# Toward a Discrete Adjoint Model for OpenFOAM

Markus Towara, Uwe Naumann

{towara,naumann}@stce.rwth-aachen.de

LuFG Informatik 12: Software and Tools for Computational Engineering

March 28, 2012

# Vita

| | |
|---|---|
| 2006 - 2011 | Computational Engineering Science @ RWTH-Aachen |
| 2010 - 2011 | Internship / Diploma thesis @ Volkswagen AG - Kassel |
| 2012 | PhD Student @ STCE, RWTH-Aachen |

# Acknowledgment


- Niloofar Sarafin for adding dco support to OpenFOAM
- Carsten Othmer for the introduction to adjointSimpleFoam

# Topics

Motivation for AD in Context of Shape Optimization

black-box AD / white-box AD

application to simpleFoam

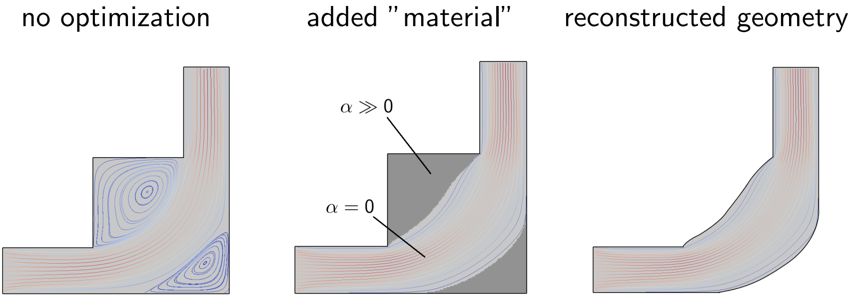
treatment of linear solver

# Motivation for AD in Context of Shape Optimization

- AD promises:
  - greater flexibility w.r.t new objectives
  - easy adaption to new solver types / generations
  - calculated derivatives are exact w.r.t the used discretization
  - higher order derivatives available
- Problems:
  - memory requirements
  - how to retain parallelism?

# Black-box AD


- black-box approach can deliver sensitivities without looking at the inside of the code
- pro: very versatile and fast turnaround time
- contra: for iterative solvers: huge memory requirements
- take a closer look at the code to identify potential simplifications




Added penalty term[1] $\alpha$ :

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

[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.

# Discrete OpenFOAM

- put `dco` into `src/OpenFOAM`
- include `dco.hpp`
- replace `doubles` with active datatype from `dco`
- OpenFOAM has own typedef for scalar floating point values
  $\rightarrow$ just one substitution
- in theory we now just need to recompile OpenFOAM and are ready to go

# 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::a1s::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::t1s::type doubleScalar;
    ...

}
```

# in Reality



- some minor changes have to be made in the OpenFOAM code:
  - unions don't support active datatypes
  - no cast from `dco::type` to `int` available, use `value_v(d)` instead
  - some functions (pow,max,min) don't use the `doubleScalar` typedef and need to be adjusted

# usage of discrete OpenFOAM - t1s

Black-Box tangent-linear Version of `simpleFoam`, calculates $\frac{\partial J}{\partial \alpha_i}$:

```
double sens = 0;
dco::t1s::set(alpha[i],1,1);

for (runTime++; !runTime.end(); runTime++)
{
      ... // solve for U,p
}
// Sum pressure over inlet faces scaled with face area
doubleScalar J = gSum( p.boundaryField()*patch.magSf() );

dco::t1s::get(J,sens,1);
```

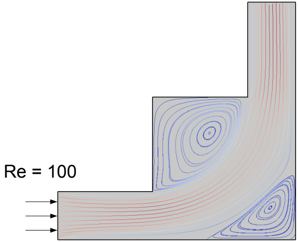
Need to do this N-times to get full sensitivity field!

Black-Box adjoint Version of `simpleFoam`, calculates gradient of $J$:

```
dco::a1s::static_tape tape(tapeSize);
double* sens = new double[alpha.size()];

for(int i=0; i<alpha.size(); i++)
  tape.register_variable(alpha[i]);

for (runTime++; !runTime.end(); runTime++)
{
     ... // solve for U,p
}
// Sum pressure over inlet faces scaled with face area
doubleScalar J = gSum( p.boundaryField()*patch.magSf() );
dco::a1s::set(J,1,-1);
tape.interpret_reverse();

for(int i = 0; i<alpha.size(); i++)
     dco::a1s::get(alpha[i],sens[i],-1);
```

# Test-Case Geometry

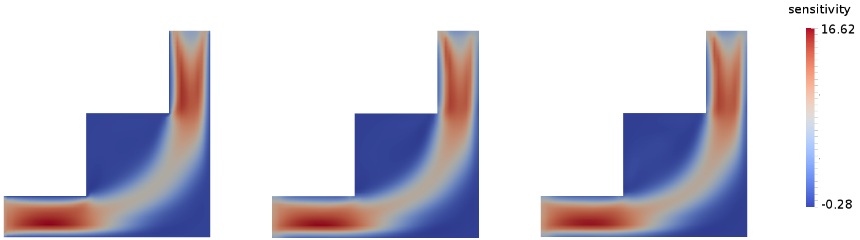
- laminar flow with $\text{Re} = 100$
- vortex areas in the corners
- $J = \int_\Gamma p \, \mathrm{d}\Gamma$
- calculate sensitivity $\frac{\partial J}{\partial \alpha}$
- black-box approach is used



# Results

- 1300 cubic cells
- differentiate over all (pseudo)-timesteps
- plotted: sensitivity with respect to flow resistance $\alpha$

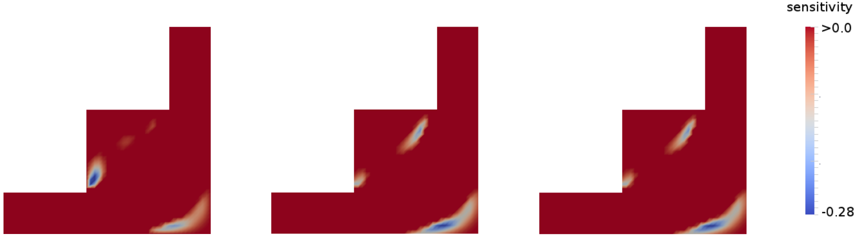
adjointSimpleFoam[2] simpleFoam t1s simpleFoam a1s




[2] `adjointShapeOptimizationFoam` in OpenFOAM 2.1.0

- plotted: sensitivity w.r.t. flow resistance $\alpha$, capped above zero
- finite-difference version available, but finding right $\Delta\alpha$ is not trivial

adjointSimpleFoam simpleFoam t1s simpleFoam a1s

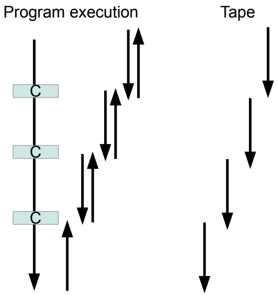


# Strategies to Reduce Memory Usage

**gray-box:** use strategies like checkpointing to store only parts of the program run

**white-box:** exploit the structure of the program

# Checkpointing

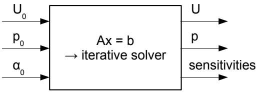



- save states of the forward run to reexecute the program from there to generate a new chunk of tape
- for efficient placement of the checkpoints see[3]

[3] A. Griewank, A. Walther: *Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation.* Transactions on Mathematical Software Vol. 26.1, 2000.

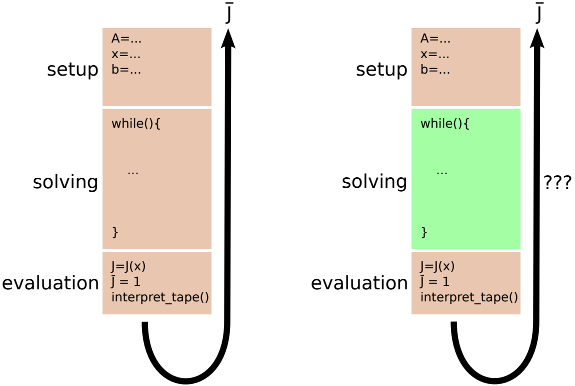
# White-box AD





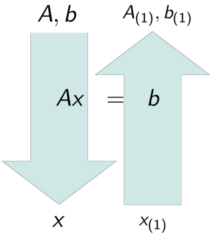

- most of the time is spent inside the solver loop
- this leads to the memory requirements
- if A is linear we can stop taping inside the loop → semi-discrete
- but further iteration is needed in the backward run

# Semi-Discrete Mode



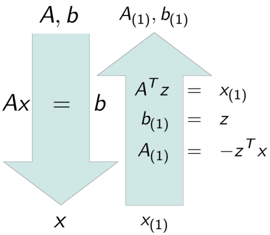
# Semi-Discrete Mode

Tape On:




Data is stored in forward run and interpreted in backward run

Tape Off:




No data is stored in forward run, another equation system needs to be solved in backward run

# Conclusion



- black-box approach works but is limited to small problems
- checkpointing schemes can help to tackle bigger problem sizes, but at the expense of computing time
- white-box ad approach has the potential to enable much bigger problem sizes

Thank you for your attention!