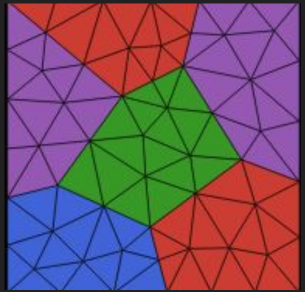


One year of `Ferrite.jl` development

Fredrik Ekre

Technische Universität Braunschweig

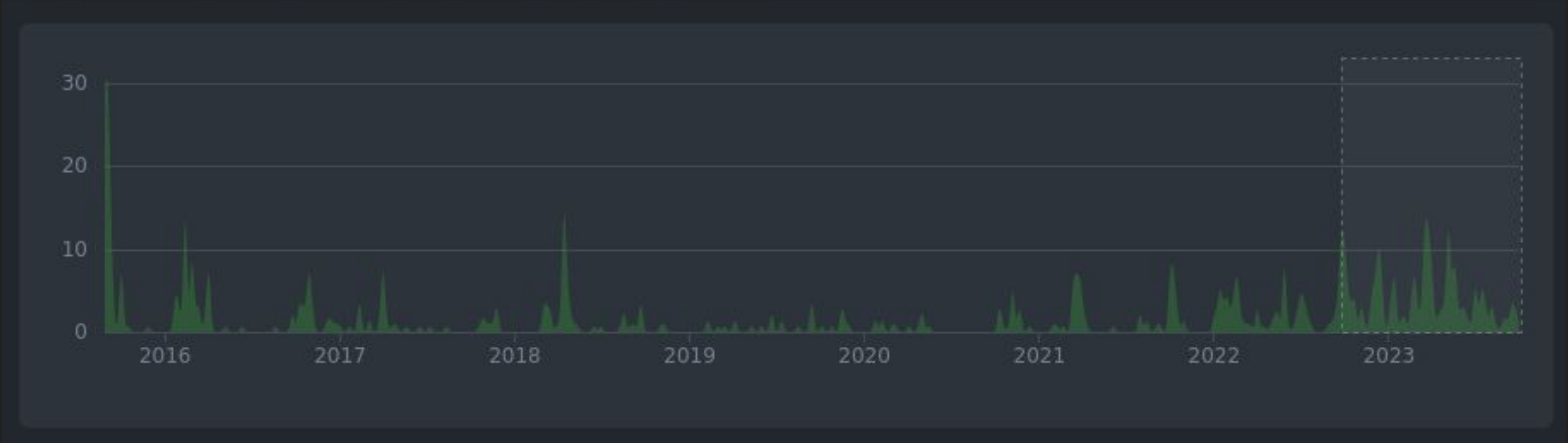
JuliaHub

Agenda

- Repository statistics
- Changes to documentation
- Features and ongoing development since last year
- Release 1.0
- Contributing

Repository statistics (`Ferrite-FEM/Ferrite.jl`)

- 7 releases (0.3.8 - 0.3.14)
- 218 commits (750 in total)
- 20 contributors (11 new, 30 in total)



https://github.com/Ferrite-FEM/Ferrite.jl/graphs/contributors?from=2022-09-26&to=2023-10-06&type=c

Documentation overhaul

- Docs: https://ferrite-fem.github.io/ (changes only visible in dev)
- New structure following the follows the Diátaxis Framework
  - Tutorials: Thoroughly explained examples aimed at introduce Ferrite.jl concepts
  - Topic guides: More in-depth explanations
  - Reference: API documentation, the documentation strings
  - How-to guides: Shorter(?) guides for specific tasks. Assume knowledge of Ferrite.
- Code gallery: Showing cool things you can do with Ferrite. Contribute!
- Developer documentation: documenting internal code

Features

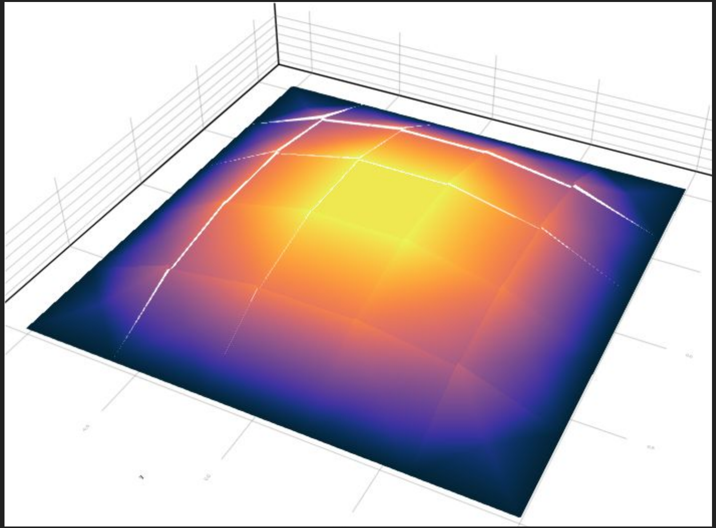
Features I will *not* talk about
(because others will)

Discontinuous Galerkin methods (Abdulaziz Hamid)

- Google Summer of Code (GSoC) 2023 project
- Sparsity pattern couplings between elements
- `InterfaceIterator` for iterating over all internal interfaces in the grid
- `InterfaceValues` for integration of internal interfaces between elements

More details in the presentation by Abdulaziz

this afternoon.



`DofHandler` rework (Kim Louisa Auth)

- Grids with mixed element types
- `SubDofHandler` for working with grid subdomains with different physics/fields

More details in the presentation by Kim this afternoon.

Mesh refinement and adaptivity (Maximilian Köhler)

- Implementation of `p4est` (tree based mesh data structure)
- Hanging node constraints

More details in the presentation by Maximilian this afternoon.

Distributed computing (Dennis Ogiermann)

- `FerriteDistributed.jl`: Data structures (`Grid`, `DofHandler`) for solving problems on multiple cores using MPI
- Support for `PartitionedArrays.jl` and `HYPRE.jl` for global matrix/vector

More details in the presentation by Dennis this afternoon.

Features I *will* talk about
(because others won't)

Local application of boundary conditions

***Global* application**

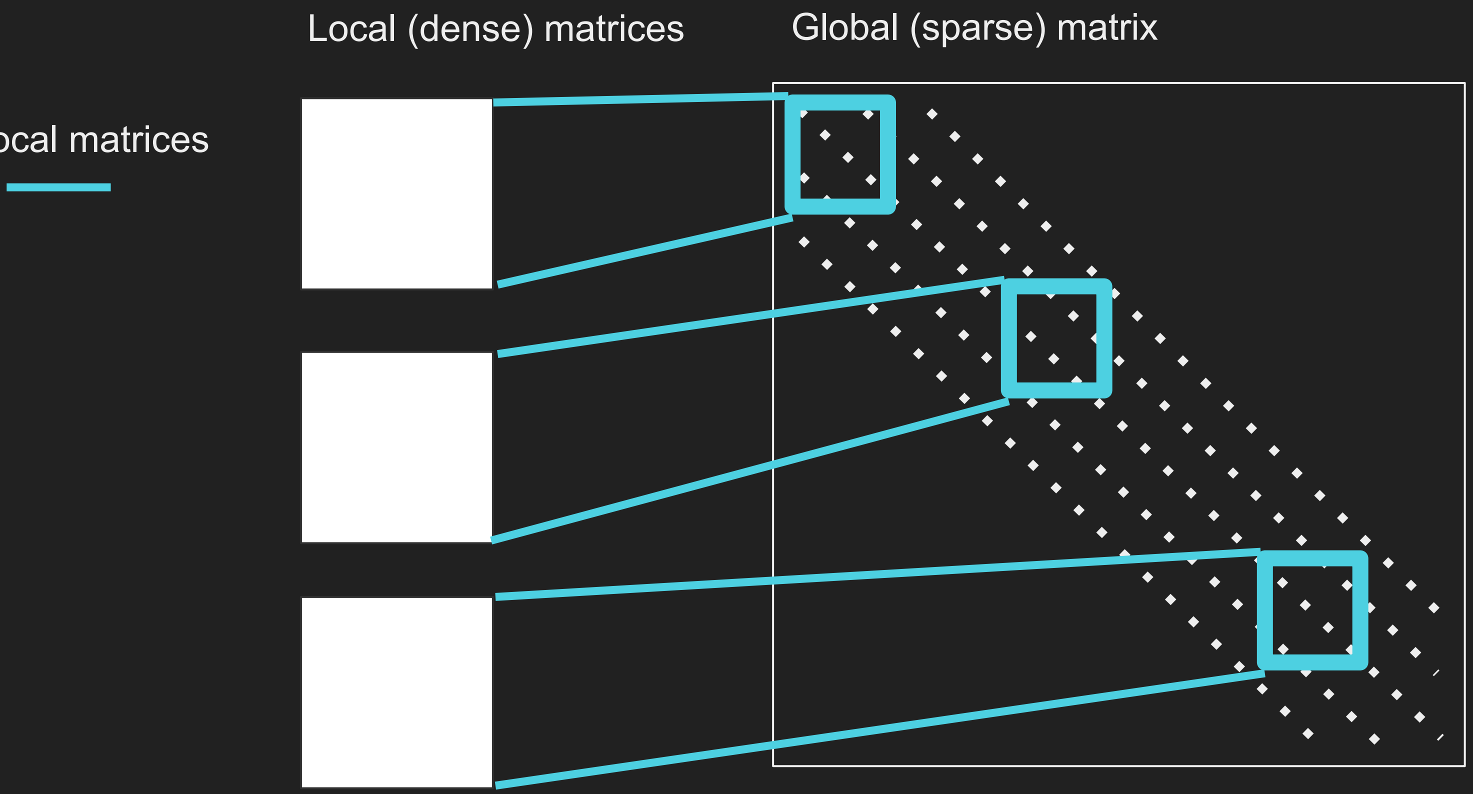
```
for all elements
    1. compute local matrix
    2. assemble into global matrix
end
3. apply boundary conditions
4. solve
```

***Local* application**

```
for all elements
    1. compute local matrix
    2. apply boundary conditions
    3. assemble into global matrix
end
4. solve
```

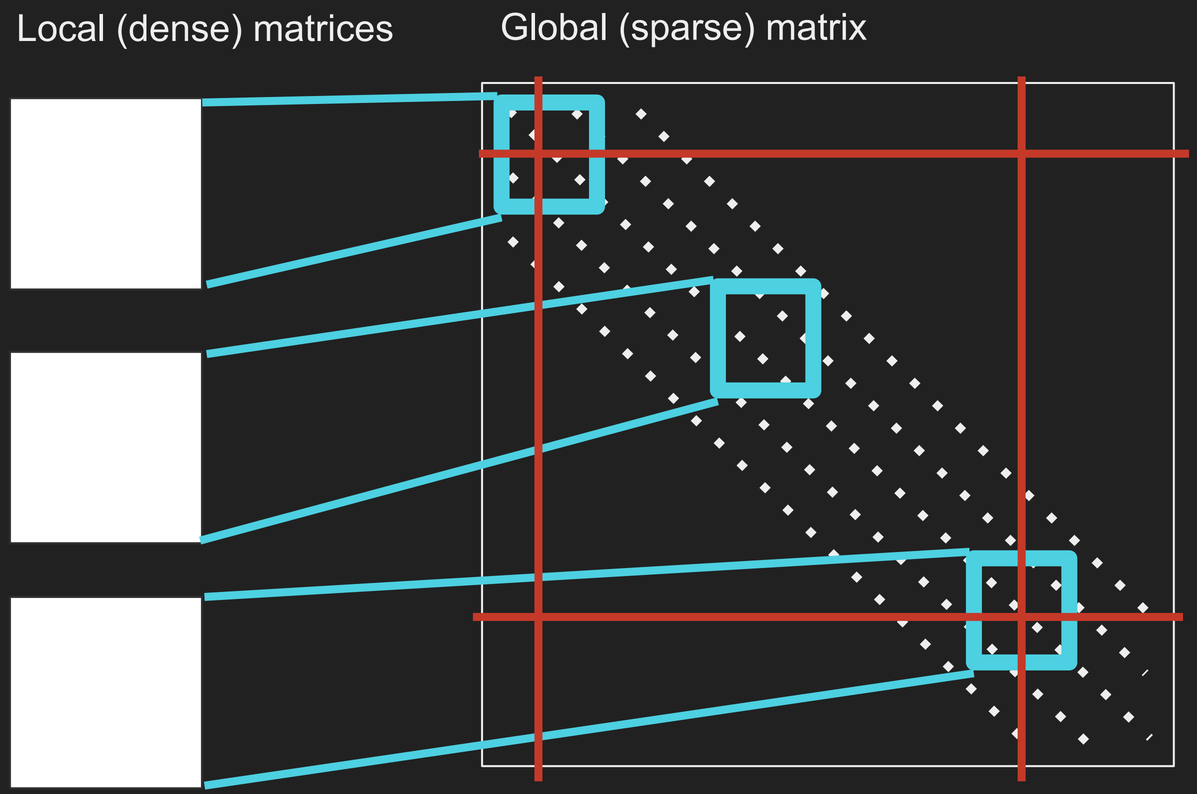
Global application of boundary conditions

1. Compute local matrices
2. Assemble



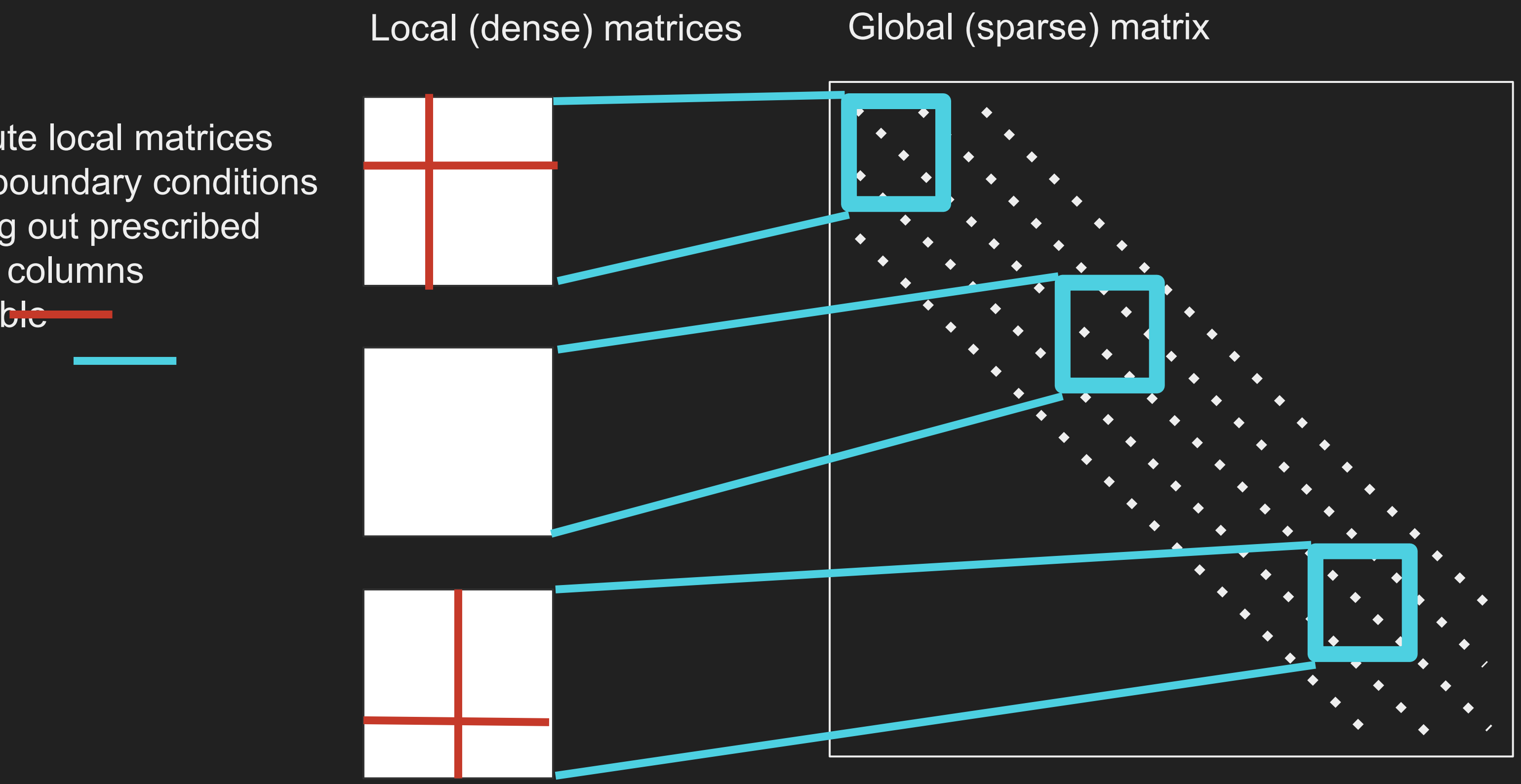
Global application of boundary conditions

1. Compute local matrices
2. Assemble
3. Apply boundary conditions by zeroing out prescribed rows and columns



Local application of boundary conditions

1. Compute local matrices
3. Apply boundary conditions by zeroing out prescribed rows and columns
2. Assemble



Local application of boundary conditions

***Global* application**

```
for cell in cells
    # Compute local matrix
    ke = assemble_element(...)
    # Assemble into global matrix
    assemble!(K, dofs, ke)
end
# Apply boundary conditions
apply!(K, f, ch)
# Solve
u = K \ f
```

***Local* application**

```
for cell in cells
    # Compute local matrix
    ke = assemble_element(...)
    # Apply boundary conditions
    apply_local!(ke, dofs, ch)
    # Assemble into global matrix
    assemble!(K, dofs, ke)
end
# Solve
u = K \ f
```

Renumbering degrees of freedom

- By default dofs enumerated element-by-element and field-by-field
- Renumber by fields/components to obtain global block system (`BlockArrays.jl`)
  - `renumber!(dh, ch, DofOrder.FieldWise())`
  - `renumber!(dh, ch, DofOrder.ComponentWise())`
- Renumber using `Metis.jl` to reduce fill-in
  - `renumber!(dh, ch, DofOrder.Ext{Metis}())`

Renumbering degrees of freedom: `BlockArrays.jl`

Example: Stokes flow (https://ferrite-fem.github.io/Ferrite.jl/dev/tutorials/stokes-flow/)

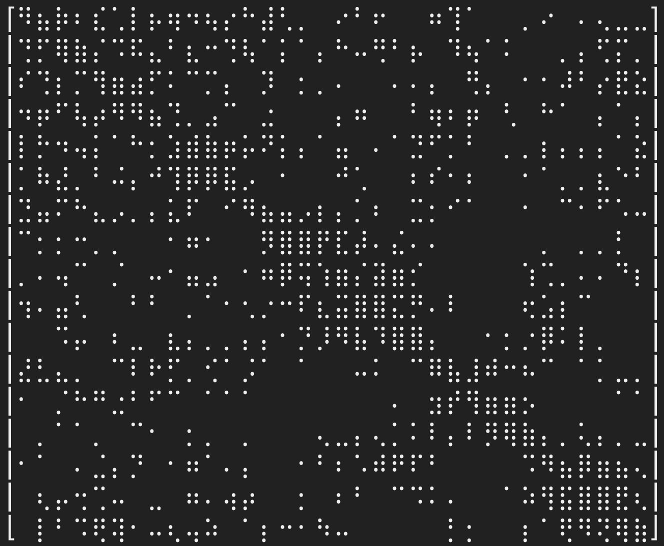
Find $(\boldsymbol{u}, p) \in \mathbb{U} \times \mathrm{L}_2$ s.t.

$$\int_\Omega \Big[[\delta\boldsymbol{u} \otimes \boldsymbol{\nabla}] : [\boldsymbol{u} \otimes \boldsymbol{\nabla}] - (\boldsymbol{\nabla} \cdot \delta\boldsymbol{u})\, p \Big] \mathrm{d}\Omega = \int_\Omega \delta\boldsymbol{u} \cdot \boldsymbol{b} \, \mathrm{d}\Omega \quad \forall \delta\boldsymbol{u} \in \mathbb{U},$$

$$\int_\Omega -(\boldsymbol{\nabla} \cdot \boldsymbol{u})\, \delta p \, \mathrm{d}\Omega = 0 \quad \forall \delta p \in \mathrm{L}_2,$$

Renumbering degrees of freedom: `BlockArrays.jl`

Default order

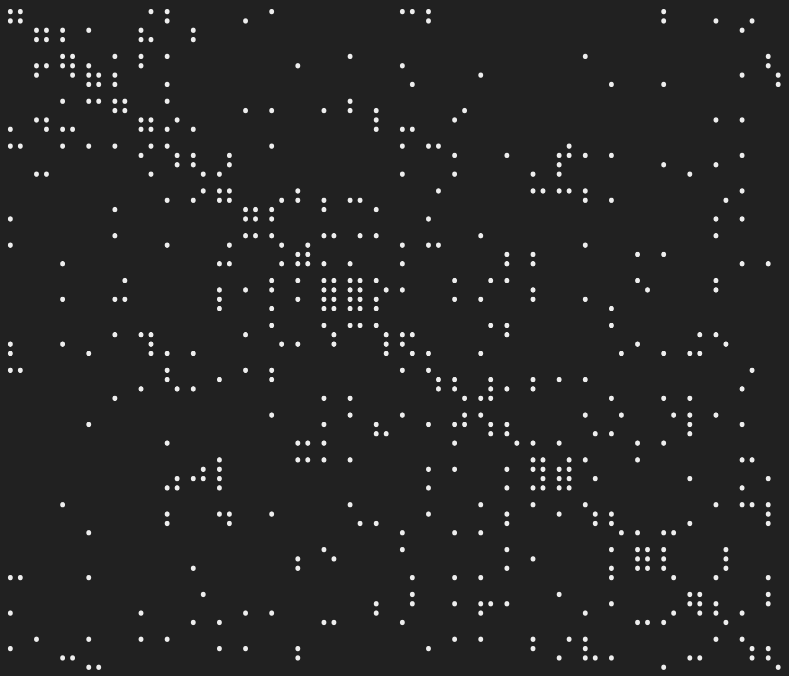


FieldWise order

Renumbering degrees of freedom: `BlockArrays.jl`

Renumbering degrees of freedom: `Metis.jl`

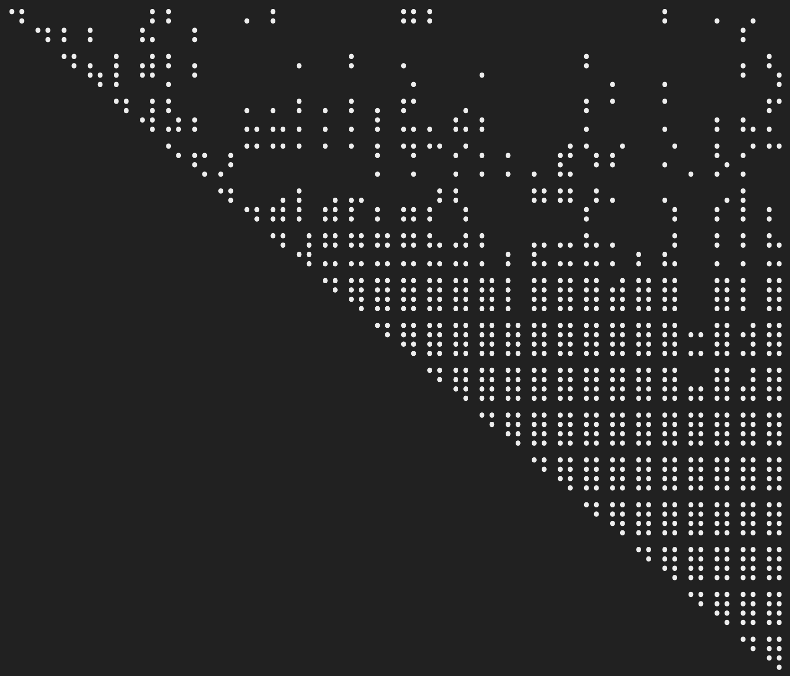
Sparse matrix default order
(5% stored values)



Sparse matrix Metis.jl order
(5% stored values)

Renumbering degrees of freedom: `Metis.jl`

Cholesky factor default order
(16% stored values)



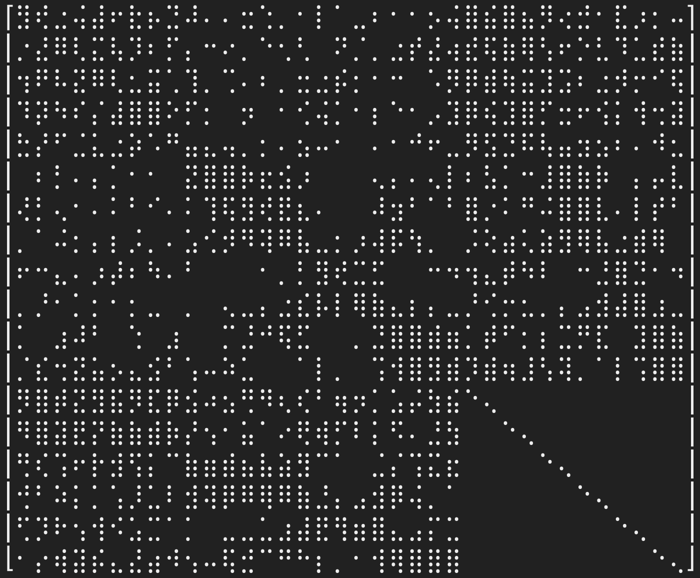
Cholesky factor Metis.jl order
(6% stored values)

Reducing the sparsity pattern: specify field coupling

Example: Stokes flow, no coupling between δp-p

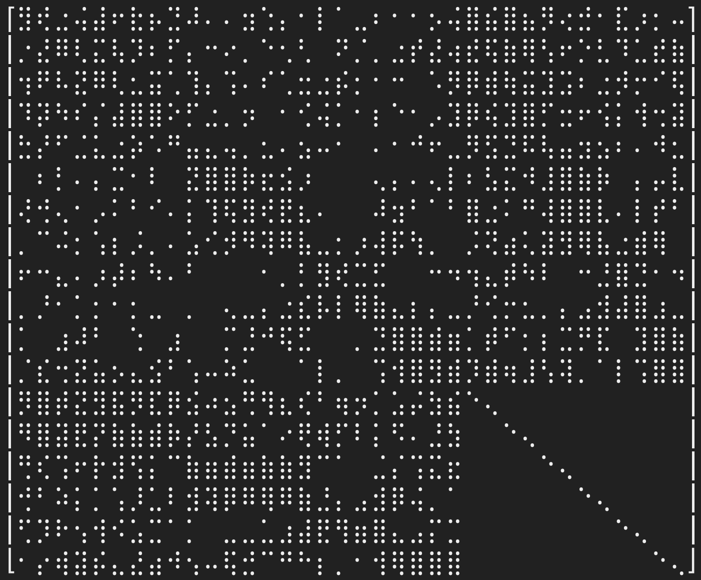
Default order

FieldWise order

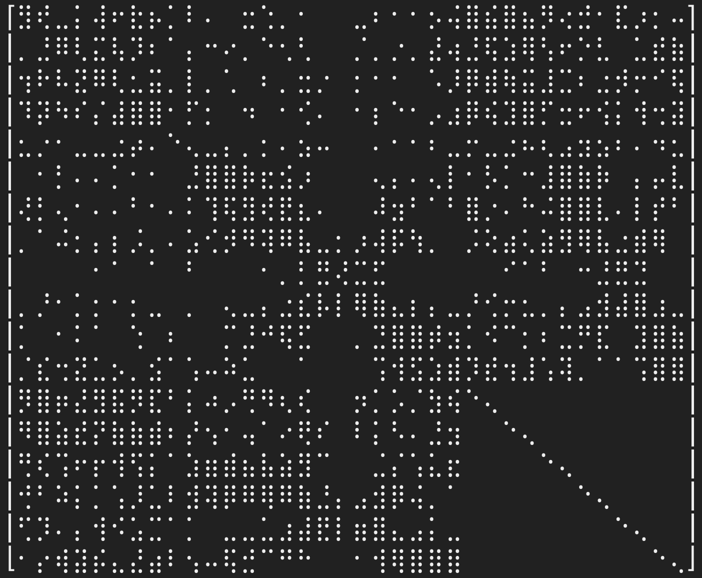


Reducing the sparsity pattern: eliminate constrained dofs

keep_constrained=true (default)
18128 non-zeroes

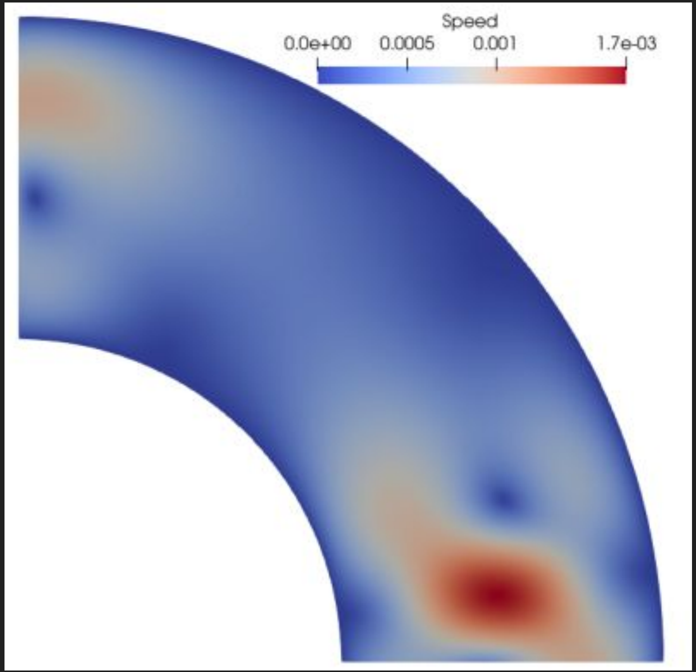


keep_constrained=false
15114 non-zeroes



`ConstraintHandler`, `Dirichlet`, `PeriodicDirichlet`

- All components of the specified field prescribed by default (instead of component 1)
- Possible to specify prescribed values as `f(x)` instead of just `f(x, t)`
- `update!` is called implicitly in `close!(ch)`
- Periodic constraints support rotations (cf. Stoke's flow example)



Boundary integration

- `FaceQuadratureRule` replaces “`dim-1`” `QuadratureRule`
- `FaceIterator` for easier iteration over faces (similar to `CellIterator`) to integrate e.g. Neumann boundaries

Reference shapes changes

Every shape has its own type:

`RefLine`	`RefCube{1}`
`RefQuadrilateral`	`RefCube{2}`
`RefHexahedron`	`RefCube{3}`
`RefTriangle`	`RefTetrahedron{2}`
`RefTetrahedron`	`RefTetrahedron{3}`
`RefPrism`	
`RefPyramid`	

Reference shapes changes

Simplifies construction of interpolations and quadrature rules

`Lagrange{RefLine, 1}()`	`Lagrange{1, RefCube, 1}()`
`Lagrange{RefTriangle, 1}()`	`Lagrange{2, RefTetrahedron, 1}()`
`QuadratureRule{RefQuadrilateral}(...)`	`QuadratureRule{2, RefCube}(...)`
`FaceQuadratureRule{RefHexahedron}(...)`	`QuadratureRule{2, RefCube}(...)`

Interpolations

- Interpolations grouped into `ScalarInterpolation` and `VectorInterpolation`
- `ScalarInterpolations` need to be explicitly vectorized for vector problems using `VectorizedInterpolation`:

```
Lagrange{RefTriangle, 1}()
```

$$N_1^{\mathrm{S}}, \quad N_2^{\mathrm{S}}, \quad N_3^{\mathrm{S}}$$

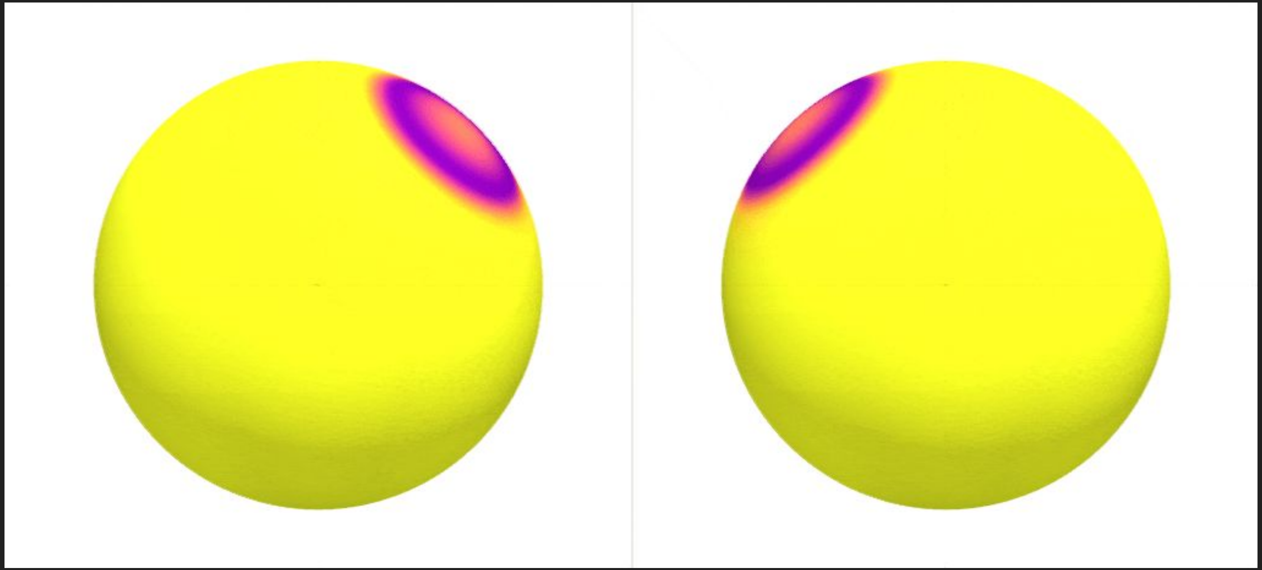
```
VectorizedInterpolation{2}(Lagrange{..}())
alt. Lagrange{RefTriangle, 1}()^2
```

$$\begin{pmatrix} N_1^{\mathrm{S}} \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ N_1^{\mathrm{S}} \end{pmatrix}, \begin{pmatrix} N_2^{\mathrm{S}} \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ N_2^{\mathrm{S}} \end{pmatrix}, \begin{pmatrix} N_3^{\mathrm{S}} \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ N_1^{\mathrm{S}} \end{pmatrix}$$

- `Enabled merging of CellScalarValues and CellVectorValues into CellValues`
- `Enables` “true” vector interpolation such as Nédélec and Raviart-Thomas

Embedding

- Clearer separation of i) reference dimension, ii) spatial dimension, and iii) vector dimension
- Enables embedded elements (e.g. 2D elements in 3D space)



More things!

All features, fixes, improvements, etc, are (should be) documented in the changelog: CHANGELOG.md

Release 1.0

- Next release will be a breaking 1.0 release
- Many “mechanical” changes (e.g. new reference shapes)
- How-to upgrade section in the CHANGELOG.md

You can contribute!

Contributor guide: CONTRIBUTING.md

Ferrite.jl contributor guide

Welcome to Ferrite.jl contributor documentation! In this document you find information about:

- Documentation
- Reporting issues
- Code changes

If you are new to open source development in general there are many guides online to help you get started, for example first-contributions. Another great resource, which specifically discusses Julia contributions, is the video Open source, Julia packages, git, and GitHub.

Documentation

Contributing to documentation is a great way to get started with any new project. As a new user you have a unique

Thanks for listening!