Transfering OpenFOAM data to Python / Pytorch

Course material developed at SCC
Scientific Centre for Computing
Karlsruhe Institute of Technology

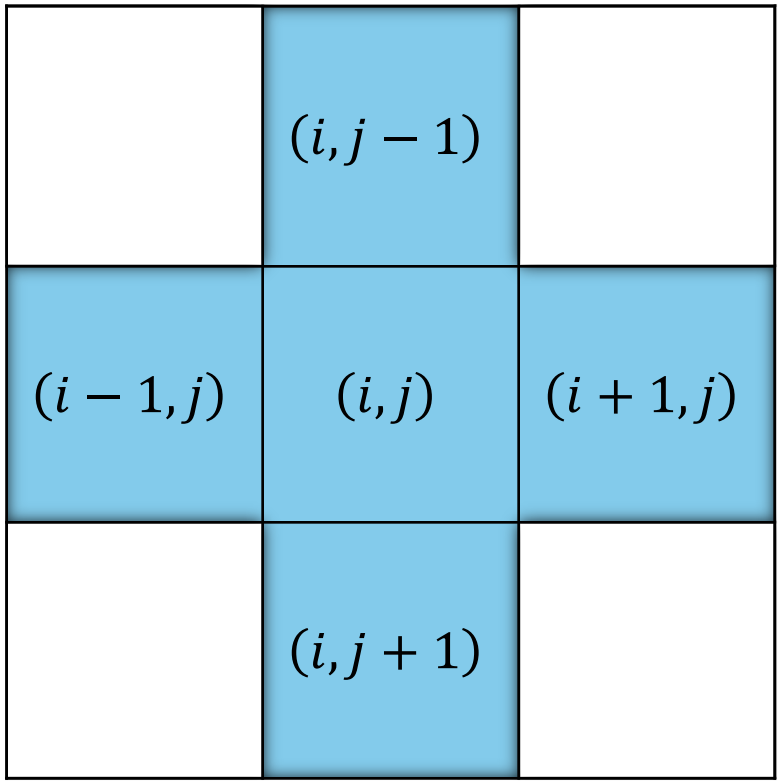
Structured grids in Finite Difference Method (not OpenFOAM)

- Cells form a regular logical lattice and are addressed by tuples (i, j) in 2D, or (i, j, k) in 3D.
- The neighbors of cell (i, j) are accessible simply by

$$(i \pm 1, j) \text{ and } (i, j \pm 1).$$

- Connectivity is implicit and nothing needs to be stored to find a neighbor.
- Cells are numbered

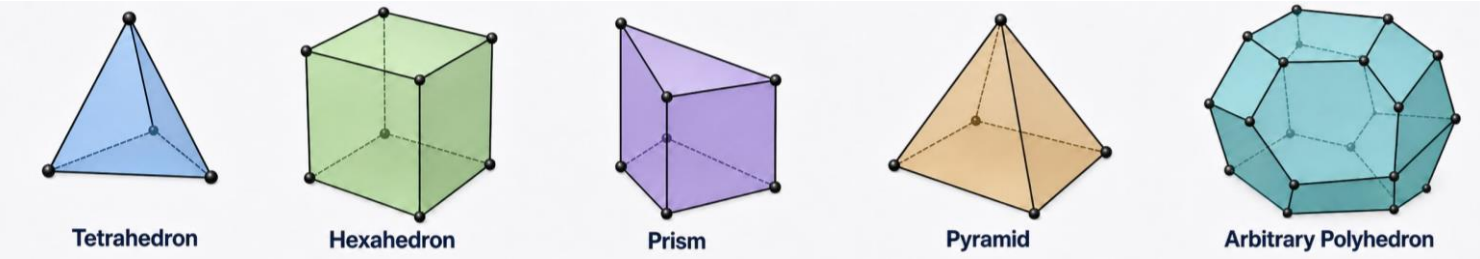
$(0,0), (1,0), (2,0) \ldots$
$(0,1), (1,1), (2,1) \ldots$
$\ldots$



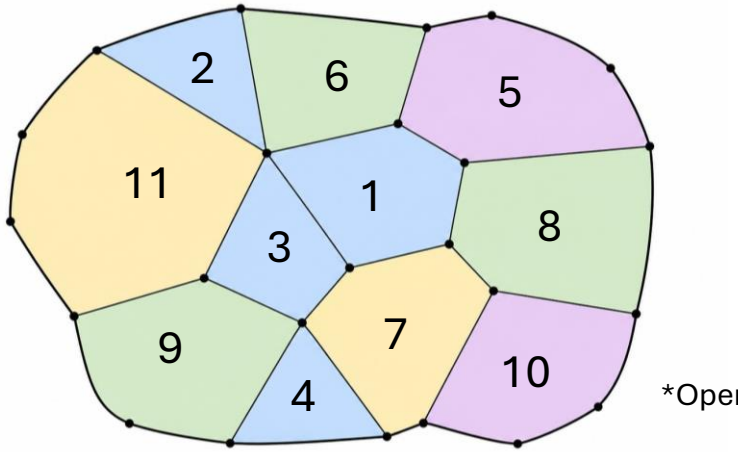
Unstructured grids in Finite Volume Method (OpenFOAM)

OpenFOAM is built around a fully unstructured, polyhedral mesh:

- ❑ Cells can be tetrahedra, hexahedra, prisms, pyramids, or arbitrary polyhedra.



- ❑ Different cell types may coexist in the same mesh.
- ❑ Cells are numbered 1, 2, 3, … in **no particular spatial order**.



*OpenFOAM stores fields cell-by-cell using the internal cell index: **field[cell]**

Developing a utility in OpenFOAM (template code)

- **Goal**: write our first OpenFOAM utility that do nothing (template).
- **Workflow**:

1. Copy the existing utility:

cp -r $FOAM_APP/utilities/postProcessing/postProcess basicCalc

1. Rename the application to *basicCalc.C*.
2. Update compilation files: *Make/files* and *Make/options*
3. Remove unnecessary postProcess functionality and keep only the basic OpenFOAM structure

Now we have a basic utility that loop over all written timesteps and do nothing, we can use it as a template in this course.

- **Exercise**: Compare

$FOAM_APP/utilities/postProcessing/postProcess

With

extraFiles/basicCalc

```
int main(int argc, char *argv[])
{


    timeSelector::addOptions();

    #include "setRootCase.H"
    #include "createTime.H"
    instantList timeDirs = timeSelector::select0(runTime, args);
    #include "createNamedMesh.H"

    forAll(timeDirs, timei)
    {
        runTime.setTime(timeDirs[timei], timei);
        Info<< "Time = " << runTime.timeName() << endl;
        // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
        // * * * * * * * * * -write your code here * * * * * * * * * //
        // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    }

    Info<< "End\n" << endl;

    return 0;
}
```

Cell indices in OpenFOAM

- **Goal**: write our first OpenFOAM utility that exports cell IDs as a field.
- **Workflow**:

1. Copy the template and rename it to *writeCellIndices.C*:
2. Update compilation files: *Make/files* and *Make/options*
3. replace write your code here with code that writes cell indices.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
volScalarField indices
(
    IOobject
    (
        "indices",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedScalar("zero", dimensionSet(0,0,0,0,0,0,0), 0.0)
);

forAll(indices, cellI)
{
    indices[cellI] = cellI;
}
indices.write();
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

* OpenFOAM does not have integerField

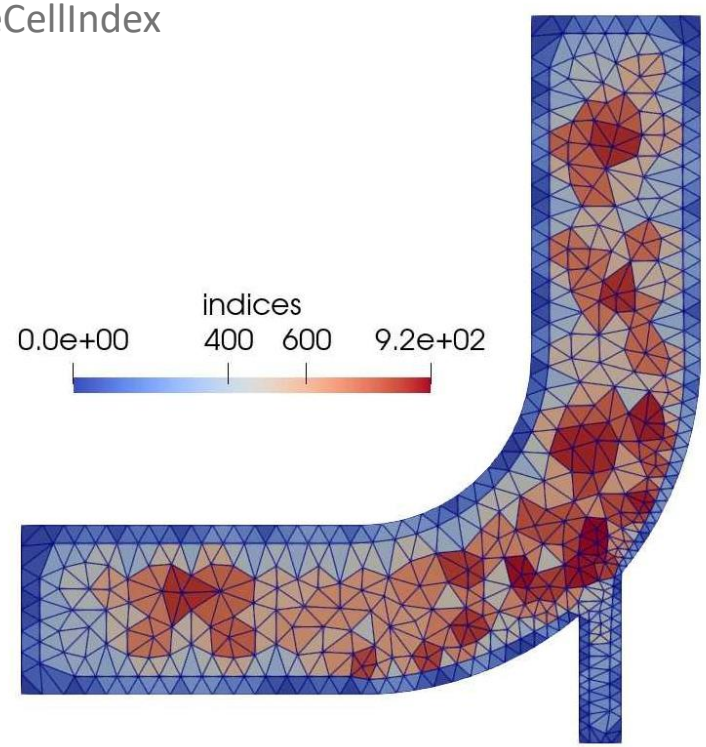
```
int main(int argc, char *argv[])
{


    timeSelector::addOptions();

    #include "setRootCase.H"
    #include "createTime.H"
    instantList timeDirs = timeSelector::select0(runTime, args);
    #include "createNamedMesh.H"

    forAll(timeDirs, timei)
    {
        runTime.setTime(timeDirs[timei], timei);
        Info<< "Time = " << runTime.timeName() << endl;
        // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
        // * * * * * * * * * -write your code here * * * * * * * * * * * * * * * * //
        // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    }

    Info<< "End\n" << endl;

    return 0;
}
```

4. Compile the utility with **wmake**

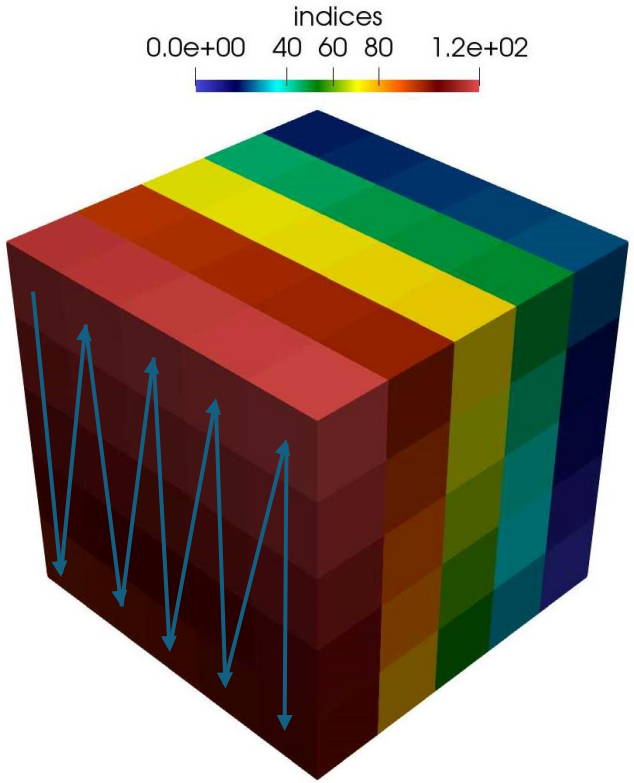
Cell indices in OpenFOAM

- Test the new utility

cd $FOAM_TUTORIALS/incompressible/icoFoam/elbow
./Allrun
writeCellIndex

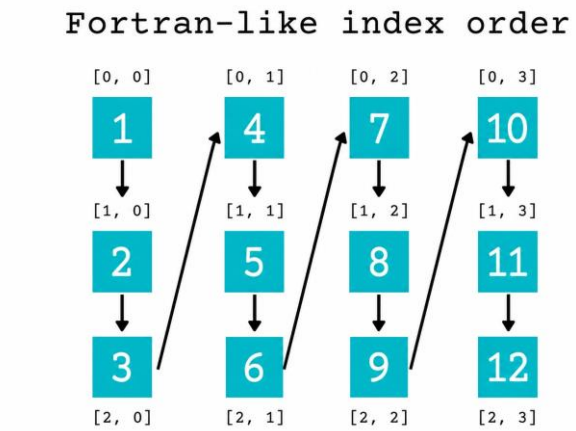


cd $FOAM_TUTORIALS/DNS/dnsFoam/boxTurb16
./Allrun
writeCellIndex



- If the mesh is generated from a single structured block using *blockMesh*, OpenFOAM stores the cells in a regular, Fortran-like order:

i changes first, then j, then k



*Use **mapFields** function object to interpolate the results on a blockMesh grid before exporting to python

Exporting blockMesh OpenFOAM data to Python & Pytorch

- **Goal**: making a utility for exporting OpenFOAM data to Python.

See "*FoamToNumpy*" in the supplementary materials

```
Info<< "Time = " << runTime.timeName() << endl;

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    ),
    mesh
);

int idx = 0;

forAll(U, i)
{
    U_flat[idx++] = U[i].x();
    U_flat[idx++] = U[i].y();
    U_flat[idx++] = U[i].z();
}

std::string filename = "U_flat_" + runTime.timeName() + ".bin";
FILE *fp = fopen(filename.c_str(), "wb");
fwrite(U_flat, sizeof(float), 3*n, fp);
fclose(fp);
```

Template utility

```
int main(int argc, char *argv[])
{


    timeSelector::addOptions();

    #include "setRootCase.H"
    #include "createTime.H"
    instantList timeDirs = timeSelector::select0(runTime, args);
    #include "createNamedMesh.H"

    forAll(timeDirs, timei)
    {
        runTime.setTime(timeDirs[timei], timei);
        Info<< "Time = " << runTime.timeName() << endl;
        // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
        // * * * * * * * * * -write your code here * * * * * * * * * //
        // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    }

    Info<< "End\n" << endl;

    return 0;
}
```

In Python:

```
velocity_field = np.fromfile("./U_flat_9.bin", dtype=np.float32)
velocity_field = velocity_field.reshape((3, 16, 16, 16), order="F") # structured numpy array
velocity_field = torch.from_numpy(velocity_field) # If you need to create a Pytorch Tensor for ML training
```

Mapping unstructured data to a structured grid

Problem: the *FoamToNumpy* utility from the previous slide assumes a Fortran-like ordering, which only exists for single-block *blockMesh* cases. For a general unstructured mesh (e.g. the elbow case), cell indices are in no particular spatial order, so we cannot reshape the flat array into a structured (i, j, k) tensor.

Solution - two steps:

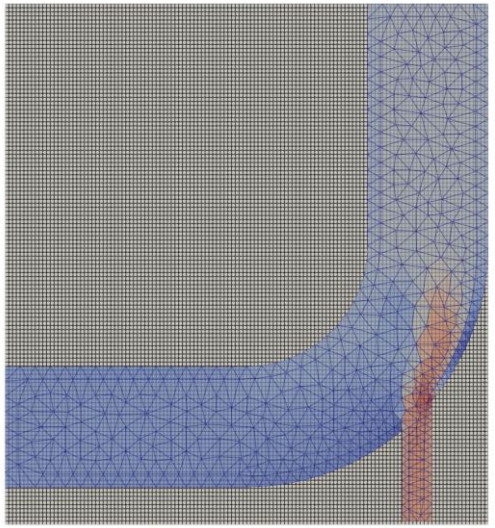
- ❑ Step 1: build a structured background grid that fully covers the original computational domain.
- ❑ Step 2: interpolate (map) the unstructured field onto this structured grid.

Workflow of Step 1:

1. Create a new case *elbowStructured* next to the original elbow case.
2. Define a structured mesh in *system/blockMeshDict*. The block must be large enough to enclose the entire original domain (a bounding box of the elbow geometry).
3. Generate the mesh by *blockMesh*.

- ❑ The structured grid will contain cells inside the original domain (which will receive interpolated values) and cells outside (which need to be masked later).

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}

scale   1;

vertices
(
    (0 -5 -0.5)
    (64 -5 -0.5)
    ...

);

blocks
(
    hex (0 1 2 3 4 5 6 7) (128 128 1) simpleGrading (1 1 1)
);

boundary
(
    patch0_half0
    {
        type empty;
        faces
        (
            (0 3 2 1)
        );
    }
    ...
);
```



Mapping unstructured data to a structured grid

Workflow of Step 2 (interpolation**):**

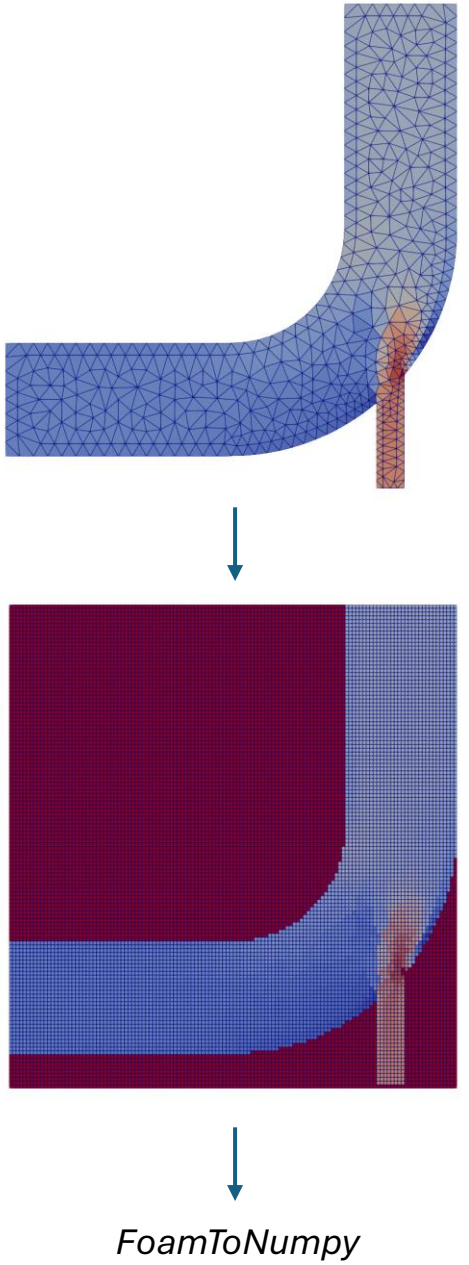
1. Initialize the target field with a large sentinel value, e.g. in 0/U: *internalField uniform (1e10 1e10 1e10);* After mapping, only the cells that overlap the source domain will be overwritten. Cells outside the original geometry keep this large value and can be masked out later in Python / PyTorch (e.g. *mask = (U < 1e9)*).
2. Write *system/mapFieldsDict*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      mapFieldsDict;
}

patchMap        ( );

cuttingPatches  ( );
```

3. Run the mapping from inside the structured case: *mapFields ../elbow -sourceTime 7 -mapMethod mapNearest* Available interpolation methods: mapNearest, interpolate, cellPointInterpolate

❑ Once the data lives on the structured grid, run our previous utility: "*FoamToNumpy*", and reshape the binary in Python exactly as before: *velocity_field.reshape((3, Nx, Ny, Nz), order="F")*

✓ *Now the data is ready for CNNs, FNOs, or any ML pipeline that expects a structured tensor input.*



FoamToNumpy