

An Introduction to *OpenFOAM*

Stefan Radl

Graz University of Technology

(Revision 1.2, February 2019)

Session C – Understand and Modify OpenFOAM

Overview

- 1) We will do some basic C/C++ concept: create a type definition (“typedef”)
- 2) We will build an application: the „meshInspector“
- 3) We will build a library that can be called as a function object: the „scalarTranspAdvanced“ functionObject

Session C: Modification of OpenFOAM

Start with:
Basic C++ (Hakan Nilssons Slides,
‘C_Programming_Nilsson’, slides 1-8)

IMPORTANT:
C/C++ CODE NEEDS TO BE COMPILED! ALWAYS CHECK IF YOU CODE
HAS BEEN COMPILED WITHOUT ERROR AND YOU ARE USING THE
NEW EXECTUABLE AND/OR LIBRARY

Task 1: Create a New *typedef*

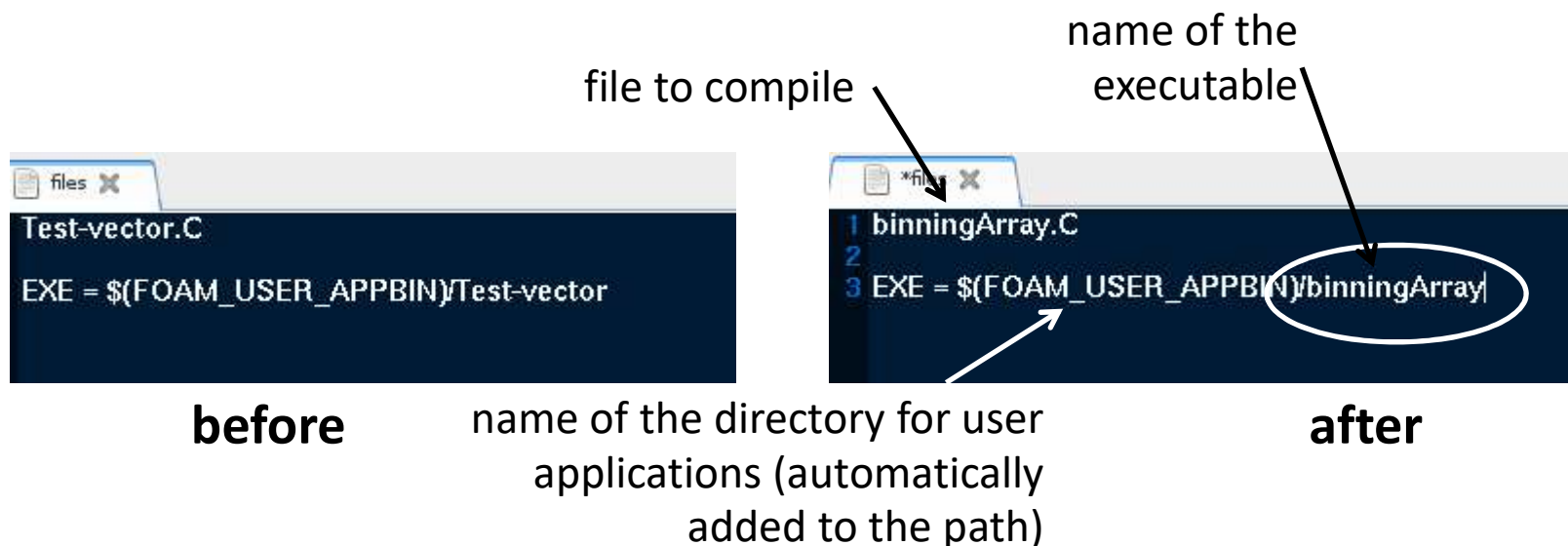
“ ~ ” means that this is your home directory!

- 1) Go to `~/OpenFOAM/<user>-5.x`
- 2) Create the directory `~/OpenFOAM/<user>-5.x/apps/`
- 3) Copy the directory `“vector”` from `“$WM_PROJECT_DIR/applications/test/”` into your `“test”` directory.
- 4) Change into your `„apps/vector“` directory.
- 5) You can now check if you are able to compile the application (do `„wclean“`, and then `„wcmake“` in your terminal; this will first clean, and then build the application). If everything works well, you can run the program by typing `„Test-vector“` in your terminal.
- 6) Copy `“~/OpenFOAM/<user>-5.x/apps/vector”`, and create the new directory `“~/OpenFOAM/<user>-5.x/apps/binningArray”`
- 7) Use a terminal to change into `„~/OpenFOAM/<user>-5.x/apps/binningArray”`, and start with the exercise!

Task 1: Create a New *typedef*

How compiling an application works

- Be sure you are in „~/OpenFOAM/<user>-5.x/apps/binningArray”
- First, we will clean the application directory (do „wclean”)
- Next, change the filename of „Test-vector.C” into „**binningArray.C**”
- We now need to tell the wmake system, which file to compile, and what is the name of the application (i.e., the name of the executable). Therefore, we need to open the file „files” in the sub-dir „Make”:

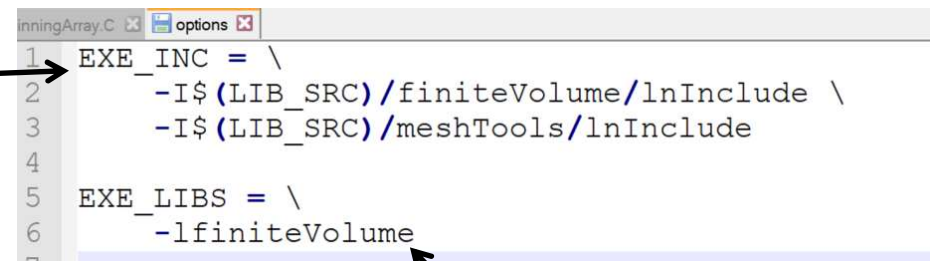


Task 1: Create a New *typedef*

How compiling an application works

- For the new application, we also need to link an existing library (i.e., a precompiled set of routines) to our program. We can specify which libraries to add in the „**options**“ file in the „Make“ sub-directory.
- We need only the library „finiteVolume“ (we will later add some more functionality to the applications...). This library can be added by placing the following code into the „options“ file:

we need to include the path to the **headers (i.e., the declarations) used in the library**, such that wmake can find the header files. Do not forget to precede „-I“ (upper case I as in Igor)



```
1 EXE_INC = \  
2 -I$(LIB_SRC)/finiteVolume/lnInclude \  
3 -I$(LIB_SRC)/meshTools/lnInclude  
4  
5 EXE_LIBS = \  
6 -lfiniteVolume  
7
```

the name of the library (or libraries) **to be linked** to the application. Do not forget to precede „-l“ (lower case l as in Lambda)

Task 1: Create a New *typedef*

How compiling an application works

- We are now in the position to compile the application. First, we clean out already compiled data using „wclean“ (be sure to be in the “~/OpenFOAM/<user>-5.x/test/binningArray” directory)
- Then, we simply type „wmake“. Your application will be compiled!
- Respect the text in the terminal if something is not working.
- If the application has been already build, the message
“make: `/home/sradl/OpenFOAM/sradl-
5.x/platforms/linux64GccDPOpt/bin/binningArray'
is up to date.”
will appear. This is ok!

Task 1: Create a New *typedef*

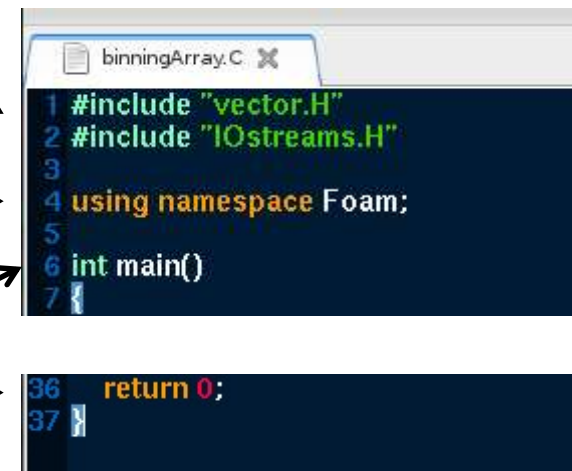
Inspecting an Application

- We should now open the file with the C-code in it. This is „binningArray.C“ in our case.
- We will still see the „old“ code from the „vector“ test example. The general structure of the code is always the same, and explained below.

includes that contain the declarations of variables and functions used in the code

we will just use functions from the „Foam“ project. So, we use the namespace „Foam“, i.e., we use all functions/classes from this group of functions/classes

we now define the „main“, i.e., the application that will be run. We choose that it will be an integer-valued function, and we will return „0“ after completion



```
1 #include "vector.H"
2 #include "IOstreams.H"
3
4 using namespace Foam;
5
6 int main()
7
36 return 0;
37
```


Task 1: Create a New *typedef*

Inspecting an Application

The „Info“ object is used to print data and messages to the screen. „endl“ changes the output to the next line. „<<“ just passes data to the left. „::“ means that e.g., zero is a variable in the „vector“ group of variables

Here we defined an object called „d“, which is of class „vector“. We initialize it with (0.5,0.5,0.5) by calling the appropriate constructor. We then normalize *d* by division with its magnitude.

We do some more vector operations, and print the results to the screen. „magSqr“ returns the squared magnitude of a vector. A full list of functions (and its use) can be found at https://cpp.openfoam.org/v5/namespacemembers_func.html

Hint: navigate to 'src', and use 'grep -r <function to search for> ./' to find the definition!

“d /=mag(d)” is the same as “d=d/mag(d)”!!

```
6 int main()
7 {
8     Info<< vector::zero << endl
9         << vector::one << endl
10        << vector::dim << endl
11        << vector::rank << endl;
12
13    vector d(0.5, 0.5, 0.5);
14    d /= mag(d);
15
16    vector dSmall = (1e-100)*d;
17    dSmall /= mag(dSmall);
18
19    Info<< (dSmall - d) << endl;
20
21    d *= 4.0;
22
23    Info<< d << endl;
24
25    Info<< d + d << endl;
26
27    Info<< magSqr(d) << endl;
28
29    vector d2(0.5, 0.51, -0.5);
30    Info<< cmptMax(d2) << " "
31        << cmptSum(d2) << " "
32        << cmptProduct(d2) << " "
33        << cmptMag(d2)
34        << endl;
35    Info<< min(d, d2) << endl;
36    return 0;
37 }
```

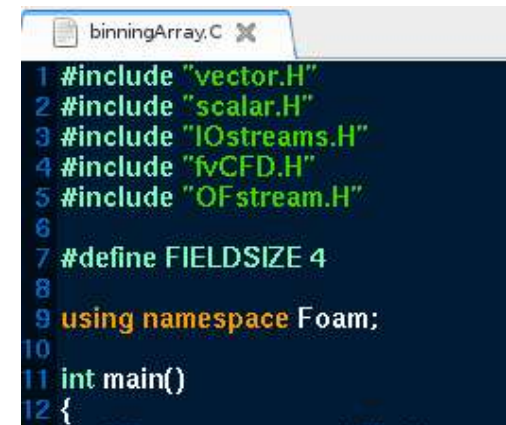
Task 1: Create a New *typedef*

Creating a New Application

- We now want to create a new application, in order to introduce some important features of OpenFOAM.
- Specifically, we will show how to work with a „Field“, as well as how to save data to disk.
- For this purpose, we simply remove/add code in the file „binningArray.C“, and finally will compile it.

We first include more headers, in order to access more OpenFOAM features. Specifically, this is “scalar.H” (operations using scalars), “fvCFD.H” (handling lists and all CFD data), as well as “OFstream.H”

We also define here a global variable “FIELD_SIZE”, that is often used in the code (the compiler simply replaced “FIELD_SIZE” with the value “4” during the compilation process).



```
1 #include "vector.H"
2 #include "scalar.H"
3 #include "IOstreams.H"
4 #include "fvCFD.H"
5 #include "OFstream.H"
6
7 #define FIELD_SIZE 4
8
9 using namespace Foam;
10
11 int main()
12 {
```

Task 1: Create a New *typedef*

Creating a New Application

We now define two new data types:

- (a) the type “binningField”, which is a Field consisting of scalars (we use the Template “Field<>” for it, and hand over the “scalar” type).
- (b) The type „binningVField“, which is a Field consisting of vectors

We now use the new types to create fields that hold a scalar, as well as vectors (we call them “particleBins” and “velBins” respectively). The constructor requires us to specify the size of the field as the first argument, and the initial value)

```
int main()
{
    //Create new typedefinition
    typedef Field<scalar> binningField;
    typedef Field<vector> binningVField;

    binningField particleBins(FIELDSIZE, 0.0);
    binningVField velBins(FIELDSIZE, vector::zero);
}
```

We use the “zero” variable in the (sub) namespace “vector” to initialize the vectors in the field.

Task 1: Create a New *typedef*

Creating a New Application

We now would like to fill in data into our fields.
Therefore, we loop over all entries, and put
some (arbitrary) values into it.

Specifically, we add the value of the
iterator to each slot in the field
(accessed by the „[]“)

This is a typical “loop” in C/C++: we first
specify the iterator (called it, it is of data
type “uint”, i.e., an unsigned integer), the
initial value (i.e., 0), the limiting
conditions (i.e., it<FIELDSIZE), and the
increment (i.e., it+)

```
//iterate over bins and add the index  
for(uint it=0; it<FIELDSIZE; it++)  
{  
    particleBins[it] += it;  
    velBins[it] += vector(it, it, it);  
}
```

To fill in the vector, we have to first
define a vector (from the components,
for which we simply pick the iterator),
and then we can add this value to the
field

Task 1: Create a New *typedef*

Creating a New Application

We print the fill list of data in the object „particleBins“ to the screen, as well as some other properties (e.g., „sum()“ returns the sum of all elements in the list).

We now create a **pointer to an unnamed object** of class „Ofstream“. This object will be connected to the file „binLog.dat“ on the disk of the operating system. We do the same for another object that points to „binVLog.dat“)

```
//Report to screen and save to disk
Info << "particleBins: " << particleBins << endl;
Info << "particleBins.size():" << particleBins.size() << endl;
Info << "sum(particleBins): " << sum(particleBins) << endl;
Info << "min(particleBins): " << min(particleBins) << endl;
Info << "max(particleBins): " << max(particleBins) << endl;
Info << "norm. particleBins: " << particleBins/sum(particleBins) << endl;
Ofstream* sPtrBinLog = new Ofstream("binLog.dat");
*sPtrBinLog << particleBins << endl;
Ofstream* sPtrBinVLog = new Ofstream("binVLog.dat");
*sPtrBinVLog << velBins << endl;
```

We now **de-reference the pointer**, in order to pass data to the (unnamed) object. This object will finally write the data to the disk.

Task 1: Create a New *typedef*

Creating a New Application

- We finally save the new source-code file „binningArray.C“, and compile using „wmake“
- Now, the application should be available for execution (just type „binningArray“) in the terminal.
- If you unsure which executable is called by binningArray, type „which binningArray“, which will show you the full path to the file you execute.

Task 1: Create a New *typedef*

Check your Learning Outcomes

At the end of this task, you should be able to:

- Copy and modify test cases from the OpenFOAM installation directory (i.e., the \$WM_PROJECT_DIR/applications/test directory),
- Compile OpenFOAM applications, and change the name of the executable,
- Link new libraries to an OpenFOAM application (by adjusting the Make/options file),
- Perform basic vector calculations using OpenFOAM,
- Work with „Fields“ in OpenFOAM,
- Write data to the disk.

A comment on self learning

If you feel NOT comfortable with C/C++ coding, it suggested to look up specific functions/syntaxes using google. A good page with loads of background info on C/C++ is <http://www.cplusplus.com>.

Task 2: Inspection of a Simple Mesh

- 1) Start from `/OpenFOAM-5.x/applications/test/mesh`
- 2) Copy, and create a new test application 'meshInspector', and rename the files, as well as adjust 'Make/files'
- 3) Code!

Task 3: library 'scalarTranspAdvanced'

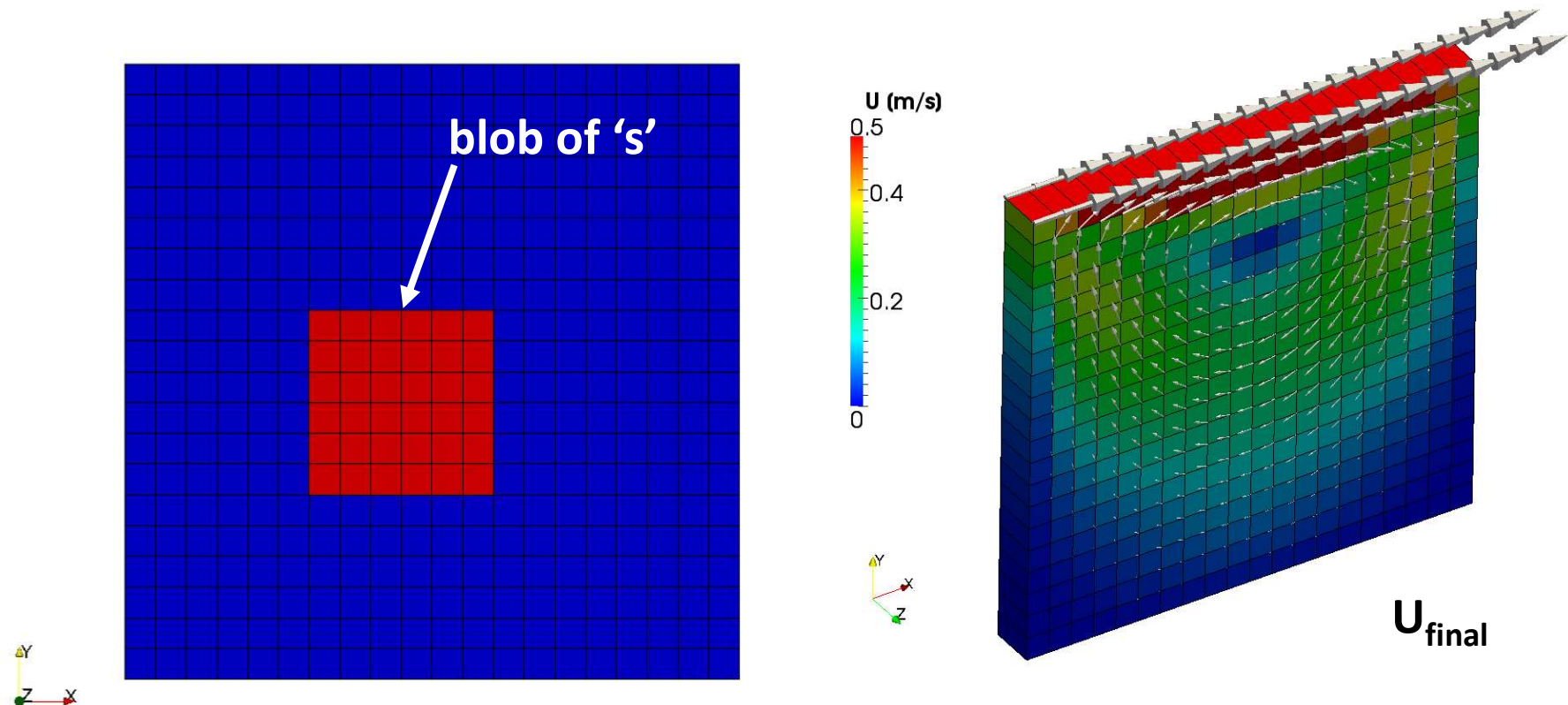
- 1) Start from /opt/OpenFOAM-5.x/src/functionObjects/solvers
- 2) Copy, and create a new library in the folder 'funcSolversIPPT'. Copy the 'scalarTransport' folder to 'scalarTranspAdvanced'. Rename the files in 'scalarTranspAdvanced', and modify content in 'scalarTranspAdvanced'
- 3) Adjust 'Make/files' to compile both 'scalarTransport' and 'scalarTranspAdvanced'
- 4) Code!

Cases

- 1) pitzDaily
- 2) cavity case

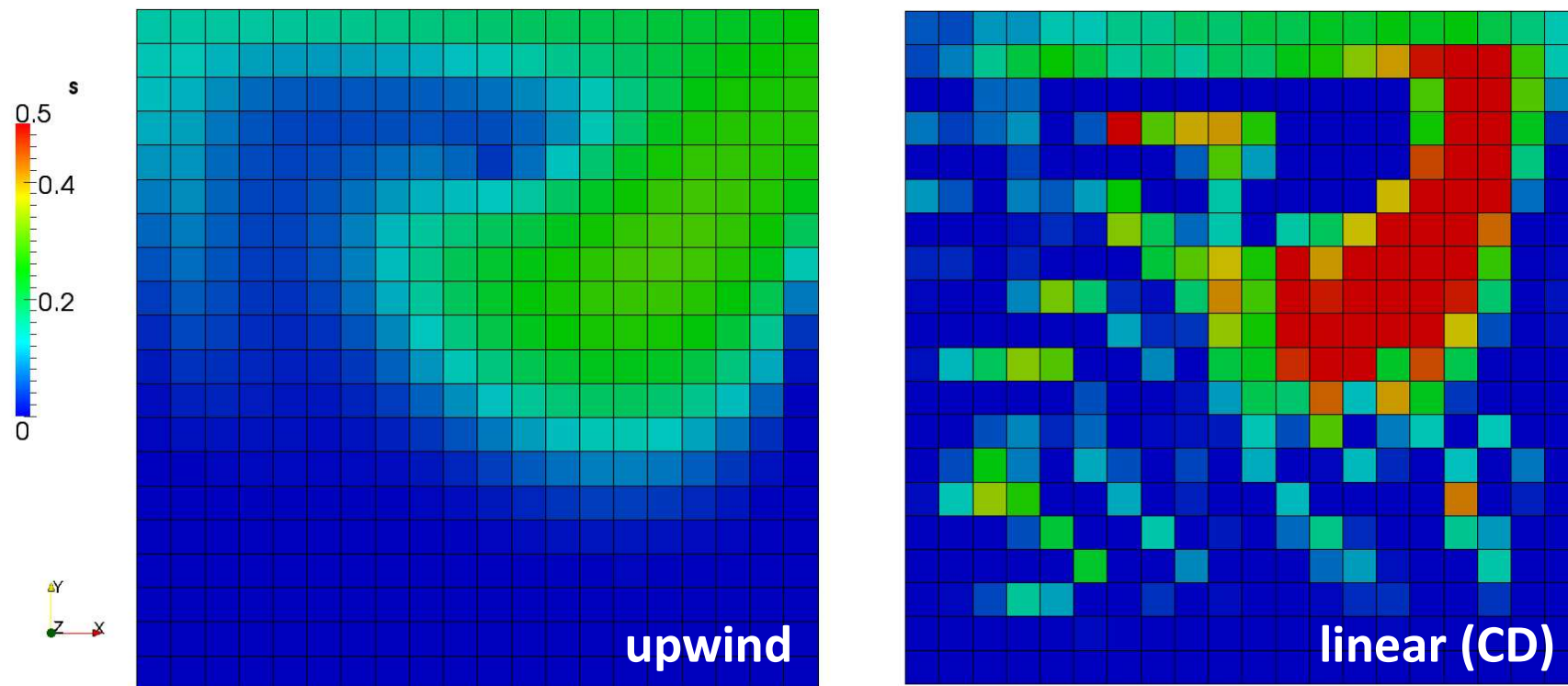
Task 3: library 'scalarTranspAdvanced'

Setup: A Simple Cavity (2D, 20x20 grid cells)



Task 3: library 'scalarTranspAdvanced'

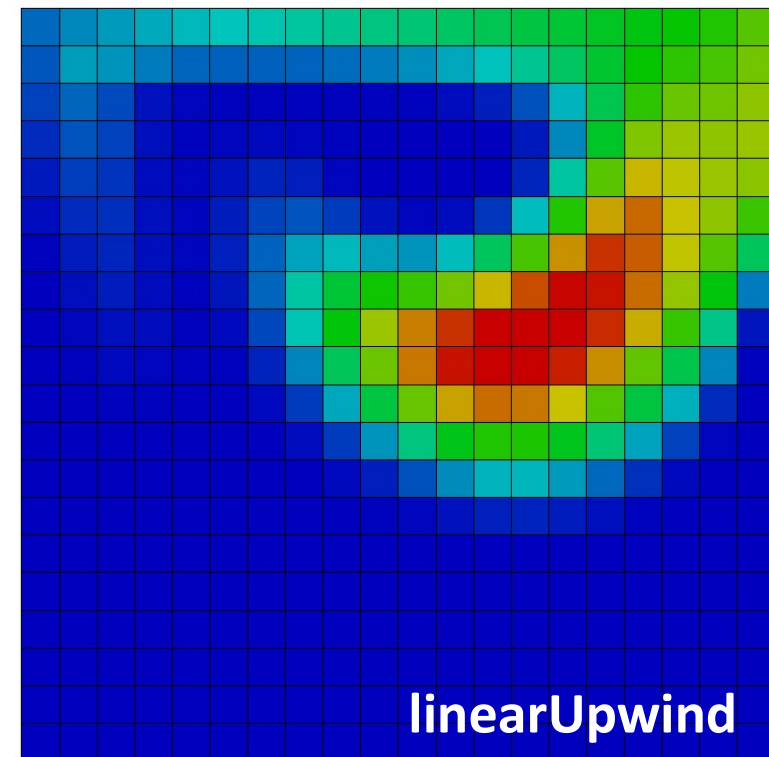
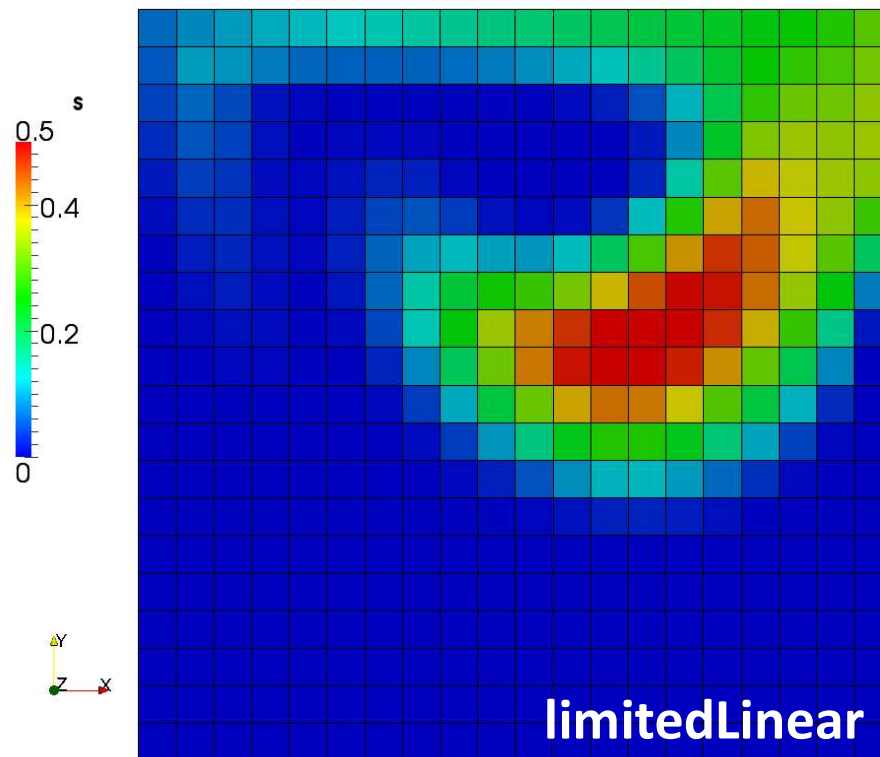
Results: Influence of the Convection Scheme



→ use 'upwind' for quick shots, **NEVER** for production runs.

Task 3: library 'scalarTranspAdvanced'

Results: Influence of the Convection Scheme



Task 3: library 'scalarTranspAdvanced'

4) More on Discretization Schemes For the Convection Term

A Selection of Convection Schemes

linear	Second order, unbounded
skewLinear	Second order, (more) unbounded, skewness correction
cubicCorrected	Fourth order, unbounded
upwind	First order, bounded
linearUpwind	First/second order, bounded
QUICK	First/second order, bounded
limitedLinear	Second order, bounded
limitedLinear01	Second order, strictly bounded between 0 and 1

Further details are in the source code, or here:

<https://www.openfoam.com/documentation/guides/latest/doc/guide-schemes-divergence.html>

Programming Tipps

- <http://www.slideeee.com/slide/openfoam-programming-tips>

Source Code and Source Code Documentation

- <http://www.openfoam.org/docs/cpp/>
- <https://www.openfoam.com/documentation/guides/latest/api/index.html>
- <https://github.com/OpenFOAM/OpenFOAM-5.x> (browse the code)

For those that would like to become pro...

<https://openfoam.org/dev/coding-style-guide/>

Copyright & Disclaimer

© Stefan Radl, TU Graz, 2019.

The material (slides, files, etc.) and the screencast are published under a CC BY-SA license (<https://creativecommons.org/licenses>)

OPENFOAM® (also OpenFOAM®) is a registered trade mark of OpenCFD Limited, producer and distributor of the OpenFOAM software.

This offering is not approved or endorsed by OpenCFD Limited, producer and distributor of the OpenFOAM software via www.openfoam.com, and owner of the OPENFOAM® and OpenCFD® trade marks.

Please note the regulations shown here:
<https://www.openfoam.com/legal/trademark-policy.php>