbf{x} = \mathbf{y}$$

With an appropriate choice of solution spaces, this can be written as a sum of cell-wise operations as:

$$\begin{aligned}\mathbf{y} = \mathcal{S}\mathbf{x} &= \sum_{k=1}^{N_{el}} P_k^T \mathcal{S}_k (P_k \mathbf{x}) \\ &= \sum_{k=1}^{N_{el}} P_k^T \mathcal{S}_k \mathbf{x}_k\end{aligned} \tag{3.7}$$

By using the structure of $\mathcal{S}_k$ and denoting degrees of freedom on the cell as $\mathbf{x}_k = [\mathbf{u}_k; p_k]$, the cell specific matrix-vector product can be written

$$\mathcal{S}_k \mathbf{x}_k = \begin{bmatrix} \mathcal{A}_k \mathbf{u}_k + \mathcal{B}_k^T p_k \\ \mathcal{B}_k \mathbf{u}_k \end{bmatrix} \tag{3.8}$$

Thus, it is clear that we can replace full system matrix operation with small cell based operators.

For the evaluation of small cell based operators, an efficient and general scheme has been described in [10] and implemented for tensor product Lagrangian Finite Elements under the name **matrix free framework** in the deal.ii software. Key idea is :

- Compute $\mathcal{A}_k \mathbf{u}_k$ (or other operator-vector products) by quadrature on cell K through evaluating the FE function $\mathbf{u}_k$ and/or its derivatives on all quadrature point and testing by all test functions related to the cell

We aim to extend this technique to support Raviart Thomas finite element which has an anisotropic tensor product structure.
We now study several operators which are important constituents of the problems which we want to tackle. In every subsection, we briefly discuss the operator evaluation as already used in matrix free framework and then develop ideas for Raviart-Thomas elements.

3.2.1 Notations

- $\mathbf{\Phi_i}$ denotes vector valued shape functions for velocity on real cell and $\hat{\mathbf{\Phi}}_i$ on reference cell. $i = 1, .., n_{sh}$
- Ψ_i denotes scalar valued shape functions for pressure on real cell and $\hat{\Psi}_i$ on reference cell. $i = 1, .., n_{qh}$
- `n_comp` = number of vector components. `dim` = space dimensions. In this report, `n_comp = dim`. For clarity, they will be referred as such
- n_Q = number of quadrature points on on real/reference cell, n_{sh} = number of shape functions on real/reference cell
- The letter c, whether in subscript or superscript denotes component number
- Whenever there is no confusion about component number:
  - $\mathbf{\Phi_i}^c$ is replaced with Φ_i
  - $\mathcal{A}_k^c$ is replaced with A. Similarly for operator B. These letters are exclusively reserved throughout the thesis

- N = matrix of shape function values evaluated on all quadrature points of real cell
- D = matrix of shape function derivatives evaluated on all quadrature points of real cell
- W = matrix of jacobians times quadrature weights
- For simplicity, $\mathbf{u}_k$ is replaced with $\mathbf{u}$

3.2.2 Operator $\mathcal{A}$ (Stokes)

The first operator which we study is cell-based operator $\mathcal{A}_k$ from (3.5) corresponding to weak form:

$$a(\mathbf{u},\mathbf{v}) = (\nabla\mathbf{u}, \nabla\mathbf{v})$$

If we consider

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^{n_{sh}} \mathbf{\Phi_i}(\mathbf{x}) u^i$$

then the operator $\mathcal{A}_k$ evaluates $(\nabla\mathbf{\Phi_j}, \nabla\mathbf{u}))_\Omega$, $j = 1, ..., n_{sh}$
In matrix form, this can be written as:

$$\mathcal{A}_k = \begin{bmatrix} (\nabla\mathbf{\Phi_1}, \nabla\mathbf{\Phi_1}) & (\nabla\mathbf{\Phi_1}, \nabla\mathbf{\Phi_2}) & ... & (\nabla\mathbf{\Phi_1}, \nabla\mathbf{\Phi_{n_{sh}}}) \\ (\nabla\mathbf{\Phi_2}, \nabla\mathbf{\Phi_1}) & (\nabla\mathbf{\Phi_2}, \nabla\mathbf{\Phi_2}) & ... & (\nabla\mathbf{\Phi_2}, \nabla\mathbf{\Phi_{n_{sh}}}) \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ (\nabla\mathbf{\Phi_{n_{sh}}}, \nabla\mathbf{\Phi_1}) & (\nabla\mathbf{\Phi_{n_{sh}}}, \nabla\mathbf{\Phi_2}) & ... & (\nabla\mathbf{\Phi_{n_{sh}}}, \nabla\mathbf{\Phi_{n_{sh}}}) \end{bmatrix}$$

An element in row i and column j of this matrix is given by:

$$\begin{aligned} \mathcal{A}_k(i,j) &= (\nabla\mathbf{\Phi_i}, \nabla\mathbf{\Phi_j}) \\ &= \int \nabla\mathbf{\Phi_i} : \nabla\mathbf{\Phi_j} d\mathbf{x} \end{aligned}$$

where : denotes the Frobenius product of the two tensors.
Then, by the definition of Frobenius product, we can write:

$$\mathcal{A}_k(i,j) = \sum_{c=1}^{n\_comp} (\nabla\mathbf{\Phi_i}^c . \nabla\mathbf{\Phi_j}^c)$$

where . denotes the vector inner product.
Consequently, we can write:

$$\mathcal{A}_k = \sum_{c=1}^{n\_comp} \mathcal{A}_k^c \tag{3.9}$$

i.e., the cell wise operator evaluation for vector valued Finite elements can be factorized into evaluation of individual vector components.
This SOP (sum of products) form has important implications for implementation as we will see later.

Let us turn our attention to a single component $\mathcal{A}_k^c$. For simplicity of notation, we remove the superscript and k to indicate operator evaluation on a component as follows:

$$\mathcal{A}_k^c = A = \begin{bmatrix} (\nabla\Phi_1, \nabla\Phi_1) & (\nabla\Phi_1, \nabla\Phi_2) & ... & (\nabla\Phi_1, \nabla\Phi_{n_{sh}}) \\ (\nabla\Phi_2, \nabla\Phi_1) & (\nabla\Phi_2, \nabla\Phi_2) & ... & (\nabla\Phi_2, \nabla\Phi_{n_{sh}}) \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ (\nabla\Phi_{n_{sh}}, \nabla\Phi_1) & (\nabla\Phi_{n_{sh}}, \nabla\Phi_2) & ... & (\nabla\Phi_{n_{sh}}, \nabla\Phi_{n_{sh}}) \end{bmatrix}$$

It has been explained in detail in [10] and dealii step 37 that the matrix form A can be factorized as:

$$A = D^T W D \tag{3.10}$$

where $D =$ derivatives evaluated on all quadrature points on real cell, is a matrix of dimensions $(n_Q \times dim) \times n_{sh}$ as:

$$D = \begin{bmatrix} \nabla\Phi_1(\mathbf{x_1}) & \nabla\Phi_2(\mathbf{x_1}) & ... & \nabla\Phi_{n_{sh}}(\mathbf{x_1}) \\ \nabla\Phi_1(\mathbf{x_2}) & \nabla\Phi_2(\mathbf{x_2}) & ... & \nabla\Phi_{n_{sh}}(\mathbf{x_2}) \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ \nabla\Phi_1(\mathbf{x_Q}) & \nabla\Phi_2(\mathbf{x_Q}) & ... & \nabla\Phi_{n_{sh}}(\mathbf{x_Q}) \end{bmatrix} \tag{3.11}$$

and $W =$ is a diagonal matrix of dimensions $(n_Q \times dim) \times (n_Q \times dim)$ as:

$$W = \begin{bmatrix} |J(\hat{\mathbf{x_1}})|w_1 & & & & & & & \\ & |J(\hat{\mathbf{x_1}})|w_1 & & & & & & \\ & & .. & & & & & \\ & & & |J(\hat{\mathbf{x_2}})|w_2 & & & & \\ & & & & .. & & & \\ & & & & & .. & & \\ & & & & & & |J(\hat{\mathbf{x_Q}})|w_Q & \\ & & & & & & & |J(\hat{\mathbf{x_Q}})|w_Q \end{bmatrix} \tag{3.12}$$

For operator evaluation, transformation from real cell to reference cell is needed.

3.2.3 Mapping : Lagrangian Finite elements

We use linear mapping as defined earlier (2.5).
On closer examination, we see that this form allows us to obtain component wise separation of gradients. Using our previous notation, this is:

$$\begin{aligned}\nabla \Phi(\mathbf{x}) &= \nabla \mathbf{\Phi^c}(\mathbf{x}) \\ &= J(\hat{\mathbf{x}})^{-1} \nabla \hat{\Phi}(\hat{\mathbf{x}})\end{aligned}$$

Note that Φ and $\hat{\Phi}$ are treated as column vectors. As already explained in dealii step 37, such a component wise separation allows us to obtain:

$$\begin{aligned}& D = \mathcal{J}^{-T} D_{ref} \\ \implies & A = \mathcal{A}_k^c \\ & = (D_{ref}^T \mathcal{J}^{-1}) W (\mathcal{J}^{-T} D_{ref})\end{aligned} \tag{3.13}$$

where D_{ref} is similar in structure to D but gradients are evaluated on reference cell instead of real cell.
$\mathcal{J}$ is a block diagonal matrix of dimensions $(n_Q \times dim) \times (n_Q \times dim)$ as:

$$\mathcal{J}^{-1} = \begin{bmatrix} J^{-1}(\hat{\mathbf{x_1}}) & & & & \\ & J^{-1}(\hat{\mathbf{x_2}}) & & & \\ & & .. & & \\ & & & .. & \\ & & & & J^{-1}(\hat{\mathbf{x_Q}}) \end{bmatrix}$$

3.2.4 Mapping : Raviart Thomas elements

We use Piola transformation mapping as defined earlier (2.6).
However, now we see that there is a coupling between different components of reference cell gradient evaluation. In general, we do not get a well separated form as we would desire (3.9) for the matrix free framework. *It is for this reason that we choose cartesian mesh cells in our current work.* With such a condition, the jacobian transformation is diagonal matrix and we are again able to obtain a component wise separation.

To see this, let us consider a case where dim=3. Then $J(\hat{\mathbf{x}})$ is a 3×3 matrix.
Denote the diagonal coefficient as $j_i = j_i(\hat{\mathbf{x}}) \frac{1}{|J(\hat{\mathbf{x}})|}, \quad i = 1,2,3$. Then we can see:

$$\nabla \mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} j_1 \partial_{\hat{x}} \hat{\mathbf{\Phi}}^{\mathbf{1}}(\hat{\mathbf{x}}) & \frac{j_1}{j_2} \partial_{\hat{y}} \hat{\mathbf{\Phi}}^{\mathbf{1}}(\hat{\mathbf{x}}) & \frac{j_1}{j_3} \partial_{\hat{z}} \hat{\mathbf{\Phi}}^{\mathbf{1}}(\hat{\mathbf{x}}) \\ \frac{j_2}{j_1} \partial_{\hat{x}} \hat{\mathbf{\Phi}}^{\mathbf{2}}(\hat{\mathbf{x}}) & j_2 \partial_{\hat{y}} \hat{\mathbf{\Phi}}^{\mathbf{2}}(\hat{\mathbf{x}}) & \frac{j_2}{j_3} \partial_{\hat{z}} \hat{\mathbf{\Phi}}^{\mathbf{2}}(\hat{\mathbf{x}}) \\ \frac{j_3}{j_1} \partial_{\hat{x}} \hat{\mathbf{\Phi}}^{\mathbf{3}}(\hat{\mathbf{x}}) & \frac{j_3}{j_2} \partial_{\hat{y}} \hat{\mathbf{\Phi}}^{\mathbf{3}}(\hat{\mathbf{x}}) & j_3 \partial_{\hat{z}} \hat{\mathbf{\Phi}}^{\mathbf{3}}(\hat{\mathbf{x}}) \end{bmatrix}$$

Using such a form, we obtain the transformation matrix from real to reference cell as:

$$D = D_{ref} \odot H$$

where $\odot$ denotes Hadamard product (point-wise). H has a repetitive structure and its elements depends on the component and quadrature point:

$$H(c1) = \begin{bmatrix} j_1 & j_1 & j_1 & \dots \\ \underline{j_1} & \underline{j_1} & \underline{j_1} & \dots \\ j_2 & j_2 & j_2 & \\ \underline{j_1} & \underline{j_1} & \underline{j_1} & \dots \\ j_3 & j_3 & j_3 & \\ j_1 & j_1 & j_1 & \dots \\ \underline{j_1} & \underline{j_1} & \underline{j_1} & \dots \\ j_2 & j_2 & j_2 & \\ \underline{j_1} & \underline{j_1} & \underline{j_1} & \dots \\ j_3 & j_3 & j_3 & \\ .. & .. & .. & .. \\ .. & .. & .. & .. \end{bmatrix} H(c2) = \begin{bmatrix} \underline{j_2} & \underline{j_2} & \underline{j_2} & \dots \\ j_1 & j_1 & j_1 & \\ j_2 & j_2 & j_2 & \dots \\ \underline{j_2} & \underline{j_2} & \underline{j_2} & \dots \\ j_3 & j_3 & j_3 & \\ \underline{j_2} & \underline{j_2} & \underline{j_2} & \dots \\ j_1 & j_1 & j_1 & \\ j_2 & j_2 & j_2 & \dots \\ \underline{j_2} & \underline{j_2} & \underline{j_2} & \dots \\ j_3 & j_3 & j_3 & \\ .. & .. & .. & .. \\ .. & .. & .. & .. \end{bmatrix} H(c3) = \begin{bmatrix} \underline{j_3} & \underline{j_3} & \underline{j_3} & \dots \\ j_1 & j_1 & j_1 & \\ \underline{j_3} & \underline{j_3} & \underline{j_3} & \dots \\ j_2 & j_2 & j_2 & \\ j_3 & j_3 & j_3 & \dots \\ \underline{j_3} & \underline{j_3} & \underline{j_3} & \dots \\ j_1 & j_1 & j_1 & \\ \underline{j_3} & \underline{j_3} & \underline{j_3} & \dots \\ j_2 & j_2 & j_2 & \\ j_3 & j_3 & j_3 & \dots \\ .. & .. & .. & .. \\ .. & .. & .. & .. \end{bmatrix}$$

and finally we get

$$\begin{aligned} A &= \mathcal{A}_k^c \\ &= (D_{ref} \odot H)^T W (D_{ref} \odot H) \\ &= D_{ref}^T \widetilde{W} (D_{ref} \odot H) \end{aligned} \tag{3.14}$$

where $\widetilde{W} = (H^T \odot W)$ which is possible because $\widetilde{W}$ is diagonal.

3.2.5 Operator $\mathcal{A}$ (Mixed diffusion)

The first operator which we study is cell-based operator $\mathcal{A}_k$ from (3.6) corresponding to weak form:

$$a(\mathbf{u}, \mathbf{v}) = (K^{-1}\mathbf{u}, \mathbf{v})$$

The operator $\mathcal{A}_k$ now evaluates $(\mathbf{\Phi_j}, K^{-1}\mathbf{u})_\Omega$, $j = 1, ..., n_{sh}$

As earlier, the cell operator can be written in matrix form:

$$\mathcal{A}_k = \begin{bmatrix} (\mathbf{\Phi_1}, K^{-1}\mathbf{\Phi_1}) & (\mathbf{\Phi_1}, K^{-1}\mathbf{\Phi_2}) & ... & (\mathbf{\Phi_1}, K^{-1}\mathbf{\Phi_{n_{sh}}}) \\ (\mathbf{\Phi_2}, K^{-1}\mathbf{\Phi_1}) & (\mathbf{\Phi_2}, K^{-1}\mathbf{\Phi_2}) & ... & (\mathbf{\Phi_2}, K^{-1}\mathbf{\Phi_{n_{sh}}}) \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ (\mathbf{\Phi_{n_{sh}}}, K^{-1}\mathbf{\Phi_1}) & (\mathbf{\Phi_{n_{sh}}}, K^{-1}\mathbf{\Phi_2}) & ... & (\mathbf{\Phi_{n_{sh}}}, K^{-1}\mathbf{\Phi_{n_{sh}}}) \end{bmatrix}$$

An element in row i and column j of this matrix is given by:

$$\begin{aligned}\mathcal{A}_k(i,j) &= (\mathbf{\Phi_i}, K^{-1}\mathbf{\Phi_j}) \\ &= \int \mathbf{\Phi_i} : K^{-1}\mathbf{\Phi_j} d\mathbf{x}\end{aligned}$$

where : denotes the Frobenius product of the two tensors.
For a general permeability tensor K where the elements are unique, such a Frobenius product does not allow us to obtain the desired SOP form (3.9) as we would like. However for the special case where K is unit matrix (or proportional to unit matrix), we can again obtain an SOP form.
And so we continue our analysis by assuming that K is unit matrix.
With this, we immediately obtain:

$$\mathcal{A}_k = \sum_{c=1}^{n\_comp} \mathcal{A}_k^c$$

where

$$\mathcal{A}_k^c = A = \begin{bmatrix} (\Phi_1,\Phi_1) & (\Phi_1,\Phi_2) & .. & (\Phi_1,\Phi_{n_{sh}}) \\ (\Phi_2,\Phi_1) & (\Phi_2,\Phi_2) & .. & (\Phi_2,\Phi_{n_{sh}}) \\ ... & ... & ... & ... \\ (\Phi_{n_{sh}},\Phi_1) & (\Phi_{n_{sh}},\Phi_2) & .. & (\Phi_{n_{sh}},\Phi_{n_{sh}}) \end{bmatrix}$$

Such a form can be easily factorized as:

$$A = N^T W N \tag{3.15}$$

where $N =$ shape functions evaluated on all quadrature points on real cell, is a matrix of dimensions $n_Q \times n_{sh}$. as:

$$N = \begin{bmatrix} \Phi_1(\mathbf{x_1}) & \Phi_2(\mathbf{x_1}) & ... & \Phi_{n_{sh}}(\mathbf{x_1}) \\ \Phi_1(\mathbf{x_2}) & \Phi_2(\mathbf{x_2}) & ... & \Phi_{n_{sh}}(\mathbf{x_2}) \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ \Phi_1(\mathbf{x_Q}) & \Phi_2(\mathbf{x_Q}) & ... & \Phi_{n_{sh}}(\mathbf{x_Q}) \end{bmatrix}$$

while $W =$ is a diagonal matrix of dimensions $n_Q \times n_Q$ as:

$$W = \begin{bmatrix} |J(\hat{\mathbf{x}}_1)|w_1 & & & & \\ & |J(\hat{\mathbf{x}}_2)|w_2 & & & \\ & & .. & & \\ & & & .. & \\ & & & & |J(\hat{\mathbf{x}}_Q)|w_Q \end{bmatrix}$$

For operator evaluation, transformation from real cell to reference cell is needed.

3.2.6 Mapping : Lagrangian Finite elements

We use linear mapping as defined earlier (2.5).
The covariant transformation is very simple:

$$\Phi(\mathbf{x}) = \hat{\Phi}(\hat{\mathbf{x}})$$

and we simply obtain:

$$\begin{aligned} \mathcal{A}_k^c &= A \\ &= N_{ref}^T W N_{ref} \end{aligned} \tag{3.16}$$

where N_{ref} is similar in structure to N but shape functions are evaluated on reference cell instead of real cell

3.2.7 Mapping : Raviart Thomas elements

We use Piola transformation mapping as defined earlier (2.6).

In this case, we find that there is no coupling between different components of reference cell gradient evaluation. We are able to get a well separated form as we would desire (3.9) for the matrix free framework. However due to our previous limitation (3.2.4), we continue to use cartesian mesh cells.
Let us consider a case where dim=3. Denote the coefficients as $j_i = j_i(\hat{\mathbf{x}}) \frac{1}{|J(\hat{\mathbf{x}})|}, \quad i = 1, 2, 3$. Clearly:

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} j_1 \hat{\mathbf{\Phi}}^{\mathbf{1}}(\hat{\mathbf{x}}) \\ j_2 \hat{\mathbf{\Phi}}^{\mathbf{2}}(\hat{\mathbf{x}}) \\ j_3 \hat{\mathbf{\Phi}}^{\mathbf{3}}(\hat{\mathbf{x}}) \end{bmatrix}$$

Using such a form, we obtain the transformation matrix from real to reference cell as:

$$\begin{aligned} N &= N_{ref} \odot H \\ A &= N_{ref}^T \widetilde{W} (N_{ref} \odot H) \end{aligned} \tag{3.17}$$

H has a repetitive structure and its elements depends on the component as in:

$$H(c1) = \begin{bmatrix} j_1(\hat{\mathbf{x_1}}) & j_1(\hat{\mathbf{x_1}}) & .. & j_1(\hat{\mathbf{x_1}}) \\ j_1(\hat{\mathbf{x_2}}) & j_1(\hat{\mathbf{x_2}}) & .. & j_1(\hat{\mathbf{x_2}}) \\ ... & ... & ... & \\ j_1(\hat{\mathbf{x_Q}}) & j_1(\hat{\mathbf{x_Q}}) & .. & j_1(\hat{\mathbf{x_Q}}) \end{bmatrix} H(c2) = \begin{bmatrix} j_2(\hat{\mathbf{x_1}}) & j_2(\hat{\mathbf{x_1}}) & .. & j_2(\hat{\mathbf{x_1}}) \\ j_2(\hat{\mathbf{x_2}}) & j_2(\hat{\mathbf{x_2}}) & .. & j_2(\hat{\mathbf{x_2}}) \\ ... & ... & ... & \\ j_2(\hat{\mathbf{x_Q}}) & j_2(\hat{\mathbf{x_Q}}) & .. & j_2(\hat{\mathbf{x_Q}}) \end{bmatrix}$$

$$H(c3) = \begin{bmatrix} j_3(\hat{\mathbf{x_1}}) & j_3(\hat{\mathbf{x_1}}) & .. & j_3(\hat{\mathbf{x_1}}) \\ j_3(\hat{\mathbf{x_2}}) & j_3(\hat{\mathbf{x_2}}) & .. & j_3(\hat{\mathbf{x_2}}) \\ ... & ... & ... & \\ j_3(\hat{\mathbf{x_Q}}) & j_3(\hat{\mathbf{x_Q}}) & .. & j_3(\hat{\mathbf{x_Q}}) \end{bmatrix}$$

and $\widetilde{W} = (H^T \odot W)$ which is possible because $\widetilde{W}$ is diagonal.

3.2.8 Operator $\mathcal{B}$

Next we consider cell based operator $\mathcal{B}_k$ which corresponds to the weak form:

$$b(\mathbf{u}, q) = -(\nabla \cdot \mathbf{u}, q)$$

Denoting by $\Psi_j, j = 1, ...n_{qh}$ the basis functions for space Q_h, then the operator $\mathcal{B}_k$ evaluates $(\nabla \cdot \mathbf{u}, \Psi_j) \quad \forall j$.
For simplicity, we denote $\mathcal{B}_k$ as B. Then, in matrix form, this can be written as:

$$B = \begin{bmatrix} (\nabla \cdot \mathbf{\Phi_1}, \Psi_1) & (\nabla \cdot \mathbf{\Phi_2}, \Psi_1) & ... & (\nabla \cdot \mathbf{\Phi_{n_{sh}}}, \Psi_1) \\ (\nabla \cdot \mathbf{\Phi_1}, \Psi_2) & (\nabla \cdot \mathbf{\Phi_2}, \Psi_2) & ... & (\nabla \cdot \mathbf{\Phi_{n_{sh}}}, \Psi_2) \\ ... & ... & ... & ... \\ ... & ... & ... & ... \\ (\nabla \cdot \mathbf{\Phi_1}, \Psi_{n_{qh}}) & (\nabla \cdot \mathbf{\Phi_2}, \Psi_{n_{qh}}) & ... & (\nabla \cdot \mathbf{\Phi_{n_{sh}}}, \Psi_{n_{qh}}) \end{bmatrix}$$

An element in row i and column j of the matrix $\mathcal{B}_k$ is given by:

$$\begin{aligned} B(i,j) &= (\nabla \cdot \mathbf{\Phi_j}, \Psi_i) \\ &= \int_K \nabla \cdot \mathbf{\Phi_j} . \Psi_i \\ &= \int_K trace(\nabla \mathbf{\Phi_j}) . \Psi_i \qquad = \sum_{q=1}^{n_Q} trace(\nabla \mathbf{\Phi_j}) . \Psi_i |J(\hat{\mathbf{x}}_q)| w_q \end{aligned}$$

where . denotes the usual product of two scalar values. The last form is obtained by transformation of integral to reference cell and replacing with appropriate quadrature rule. using this, the operator matrix can be factorized as:

$$B = N^T W S \tag{3.18}$$

where N is $n_Q \times n_{qh}$ matrix of pressure shape functions and W is same as earlier. S is a matrix which contains divergence evaluation for each shape function at all quadrature points on real cell.

For operator evaluation, transformation from real cell to reference cell is needed.

$$\Psi(\mathbf{x}) = \hat{\Psi}(\hat{\mathbf{x}})$$

For Lagrangian Finite elements, this allows us to simply replace N with N_{ref}. For S, we note that $\nabla \cdot \mathbf{\Phi_j} = trace(\nabla \mathbf{\Phi_j})$, and therefore we use covariant transformation or Piola transformation depending on the Finite element space we are dealing with.

3.2.9 Operator $\mathcal{B}^T$

The last operator which we study is cell-based operator $\mathcal{B}^T$ which evaluates the weak form:

$$b(\mathbf{v}, p) = -(\nabla \cdot \mathbf{v}, p)$$

Denote

$$p(\mathbf{x}) = \sum_{i=1}^{n_{qh}} \Psi_i(\mathbf{x}) p^i$$

then the operator $\mathcal{B}_T$ evaluates $(\nabla \cdot \mathbf{\Phi_i}, p), i = 1, ...n_{sh}$

Going by this way it turns out that the resulting factorization will be quite complex. We instead note that the weak form can be written as:

$$\begin{aligned} b(\mathbf{v}, p) &= -(\nabla \cdot \mathbf{v}, p) \\ &= -(\nabla \mathbf{v}, \mathbb{I}.p) \end{aligned}$$

This allows us to combine bilinear forms $a(\mathbf{u}, \mathbf{v})$ and $b(\mathbf{v}, p)$ as:

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) &= (\nabla \mathbf{u}, \nabla \mathbf{v}) + -(\nabla \mathbf{v}, \mathbb{I}.p) \\ &= (\nabla \mathbf{u} - \mathbb{I}.p, \nabla \mathbf{v}) \end{aligned} \tag{3.19}$$

This is what we will make use of in our work.

3.3 Tensorial evaluation of shape functions

The finite elements which we consider in this work have a tensor product form in each component. Further the quadrature points also show tensor product structure. We want to utilize this information in the evaluation of N_{ref} and D_{ref} matrices.

Denoting by r, s and t the individual space dimensions.
Let us first consider in 2-dimensions.

From one-dimensional Gauss quadrature rule, we construct multi-dimensional rule by a tensor product as:

- quadrature point $\hat{\mathbf{x}}_q = (r_{q1}, s_{q2})$
- quadrature weight $w_q = w_{q1} w_{q2}$

Denote by $\hat{\Phi} := \hat{\mathbf{\Phi}}^c$ the per component evaluation of a shape function, we notice that:

$$\{\hat{\Phi}_1, \hat{\Phi}_2, ..., \hat{\Phi}_{n_{sh}}\} = \{\phi_s \otimes \phi_r\}$$

where

$$\phi_r = \{\phi_{r1}, ..., \phi_{rl}\} \quad \phi_s = \{\phi_{s1}, ..., \phi_{sm}\}$$

are the 1-D shape functions and $l \times m = n_{sh}$

With this result, we have the factorization of N_{ref} as:

$$\begin{aligned} N_{ref}(2D) &= N^s_{1D} \otimes N^r_{1D} \\ N_{ref}(3D) &= N^t_{1D} \otimes N^s_{1D} \otimes N^r_{1D} \end{aligned} \tag{3.20}$$

The same concept can be used to find out factorization of D_{ref}.
D_{ref} can be written as $D_{ref} = [D^r_{ref}\ D^s_{ref}]$

$$\begin{aligned} D^r_{ref}(2D) &= N^s_{1D} \otimes D^r_{1D} \\ D^s_{ref}(2D) &= D^s_{1D} \otimes N^r_{1D} \end{aligned} \tag{3.21}$$

and

$$\begin{aligned} D^r_{ref}(3D) &= N^t_{1D} \otimes N^s_{1D} \otimes D^r_{1D} \\ D^s_{ref}(3D) &= N^t_{1D} \otimes D^s_{1D} \otimes N^r_{1D} \\ D^t_{ref}(3D) &= D^t_{1D} \otimes N^s_{1D} \otimes N^r_{1D} \end{aligned} \tag{3.22}$$

3.3.1 Tensorial evaluation for Lagrangian Finite elements

In this case, we have simply

$$\begin{aligned} N^s_{1D} &= N^r_{1D} = N^t_{1D} \\ D^s_{1D} &= D^r_{1D} = D^t_{1D} \end{aligned}$$

3.3.2 Tensorial evaluation for Raviart Thomas Finite elements

In this case, because of the structure of Raviart-Thomas polynomial, we have:

- 1-D shape function matrices and 1-D shape gradient matrices are different in two space dimensions
- Because of a certain symmetry, these 1-D matrices are identically used in different component. Although they are used in different space dimensions in each component

There is another point which is specific to the manner in which Raviart Thomas basis functions are implemented in the dealii library. This is mentioned in dealii documentation of compute_node_matrix and Generalized support points

The basis of Raviart Thomas shape functions on the reference cell is chosen in such a way that the node functional evaluation on this basis is equal to Kronecker delta function. If σ_i denotes node functionals, we want to have the property:

$$\sigma_i(\hat{\mathbf{\Phi}}_j) = \delta_{ij}$$

For practical reasons we use a basis which is tensor product of Lagrange polynomials. Let us call this as "raw" basis consistent with the terminology of dealii. Note that the node functionals do not evaluate to kronecker delta on this raw basis. Therefore we need to transform from raw basis to the actual basis.

$$\hat{\mathbf{\Phi}}_i = \sum_{j=1}^{n} c_{ij} \hat{\mathbf{\Phi}}_{raw}, j \tag{3.23}$$

where c_{ij} are expansion coefficients in the basis transformation matrix $\mathcal{C}$

Recall that Raviart-Thomas polynomials have *independent* tensor product structure in each component with no coupling between components. This has implication that every shape function (in any chosen basis) will be strictly zero in all but one component. This implies that the matrix $\mathcal{C}$ will be block diagonal. A small visualization for 3-dimensions case will help here:

$$\begin{bmatrix} \hat{\Phi}_1 \\ \dots \\ \hat{\Phi}_n \\ \hat{\Phi}_{n+1} \\ \dots \\ \hat{\Phi}_{2n} \\ \hat{\Phi}_{2n+1} \\ \dots \\ \hat{\Phi}_{3n} \end{bmatrix} =$$

$$\begin{bmatrix}
c_{1,1} & c_{1,2} & \cdots & c_{1,n} & & & & & & \\
\cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & & & & & & \\
c_{n,1} & c_{n,2} & \cdots & c_{n,n} & & & & & & \\
 & & & & c_{n+1,n+1} & c_{n+1,n+2} & \cdots & c_{n+1,2n} & & \\
 & & & & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & & \\
 & & & & c_{2n,n+1} & c_{2n,n+2} & \cdots & c_{2n,2n} & & \\
 & & & & & & & & c_{2n+1,2n+1} & c_{2n+1,2n+2}\cdots c_{2n+1,3n} \\
 & & & & & & & & \cdot\cdot & \cdot\cdot\cdot\cdot\cdot\cdot \\
 & & & & & & & & c_{3n,2n+1} & c_{3n,2n+2}\cdots c_{3n,3n}
\end{bmatrix}$$

$$\begin{bmatrix}
\hat{\Phi}_{1,raw} \\
\cdots \\
\hat{\Phi}_{n,raw} \\
\hat{\Phi}_{n+1,raw} \\
\cdots \\
\hat{\Phi}_{2n,raw} \\
\hat{\Phi}_{2n+1,raw} \\
\cdots \\
\hat{\Phi}_{3n,raw}
\end{bmatrix}$$

with

$$\mathcal{C} = \begin{bmatrix} C1 & & \\ & C2 & \\ & & C3 \end{bmatrix}$$

Such a block-diagonal structure allows us to work on component wise manner as we would like to. For any component, we have:

$$\begin{aligned}
N_{ref} &= C_c N_{ref,raw} \\
N^s_{1D} \otimes N^r_{1D} &= C_c(N^s_{1D,raw} \otimes N^r_{1D,raw}) \\
&= (\mathcal{X} \otimes \mathcal{Y})(N^s_{1D,raw} \otimes N^r_{1D,raw}) \\
&= (\mathcal{X} N^s_{1D,raw}) \otimes (\mathcal{Y} N^r_{1D,raw})
\end{aligned} \tag{3.24}$$

where the last two equalities follows from the mixed-product property of the Kronecker product of matrices. From this we get that:

$$\begin{aligned}
N^s_{1D} &= \mathcal{X} N^s_{1D,raw} \\
N^r_{1D} &= \mathcal{Y} N^r_{1D,raw}
\end{aligned} \tag{3.25}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are of appropriate dimensions.

Using similar arguments, we also obtain that:

$$\begin{aligned}
D^s_{1D} &= \mathcal{X} D^s_{1D,raw} \\
D^r_{1D} &= \mathcal{Y} D^r_{1D,raw}
\end{aligned} \tag{3.26}$$

This can be naturally extended in 3-dimensions.

3.4 Sum Factorization

For efficient tensor product evaluations, a technique called sum-factorization is used. We refer the reader to [1] and briefly mention the concept here.
Consider the product

$$z = (Y \otimes X)u$$

where matrices X, Y and vector u are of compatible dimensions.
Such a computation can be factored out as the product of the three matrices:

$$Z = XUY^T \tag{3.27}$$

where U is a matrix in which elements from u have been laid out in fortran style column major order. The number of columns of U is equal to number of columns of Y.
Finally, the matrix Z contains elements from z which have been laid out in fortran style column major order. The number of rows of Z is equal to number of rows of X.
Benefits:

- Reduced cost. For square matrices, cost reduces from $\mathcal{O}(m^4)$ to $\mathcal{O}(m^3)$
- For matrix free operator evaluation examples, this is directly applicable and is used in the implementation

Chapter 4
Software implementation Methodology

The software implementation discussed here is available as open-source in author's `github` repository under *immutable* commit IDs:

- `https` link to commit id: `473a4fef9ce`
- `https` link to commit id: `421f9873e`

4.1 Components of Matrix Free framework

4.1 and 4.2 show the component diagram of the existing Matrix Free framework.

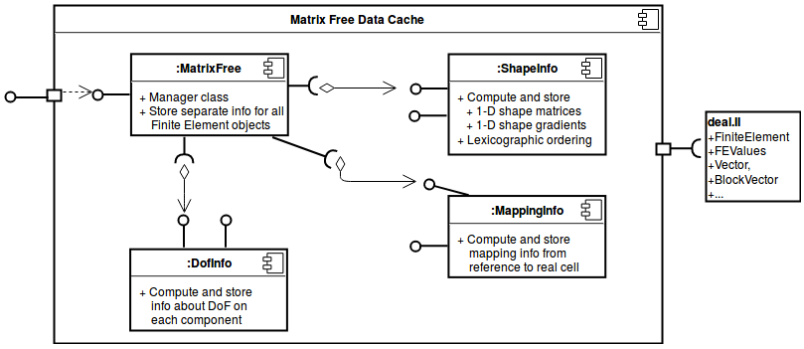

Fig. 4.1 Architecture for MatrixFree Data cache

In our work, we retain the same architecture and extend the design of different components to support Raviart Thomas elements in 2-dimensions. The design choices were kept generic by considering future extensions 6.2.

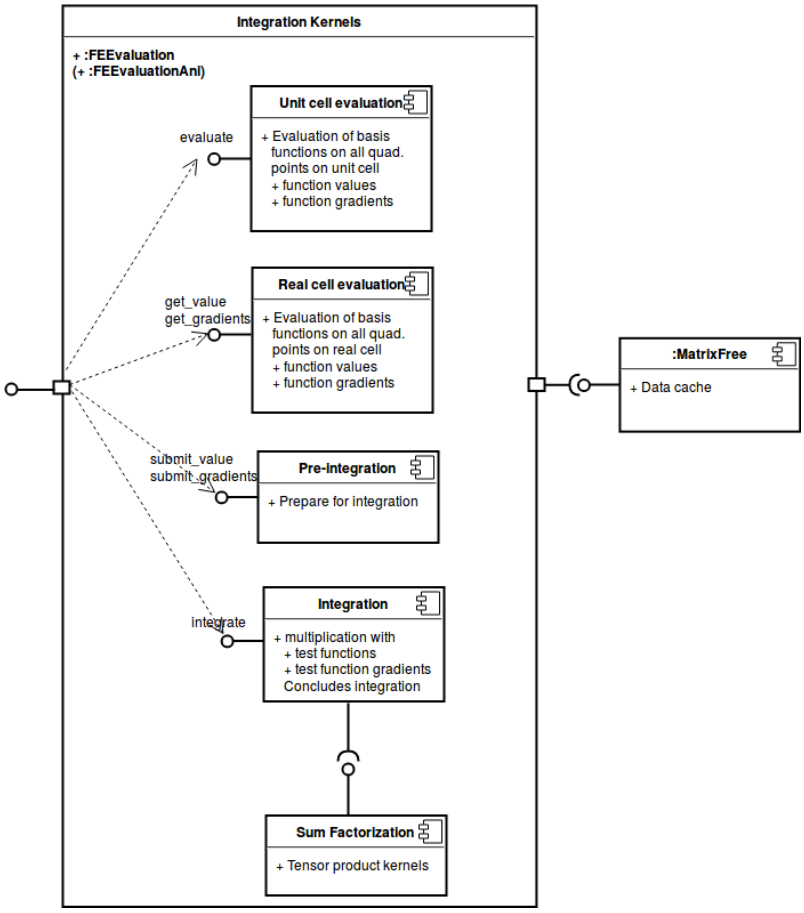

Fig. 4.2 Architecture for Integration kernels

4.2 Matrix Free Data Cache

Class MatrixFree This class manages the computation and storage of data which is needed for matrix free implementation. The data is stored in aggregated classes. There is little impact to this class:

- The class now maintains the choice of mapping - covariant (for Lagrangian FE) or Piola transformation (for Raviart-Thomas FE)

Class MappingInfo This class stores the geometry dependent data for interior cells. No impact.

Class DoFInfo This class stores information related to degrees of freedom on each component. No impact.

4.2.1 Class ShapeInfo

This class stores the shape function values, gradients and Hessians evaluated for the tensor product finite element and tensor product quadrature formula on the reference cell. We only need to store 1-D data. The tensor product is performed in integration kernels.

Storage of multi-dimensional, multi-component data The actual tensorial evaluation is repetitively performed in the integration kernels by using the data stored in ShapeInfo. Processor cache utilization is an important factor in such a case. Therefore, the storage of shape info should be optimized as much as possible.
Lagrangian finite elements are isotropic and have same structure in all components. So the storage of 1-D data along one coordinate direction is enough. In contrast, the structure of Raviart-Thomas element (RT_l with degree l) poses several challenges.

- anisotropy along directions
- different tensor product structure in different components

However, if we observe carefully, we find that there is some symmetry in this element:

- In any component, the degree of 1-D shape functions is equal ($=l$) for all except one direction ($=l+1$)
- Across different components, the structure is simply rotated in coordinates

Using this knowledge it is clear that irrespective of dimension and degree of Raviart-Thomas element, we only need 1-D shape info along these two coordinate directions.
We define new data structures:

1. `base_shape_values` and `base_shape_gradients`: These store shape function values and gradients respectively for 1-D polynomial shape functions in two coordinate directions
   - one with degree $l+1$ evaluated for `n_q_points_1d` quadrature points
   - second with degree l evaluated for `n_q_points_1d` quadrature points
2. `shape_values_vec` and `shape_values_gradients` : These store the knowledge of anisotropy. The following diagram (4.3) illustrates the relation:

Generation of 1-D shape function values and gradients on unit interval For practical reasons, we want to re-use the existing software modules as much as possible. The current implementation of Raviart-Thomas Finite element in deal.ii is such that the shape functions on reference cell are generated from *raw* basis functions after basis transformation. The raw basis

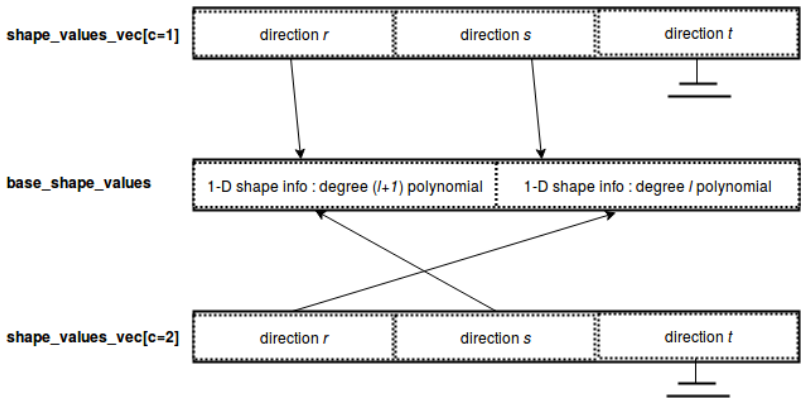

Fig. 4.3 Shape info storage for Raviart-Thomas element in 2-dimensions

functions are tensor product of 1-D lagrange polynomials. As mentioned in section 3.3.2, for generation of corresponding 1-D shape functions needed for matrix free, the following high-level steps are needed:

- Evaluate raw basis functions
- Find out the basis transformation matrix $\mathcal{C}$
- Identify tensorial structure in $\mathcal{C}$ and evaluate 1-D shape functions from 1-D *raw* shape functions

Each step will now be discussed in detail.

Evaluation of raw basis functions: Although deal.ii already provides this functionality, but it was found that the resulting basis functions are not indexed properly. In 2-dimensions, we expect a structure with tensor product as $(Q_{l+1}(r) \otimes Q_l(s)) \times (Q_l(r) \otimes Q_{l+1}(s))$. Instead what we get is $(Q_{l+1}(r) \otimes Q_l(s)) \times (Q_{l+1}(s) \otimes Q_l(r))$

Algorithm 4.1 : Reorder Raviart-Thomas *raw* basis functions (2-dim) Pre-condition: input polynomials are generated with tensor product such that outer index runs fastest
$\{p_1, p_2, p_3\} \otimes \{q_1, q_2\} = \{p_1q_1, p_2q_1, p_3q_1, p_1q_2, p_2q_2, p_3q_2\}$

1. If (First component)
   a. Polynomials corresponding to first component are already in expected order
   b. new index = current index
2. If (Second component)
   a. Store the polynomials in $(l+1) \times l$ matrix in C-row major format
   b. Transpose the matrix
   c. new index = location in the transposed matrix

Output: reordered index mapping for *raw* basis polynomials
This has been implemented in `PolynomialsRaviartThomas<dim>::create_poly_mapping ()`

Finding out basis transformation matrix $\mathcal{C}$: Again, although deal.ii already provides this functionality, yet it was found that the basis transformation matrix does not show the block diagonal structure as explained in section 3.3.2. The task is to develop a procedure so that we can generate this matrix with the required structure.

For this, we need to answer two questions:
1. *How to evaluate shape functions in actual basis?* - We can create basis transformation matrix by clever use of linear node functionals. Since we decided to choose shape functions with the property that $\sigma_i(\hat{\mathbf{\Phi}}_\mathbf{j}) = \delta_{ij}$, we have that:

$$\mathit{\Pi}(\hat{\mathbf{\Phi}}) = I \quad \text{where} \quad \mathit{\Pi}(\hat{\mathbf{\Phi}})(i,j) = \sigma_i(\hat{\mathbf{\Phi}}_\mathbf{j})$$

For more details, reader is referred to `FETools::compute_node_matrix`

2. *In which order should node functionals be evaluated?* - In general this is immaterial and therefore ignored in current implementation of dealii. However, since we want that the transformation matrix with block diagonal structure, this is required.
There is a natural ordering of node functionals (=the degrees of freedom) for `Hdiv` elements which comes from the tensor product of constituent 1-D functions. We refer the reader to [14] section 2.4.3 and 2.4.4 for details. A visualization for RT_1 in 2-dimensions will illustrate the concept (4.4):

- In first component (direction r), the tensor structure is $(Q_2 \otimes Q_1)$. The first 1-D basis will have 3 degrees of freedom associated with vertices and interior. The second 1-D basis function will have 2 degrees of freedom, both associated with interior.

- In second component (direction s), the tensor structure is $(Q_1 \otimes Q_2)$. The first 1-D basis will have 2 degrees of freedom, both associated with interior while the second 1-D basis function will have 3 degrees of freedom associated with vertices and interior.

By following the rule that in a tensor product, the outer space runs fastest, we get a natural indexing as shown here (4.4):
We are now ready to formulate the algorithm.

Algorithm 4.2 : Generate block diagonal basis transformation matrix (2-

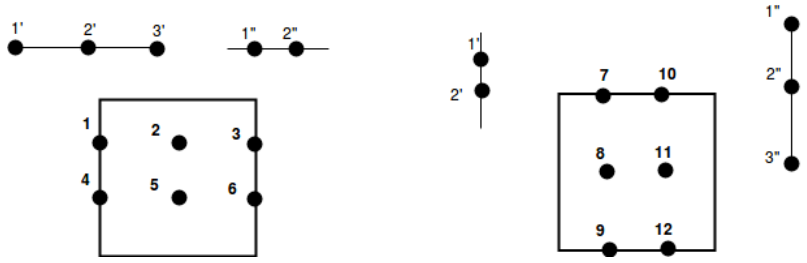

Fig. 4.4 Geometrical location and indexing of Raviart-Thomas degrees of freedom in 2-dimensions

dim)
Precondition: *raw* basis functions in correct order as per (4.2.1)

1. Loop over components
   a. Arrange the node functionals in natural tensor product order. This corresponds to column of Π matrix
   b. The arrangement of *raw* basis functions should correspond to row of Π matrix
   c. Evaluate $\Pi(\hat{\mathbf{\Phi}}^{raw})(i,j) = \sigma_i(\hat{\mathbf{\Phi}}^{raw}_j(\hat{x}))$
   d. We have that $\mathcal{C} = (\Pi(\hat{\mathbf{\Phi}}^{raw}))^{-1}$

Output: $\mathcal{C}$ in block diagonal form with each block having kronecker product structure
This has been implemented in `ShapeInfo::internal_reinit_vector` function. The functionality currently works for only for first three Raviart-Thomas elements.

dof index renumbering for input and output vectors (lexicographic renumbering) By using $\mathcal{C}$, we will get the shape functions in tensor product order. But this is not the natural order which is used for shape functions in deal.ii. For the Operator-vector `vmult` operations using matrix free, the dof values must be re-ordered from deal.ii ordering to tensor product order.
Such a concept is also used in matrix free framework for Lagrangian Finite elements where it is (obviously) called `lexicographic renumbering`. We retain the same terminology for Raviart-Thomas elements.
This has been implemented in `ShapeInfo::raviart_thomas_lexicographic_renumber` function. The functionality currently works for only for first three Raviart-Thomas elements.

4.3 Integration Kernels

4.3.1 Class `FEEvaluationAni`

This class has been added as a parallel to `FEEvaluation` class. It provides functionality necessary to evaluate functions at quadrature points and cell integrations for *anisotropic* tensor product Finite Element. By a judicious combination of the interfaces provided by this class, several different Finite element operators can be evaluated on the input vector. Currently it supports:

- Raviart-Thomas finite element in 2-dimensions
- For experimentation purposes, evaluation of isotropic Lagrangian FE in 2/3 dimensions

A small discussion on the design approach for this class now follows.

Policy based design :

The signature for `FEEvaluation` is:
`template <int dim, int fe_degree, int n_q_points_1d, int n_components_,typename Number >` class FEEvaluation‘

The figure (4.5) shows how the template information is percolated for isotropic vector valued Lagrangian finite elements

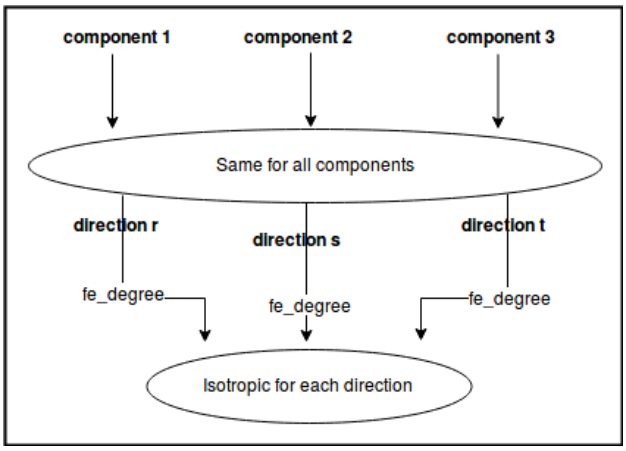

Fig. 4.5 FEEvaluation top-to-bottom view

In comparison, the figure (4.6) shows how the information is required for anisotropic vector valued finite elements

Clearly, we need the information about different `fe_degrees` to be available to the deepest level. Further:

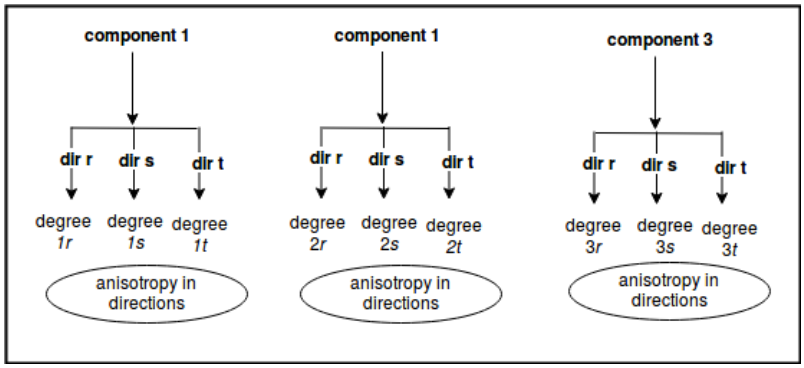

Fig. 4.6 FEEvaluationAni top-to-bottom view

- it has been mentioned in the [10] that using `fe_degree` as compile time parameter allows efficient loop unrolling and a huge performance impact
- we would like the new interface generic enough to be able to support other anisotropic finite elements in future

For the above reasons, we have chosen policy based class design ([3]) where the type of Finite Element is used as template policy to determine those compile time characteristics which are needed for fast and efficient evaluation.

The signature for `FEEvaluationAni` is:

```
template <typename FEType, int dim, int base_fe_degree,
int n_q_points_1d, typename Number > class FEEvaluationAni
```

- `FEType` is provided as `FE_RaviartThomas<dim>` (or `FE_Q<dim>`)
- `n_components` and `fe_degrees` for each direction are derived from `FEType` using traits

4.3.2 Sum Factorization

'Class FEEvaluationImplAni' : This is where sum factorization is actually performed. This class is a parallel to `FEEvaluationImpl`. Notable differences are:

- Sum factorization for anisotropic tensor product using `apply_anisotropic` function
- Operates on a single component. The component number is made template parameter. Caller has to run compile time loop on components
- Currently supports evaluation of values and gradients in 1 and 2 dimensions

'apply_anisotropic' : This is parallel to `EvaluatorTensorProduct::apply` method. These methods provide fast matrix-matrix multiplication kernels. They are used in conjunction with `FEEvaluationImplAni` for sum factorization evaluation for tensor product form e.g. : $z = (Y \otimes X)u$

- `apply` method works on matrices of similar dimensions, in 1, 2 and 3-dimensions
- `apply_anisotropic` method works on cases where $row(X) \neq row(Y); col(X) = col(Y)$ in 1 and 2-dimensions. This is sufficient since we use kronecker product of 1-D shape matrices for which columns (= number of quadrature points in 1-D) are equal

There are two fundamental matrix-matrix multiplication operations which are provided by these kernels:

- Evaluation of $\mathcal{U}X$ (and $\mathcal{U}X^T$) where $\mathcal{U} = U^T$
- Evaluation of $Y^T\mathcal{W}$ (and $Y\mathcal{W}$)

These operations provide facility to evaluate sum-factorization (3.4, 3.3.2) for all the operators which we have considered in 3.
The common forms from these operators is:

- Case 1:

$$Z = X^T UY = X^T \mathcal{U}^T Y = (\mathcal{U}X)^T Y = (Y^T W)^T$$

where $W = \mathcal{U}X$

- Case 2:

$$Z = XUY^T = X\mathcal{U}^T Y^T = (\mathcal{U}X)^T Y^T = (YW)^T$$

where $W = \mathcal{U}X^T$

- Case 3:

$$Z = XUY = X\mathcal{U}^T Y = (\mathcal{U}X^T)^T Y = (Y^T W)^T$$

where $W = \mathcal{U}X^T$

The last two operations in each case correspond to sum-factorization in direction r and direction s.

Algorithm 4.3 : anisotropic matrix-matrix product kernel (2-dimensions)
Precondition: Matrices X, Y and vector u are available
We refer the reader to actual implementation of `apply_anisotropic` kernel which is small and self explanatory. Here we talk about template parameters and what they mean in the context of 3 cases above.

- `dim` - space dimensions
- `fe_degree` - Degree of 1-D shape functions. For Raviart-Thomas, it depends on the direction in which sum-factorization is being performed

- `n_q_points_1d` - Must be same for each 1-D shape element
- `add` - whether final result be added to existing values in output buffer
- `direction` - See table below
- `dof_to_quad` - See table below
- `inter_dim` - Provides compatibility between `dir0` and `dir1`. See table below

The following table provides the values for these template parameters for the three cases defined above:

Case 1

`dir`	`dof_to_quad`	`inter_dim`
0	TRUE	rows(Y) (= rows($\mathcal{U}$)
1	TRUE	cols(X) (=cols(W))

Case 2

`dir`	`dof_to_quad`	`inter_dim`
0	FALSE	cols(Y) (= rows($\mathcal{U}$)
1	FALSE	rows(X) (=cols(W))

Case 3

`dir`	`dof_to_quad`	`inter_dim`
0	FALSE	rows(B) (= rows($\mathcal{U}$)
1	TRUE	rows(A) (=cols(W))

4.3.3 Others

Reference cell evaluation :

- The interface in `FEEvaluationAni` remains the same as in `FEEvaluation` as `evaluate` method
- Evaluation is performed using `SelectEvaluatorAni` instead of `SelectEvaluator`

Real cell evaluation :
The methods `get_value` and `get_gradients` apply Piola transformation if operation is requested for Raviart-Thomas finite elements

Pre integration : No impact

Integration

- The interface in `FEEvaluationAni` remains the same as in `FEEvaluation` as `integrate` method
- Evaluation is performed using `SelectEvaluatorAni` instead of `SelectEvaluator`
- Sum factorization is performed as explained in (4.3.2)

Chapter 5
Numerical results

In this section we investigate the numerical performance of our approach. Finally, we collectively discuss the findings.

Results were taken on a machine with following specifications:

Quantity	Rating
Processor	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
CPU count	8 (4 core, 2 threads/core)
Hyperthreading	On
System memory	7832 Mib
L1D cache	32K
L1I cache	32K
L2 cache	256K
L3 cache	8192K

5.1 Operator evaluation accuracy

Objective To compare the result of Operator-vector multiplication evaluated using matrix free framework with the usual deal.ii Sparse Matrix-vector multiplication.

Test specifications

- Raviart Thomas elements in 2 dimensions, RT_0, RT_1 and RT_2
- square mesh (-1,1) refined 6 times, giving total of 4096 cells
- All the operators as described in section 3.2

Findings The following table shows the relative error accuracy for the operators of Mixed diffusion problem.

Operator	RT0	RT1	RT1
A	3.19E-15	2.93E-16	7.23E-16
B^T	1.46E-15	2.26E-16	6.68E-16
B	7.64E-17	2.71E-16	5.20E-16

The following table shows the relative error accuracy for the operators of Stokes problem.

Operator	RT0	RT1	RT1
A	6.14E-15	2.18E-16	7.44E-16
B^T	6.15E-15	2.36E-16	7.53E-16
B	7.64E-17	2.26E-16	5.2E-16

5.2 Performance results

Objective To evaluate performance of operator evaluation and analyze weaknesses.

Test specifications

- All the operators as described in section 3.2
- 2 dimensions, degree 0,1 and 2 for Raviart Thomas elements and 1,2 and 3 for FE_Q elements
- Mesh refined until $\geq 10^6$ degrees of freedom were obtained
- All calculations in double precision arithmetic
- Three set of results are taken:
  - `FE_Q(k) MF` - FE_Q vector valued elements evaluated using existing matrix free framework - basically isotropic evaluation
  - `FE_Q(k) anisotropic MF` - FE_Q vector valued elements evaluated using our extensions to matrix free framework - basically anisotropic evaluation of isotropic elements
  - `RT(k-1) MF` - Raviart-Thomas elements using our modifications to matrix free framework
- Time measurements are given as average CPU time in seconds/million degrees of freedom
- Cache utilization was simulated using Valgrind Cachegrind tool ([15])

Findings The measurements for time benchmarking are given in below table and summarized in figures 5.1, 5.2, 5.3

Operator A						
Mixed diffusion equation				Stokes equation		
Degree, k	1	2	3	1	2	3
isotropic `FE_Q(k)`	0.01931	0.00934	0.01419	0.03148	0.01671	0.02531
anisotropic `FE_Q(k)`	0.04113	0.08114	0.13372	0.05829	0.12047	0.20151
Raviart-Thomas `(k-1)`	0.06281	0.04296	0.08234	0.10016	0.06381	0.12411

Operator B^T						
Mixed diffusion equation				Stokes equation		
Degree, k	1	2	3	1	2	3
isotropic `FE_Q(k)`	0.03895	0.01978	0.03138	0.03943	0.01957	0.03011
anisotropic `FE_Q(k)`	0.0736	0.15112	0.25416	0.06723	0.13545	0.22987
Raviart-Thomas `(k-1)`	0.11782	0.0735	0.147	0.11969	0.07253	0.13839

Operator B						
Mixed diffusion equation				Stokes equation		
Degree, k	1	2	3	1	2	3
isotropic `FE_Q(k)`	0.02199	0.0105	0.01674	0.02199	0.0105	0.01674
anisotropic `FE_Q(k)`	0.0406	0.08481	0.14328	0.04031	0.08585	0.1428
Raviart-Thomas `(k-1)`	0.07098	0.04408	0.08423	0.06934	0.04455	0.08514

For impact analysis of differences in isotropic and anisotropic results, the measurements for L1 and LLC (lowest layer cache) utilization efficiency were taken. The results are summarized in the below table:

Results for Operator A (Stokes)					
		L1 miss rates		LL miss rates	
	Instructions	Instr	Data	Instr	Data
isotropic `FE_Q(k)`	150971578	0.01%	2.20%	0.00%	1.00%
anisotropic `FE_Q(k)`	175757984	0.01%	1.90%	0.00%	0.80%
	Branches	Misprediction rate			
isotropic `FE_Q(k)`	19876569	2.10%			
anisotropic `FE_Q(k)`	23869311	2.70%			

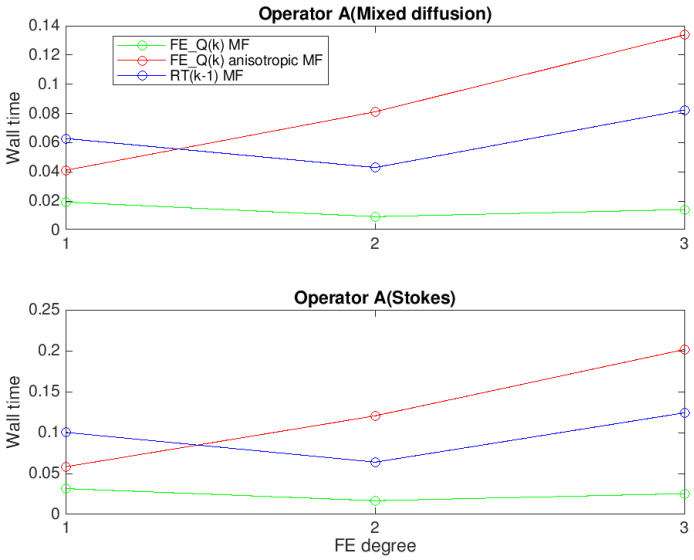

Fig. 5.1 Time benchmarks for operator A

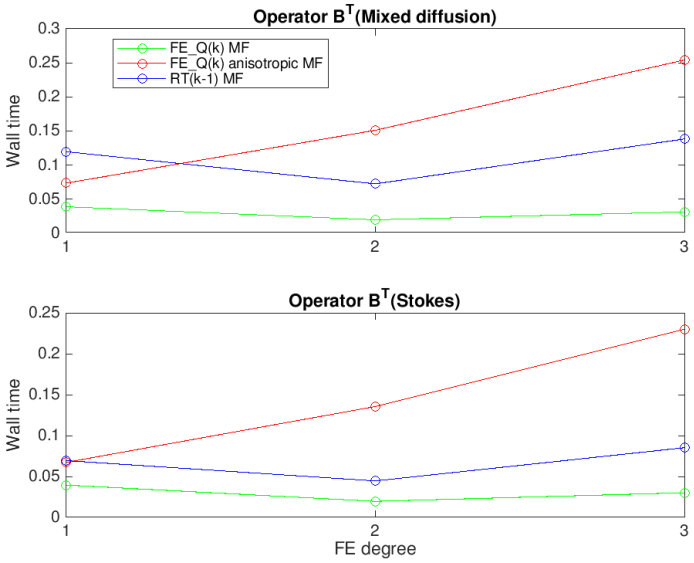

Fig. 5.2 Time benchmarks for operator B^T

Results for Operator A (Diffusion)					
		L1 miss rate		LL miss rate	
	Instruction references	Instr	Data	Instr	Data
isotropic `FE_Q(k)`	149269817	0.01%	2.2%	0.00%	1.0%
anisotropic `FE_Q(k)`	173448981	0.01%	2.0%	0.00%	0.8%
	Branches	Misprediction rate			
isotropic `FE_Q(k)`	19842191	2.1%			
anisotropic `FE_Q(k)`	23834922	2.7%			

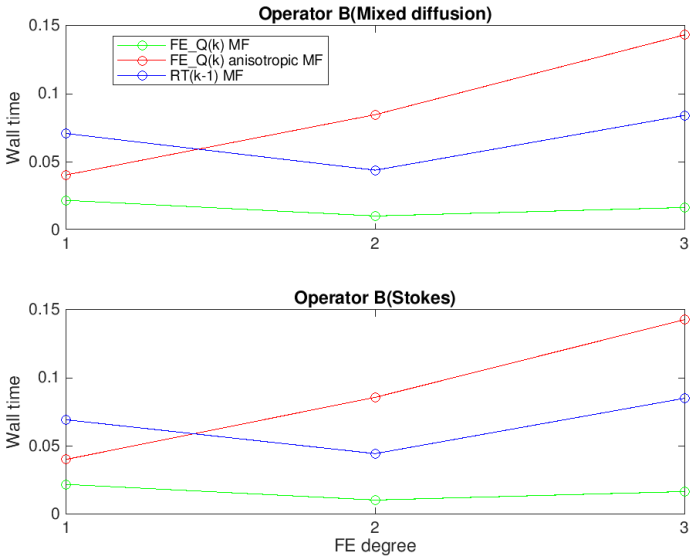

Fig. 5.3 Time benchmarks for operator B

5.3 Discussion

1. Operator evaluation accuracy is good for all practical purposes. The relative error is close to machine accuracy
2. The comparison of isotropic and anisotropic evaluation of `FE_Q` elements is helpful as:
   - to analyze impact of changes we have done
   - to get an estimate of lower bound for evaluation using Raviart-Thomas
3. The impact of extra calculations due to Piola transformation are visible in (5.1). Since this is double precision arithmetic, we see significant difference
4. It seems that evaluation of Raviart-Thomas elements is ≈ 3 times slower than using Lagrangian finite elements with comparable polynomial degree But the x-axis range in our test is small, so this result is not conclusive
5. There is almost no degradation in cache utilization efficiency when using anisotropic mode of matrix free
6. The increase in time can be attributed to $\approx 20\%$ increase in instructions and branches which are mostly double precision arithmetic Note that this figure is for `FE_Q` elements. For Raviart-Thomas elements, increase is expected to be $< 10\%$

5.4 Application to mixed diffusion equation

Objective To apply the modified matrix free framework on a PDE problem. We solve the mixed diffusion problem as specified in (3.3) using the modified matrix free framework.

Test specifications

- Raviart Thomas elements in 2 dimensions, RT_0, RT_1 and RT_2
- square mesh (-1,1) refined 6 times, giving total of 4096 cells
- Face evaluation is not supported. So, for simplicity the term $\langle p^D, \mathbf{v}\cdot\mathbf{n}\rangle_{\Gamma_D}$ is treated as constant
- f is chosen as non-constant function
- Minimum residual solver `SolverMinRes` as available in deal.ii was used

Results The following table summarizes the results.

Degree	0	1	2
CPU time, Sparse MV	0.57302	9.24938	111.564
Wall time, Sparse MV	0.099	1.17765	14.04
CPU time, matrix free	1.38516	9.597	49.6304
Wall time, matrix free	0.21423	1.21478	6.38564
Convergence iterations	808	1515	4281
Solution accuracy	1.43E-14	1.51E-14	1.3E-13

We are able to achieve the desired solution accuracy using our software. On the semilog plot (5.4), we can see beyond RT_2 the benefits of using matrix free approach are clearly visible. The wall clock time results are $> 50\%$ better than using sparse matrix-vector product approach. The same trend is also shown for total CPU time.

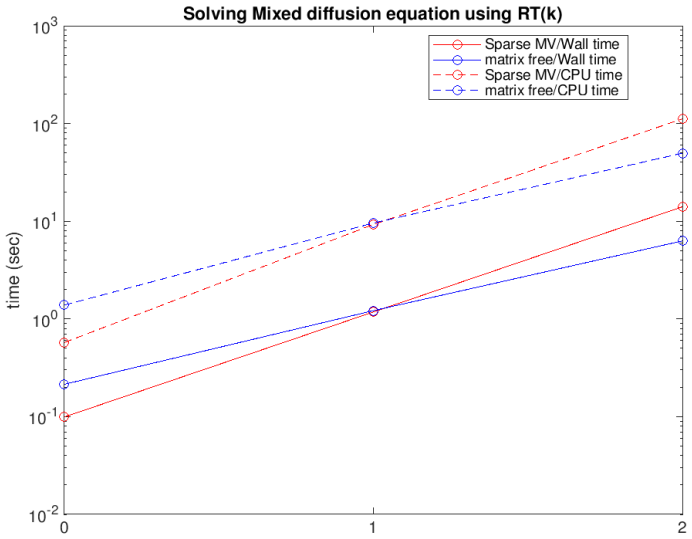

Fig. 5.4 Solution of Mixed Diffusion problem

Chapter 6
Conclusion

6.1 Summary

We developed extensions to matrix free framework to support anisotropic Raviart-Thomas Finite elements. After a brief review of operator notation in chapter 3.1, we turned to two mixed finite element problems (3.3) and (3.4) for theoretical development. This helped us to localize on the cell level operators which are the fundamental building blocks of matrix free approach. These included operators A, B^T and B from both of these problems.
This was the first place where we found that we can not directly support non-cartesian mesh for Raviart-Thomas elements in the the existing matrix free framework. However, cartesian mesh can be supported and we used this as our basic assumption to proceed.
In 4, we did an impact analysis of affected components in matrix free architecture. We found that identifying tensor product form for basis transformation matrix is a key point. It is at this point that we were limited to Raviart-Thomas degrees 0,1 and 2 with our new implementation.
Finally we analyzed that the sum factorization kernels need significant changes to support anisotropic elements.
Throughout the design, we strived for decisions which should permit addition of other anisotropic elements easier in future.
In 5, we checked the performance and reliability of our implementation. We were able to control the efficient usage of cache which is key to fast kernels in this approach.
Finally we verified the implementation by solving mixed diffusion equation and observed that the matrix-free approach is far more time efficient that usual sparse matrix-vector based approach.

This report was generated using `DocOnce` tool ([11])

6.2 Future extensions

Possibilities for future extensions:

1. Support Raviart-Thomas elements with degrees > 2
2. Support Raviart-Thomas elements in 3-dimensions
3. Develop concepts for face evaluation to fully support discontinuous Galerkin Stokes problem
4. Develop concepts to support non-cartesian mesh
5. Identify opportunities to utilizing massively parallel architectures like CUDA

Chapter 7
Acknowledgements

I would like to thank Prof. Dr. Guido Kanschat for his interesting lectures in finite element methods, and for accepting to be my supervisor for this thesis. Thanks to Dr. Daniel Arndt for suggesting this topic and for guiding me throughout the work. Further thanks to Julius Witte for helping me with a difficult part in this thesis.

I am extremely grateful to my family for the sacrifices they made to support me in my studies.

References

[1] .. *Efficient evaluation of weak forms in discontinuous Galerkin methods.* SPPEXA, 2017.

[2] Ern A. and Guermond JL. *Finite Element Interpolation. In: Theory and Practice of Finite Elements.* Applied Mathematical Sciences. SIAM, 2004.

[3] Alexandrescu and Andrei. *Modern C++ design: generic programming and design patterns applied.* Addison-Wesley Longman Publishing Co., Inc., 2001.

[4] D. Arndt, W. Bangerth, D. Davydov, and T. Heister and. The `deal.II` library, version 8.5. volume 25, pages 137–146, 2017.

[5] Boffi D., Brezzi F., and Fortin M. *Mixed Finite Element Methods and Applications.* Springer, 2013.

[6] Christian Grossmann, Hans-G. Roos, and Martin Stynes. *Numerical Treatment of Partial Differential Equations.* Springer, 2007.

[7] Kanschat Guido. *Lecture notes: Numerical methods for Partial Differential equations.* 2008.

[8] Kanschat Guido. *Lecture notes: Mixed finite elements.* 2017.

[9] Guido Kanschat and Youli Mao. Multigrid methods for hdiv-conforming discontinuous galerkin methods for the stokes equations. volume 23, pages 51–66, 2015.

[10] Martin Kronbichler and Katharina Kormann. A generic interface for parallel cell-based finite element operator application. volume 63, pages 135–147, 2012.

[11] Hans Petter Langtangen. *DocOnce A lightweight markup language - document once, include anywhere.* 2016.

[12] Ljungkvist and Karl. Finite element computations on multicore and graphics processors. 2017.

[13] McCool, Michael, Reinders, James, Robison, and Arch. *Structured Parallel Programming: Patterns for Efficient Computation, 1st edition.* Morgan Kaufmann Publishers Inc., 2012.

[14] A. T. T. McRae, G.-T. Bercea, L. Mitchell, D. A. Ham, , and C. J. Cotter. Automated generation and symbolic manipulation of finite elements. volume 38, pages S25–S47, 2016.

[15] Nethercote, Nicholas, Seward, and Julian. Valgrind: A framework for heavyweight dynamic binary instrumentation. volume 42, pages 89–100, 2007.

[16] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design, 4th edition*. Morgan Kaufmann, Burlington, 2009.

Declaration

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den März 12, 2018 **Saurabh Mehta**