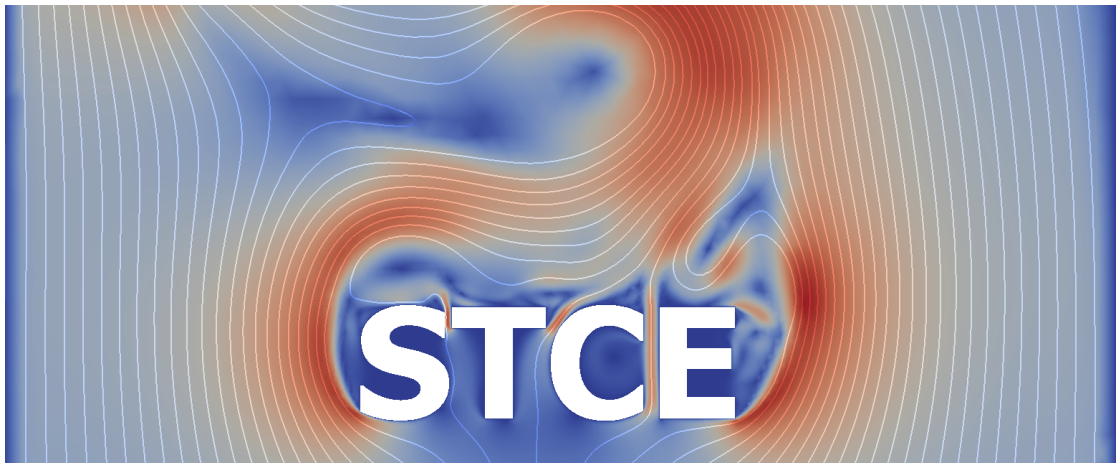


Discrete Adjoint OpenFOAM

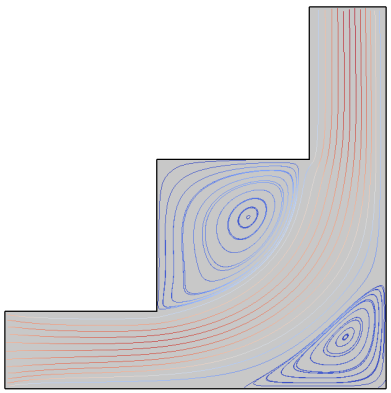
Markus Towara
Software and Tools for Computational Engineering
RWTH Aachen University

OpenFOAM

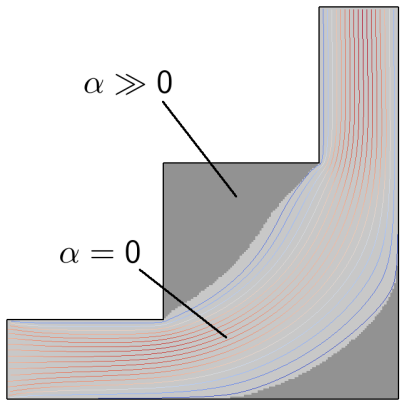
- **Open** **F**ield **O**peration and **M**anipulation
- Open-Source (GPLv3) CFD solver
- developed by OpenCFD Ltd., currently at version 3.0.x
- includes tools for meshing, pre-, post-processing
- rising adoption in industry and academia due to lack of licence costs $\rightarrow$ well suited for parallel architectures

Topology Optimization

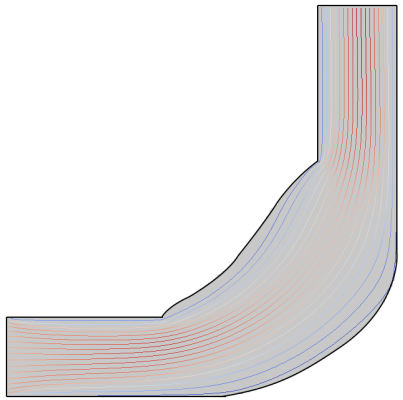
no optimization



added "material"



reconstruction



Add penalty term α to Navier-Stokes equation[1] :

$$(\mathbf{v} \cdot \nabla)\,\mathbf{v} = \nu \nabla^2 \mathbf{v} - \nabla p - \alpha \mathbf{v}$$

[1] C. Othmer: *A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows.* Intern. J. f. Num. Meth. in Fluids. p. 861–877, 2008.

How to find appropriate α

- Define cost function J, e.g. total pressure loss between inlet and outlet:

$$J = \int_{\Gamma} p + \frac{1}{2} v_n^2 \, \mathrm{d}\Gamma$$

- Calculate sensitivity of the cost function w.r.t. parameters α_i

$$\frac{\partial J}{\partial \alpha_i} = ???$$

- Calculate updated porosity field α^{n+1}, e.g.:

$$\alpha_i^{n+1} = \alpha_i^n - \lambda \cdot \frac{\partial J^n}{\partial \alpha_i^n}, \quad \text{while insuring} \quad 0 < \alpha_i < \alpha_{max}$$

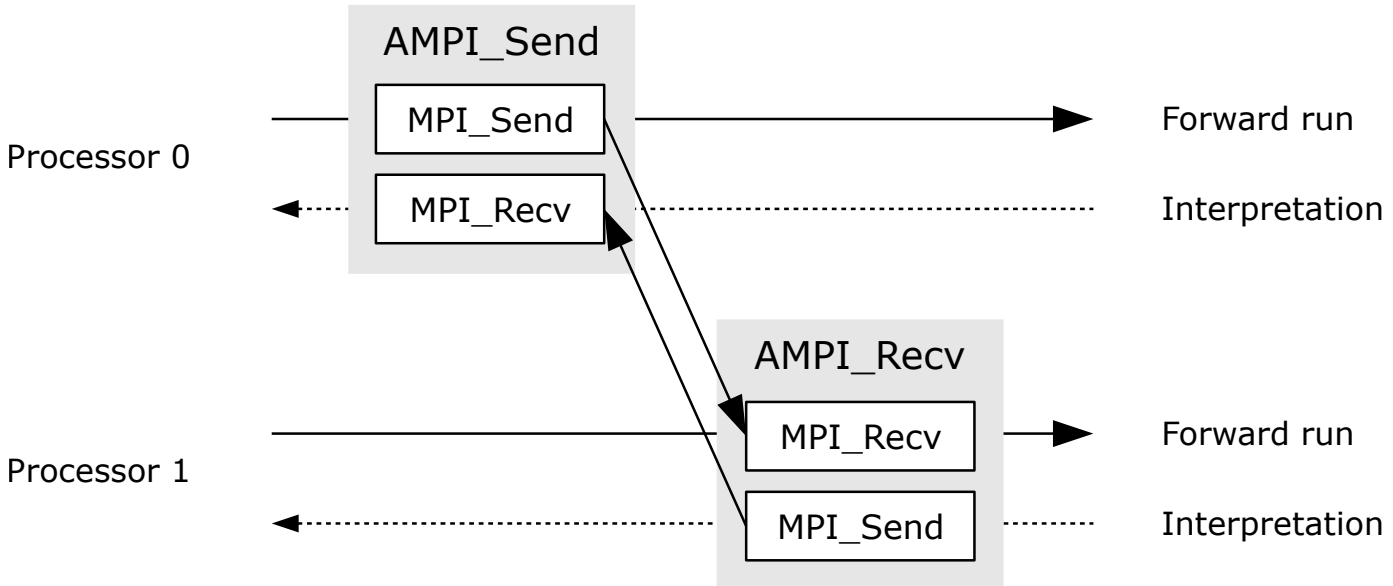
- Loop until α converged...

AD with Operator Overloading

- Replace **all** floating point values with custom AD datatype.
- Operators (e.g. +,-,*,/,sin,sqrt,...) are overloaded to calculate sensitivities in addition to the primal values.
- For adjoint mode the computation is split into forward execution (computation of the primals, storing operations and intermediate values in tape) and interpretation (propagation of the adjoints trough the tape from outputs to inputs).

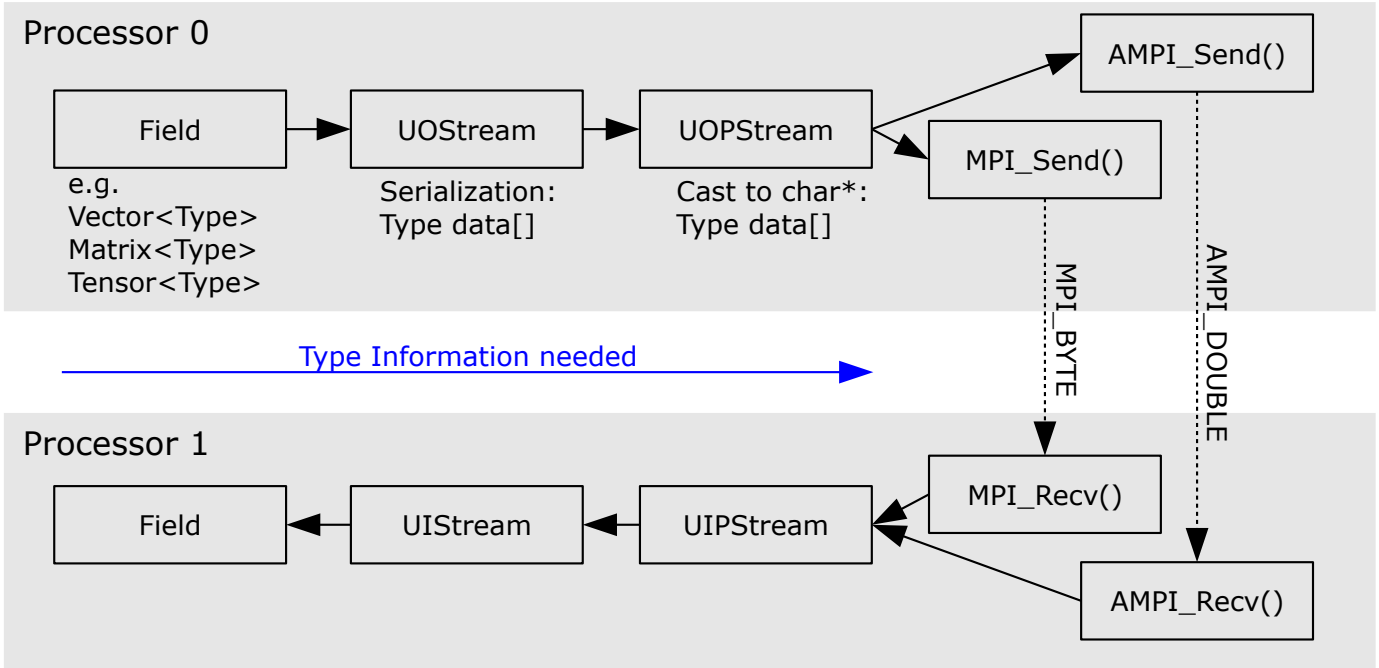
AMPI

- Adjoining programs which use MPI is not straightforward
- Library AMPI developed by STCE in cooperation with Inria FR and Argonne US
- see github.com/michel2323/AdjointMPI



AMPI in OpenFOAM

- OpenFOAM casts all Messages to char* before sending them
- This approach is not feasible for adjoints



Results Distributed Machine

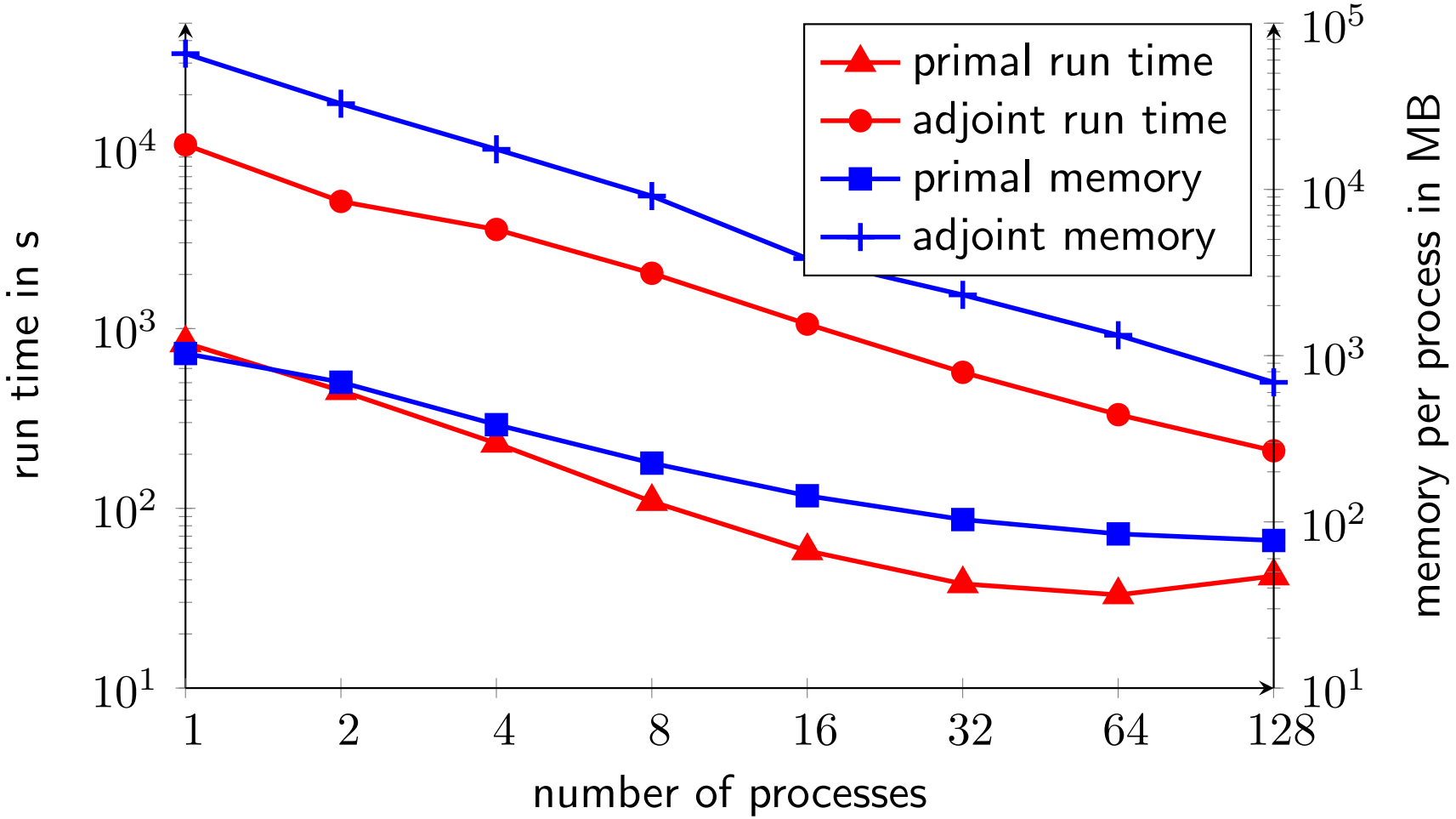


Figure: Run time and memory scaling on RWTH Compute Cluster

OpenFOAM a1s mode

in `src/OpenFOAM/primitives/Scalar/doubleScalar/doubleScalar.h`: replace:

```
namespace Foam
{

    typedef double doubleScalar;
    ...

}
```

with:

```
#include "dco.hpp"
namespace Foam
{

    typedef dco::ga1s<double>::type doubleScalar;
    ...

}
```

OpenFOAM t1s mode

in `src/OpenFOAM/primitives/Scalar/doubleScalar/doubleScalar.h`: replace:

```
namespace Foam
{

    typedef double doubleScalar;
    ...

}
```

with:

```
#include "dco.hpp"
namespace Foam
{

    typedef dco::gt1s<double>::type doubleScalar;
    ...

}
```

Example Adjoint Solver, start with simpleFoam

```
1  int main(int argc, char *argv[])
2  {
3      #include "createFields.H"
4
5      simpleControl simple(mesh);
6      // run until end time reached / converged
7      while (simple.loop())
8      {
9          // Pressure-velocity SIMPLE corrector
10         #include "UEqn.H"
11         #include "pEqn.H"
12
13         turbulence->correct();
14         runTime.write();
15     }
16     return 0;
17 }
```

Calculation of the Residual in OpenFOAM

Momentum Equation:

$$\nabla \cdot (\phi, \mathbf{U}) - \nabla \cdot (\nu \nabla \mathbf{U}) + \alpha \mathbf{U} = -\nabla \mathbf{p}$$

with mass flux through faces $\phi = \rho A \mathbf{U} \cdot \mathbf{n}$

`UEqn.H:`

```
fvVectorMatrix UEqn
(
    fvm::div(phi, U)
  - fvm::laplacian(nu, U)
  + fvm::Sp(alpha, U)
  ==
    fvOptions(U)
);
fvOptions.constrain(UEqn);
UEqn.relax();
fvVectorMatrix UEqnFull(UEqn == -fvc::grad(p));
```

Example Adjoint Solver, start with simpleFoam

```
1  int main(int argc, char *argv[]){
2      dco::ga1s<double>::global_tape = dco::ga1s<double>::tape_t::create();
3      dco::ga1s<double>::global_tape->register_variable(alpha[i],n);
4
5      while (simple.loop()){
6          #include "UEqn.H"
7          #include "pEqn.H"
8          turbulence->correct();
9      }
10
11     scalar J = 0;
12     forAll(costFunctionPatches(),patchI)
13         J += calcCost(patchI);
14
15     dco::derivative(J) = 1.0;
16     dco::ga1s<double>::global_tape->interpret_adjoint();
17
18     forAll(alpha,i) // get adjoint sensitivitys, scale with cell volume, write to sens
19         sens[i] = dco::derivative(alpha[i])/mesh.V()[i];
20 }
```

Example Adjoint Solver, start with simpleFoam

```
1  int main(int argc, char *argv[]){
2      dco::ga1s<double>::global_tape = dco::ga1s<double>::tape_t::create();
3      dco::ga1s<double>::global_tape->register_variable(alpha[i],n);
4
5      while (simple.loop()){
6          #include "UEqn.H"
7          #include "pEqn.H"
8          turbulence->correct();
9      }
10
11     scalar J = 0;
12     forAll(costFunctionPatches(),patchI)
13         J += calcCost(patchI);
14
15     dco::derivative(J) = 1.0;
16     dco::ga1s<double>::global_tape->interpret_adjoint();
17
18     forAll(alpha,i) // get adjoint sensitivitys, scale with cell volume, write to sens
19         sens[i] = dco::derivative(alpha[i])/mesh.V()[i];
20 }
```

Example Adjoint Solver, start with simpleFoam

```
1  int main(int argc, char *argv[]){
2      dco::ga1s<double>::global_tape = dco::ga1s<double>::tape_t::create();
3      dco::ga1s<double>::global_tape->register_variable(alpha[i],n);
4
5      while (simple.loop()){
6          #include "UEqn.H"
7          #include "pEqn.H"
8          turbulence->correct();
9      }
10
11     scalar J = 0;
12     forAll(costFunctionPatches(),patchI)
13         J += calcCost(patchI);
14
15     dco::derivative(J) = 1.0;
16     dco::ga1s<double>::global_tape->interpret_adjoint();
17
18     forAll(alpha,i) // get adjoint sensitivitys, scale with cell volume, write to sens
19         sens[i] = dco::derivative(alpha[i])/mesh.V()[i];
20 }
```

Example Adjoint Solver, start with simpleFoam

```
1  int main(int argc, char *argv[]){
2      dco::ga1s<double>::global_tape = dco::ga1s<double>::tape_t::create();
3      dco::ga1s<double>::global_tape->register_variable(alpha[i],n);
4
5      while (simple.loop()){
6          #include "UEqn.H"
7          #include "pEqn.H"
8          turbulence->correct();
9      }
10
11     scalar J = 0;
12     forAll(costFunctionPatches(),patchI)
13         J += calcCost(patchI);
14
15     dco::derivative(J) = 1.0;
16     dco::ga1s<double>::global_tape->interpret_adjoint();
17
18     forAll(alpha,i) // get adjoint sensitivitys, scale with cell volume, write to sens
19         sens[i] = dco::derivative(alpha[i])/mesh.V()[i];
20 }
```

Thank you!
Questions?