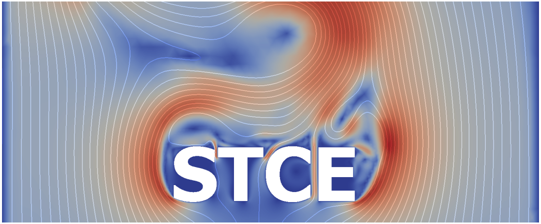


Discrete OpenFOAM

Markus Towara, Arindam Sen
Software and Tools for Computational Engineering Science
RWTH Aachen University

OPT-i, Kos Greece, June 4th 2014

Outline

OpenFOAM Introduction

From OpenFOAM 1.7 to 2.3

Features of discrete adjoint OpenFOAM

Outline

OpenFOAM Introduction

From OpenFOAM 1.7 to 2.3

Features of discrete adjoint OpenFOAM

OpenFOAM

- **Open F**ield **O**peration and **M**anipulation
- Open-Source (GPLv3) CFD solver
- developed by OpenCFD Ltd., currently at version 2.3.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

CFD Basics

- In our applications usually some form of Navier Stokes equation
- Navier Stokes equations for incompressible steady flow:

$$\mathbf{v} \cdot \nabla \mathbf{v} = \nu \nabla^2 \mathbf{v} - \frac{1}{\rho} \nabla p \qquad \text{momentum conservation}$$

$$\nabla \cdot \mathbf{v} = 0 \qquad \text{mass conservation}$$

- Or in three Dimensions:

$$v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} + v_z \frac{\partial v_x}{\partial z} = -\frac{\partial p'}{\partial x} + \nu \left(\frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_x}{\partial y^2} + \frac{\partial^2 v_x}{\partial z^2} \right)$$

$$v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} + v_z \frac{\partial v_y}{\partial z} = -\frac{\partial p'}{\partial y} + \nu \left(\frac{\partial^2 v_y}{\partial x^2} + \frac{\partial^2 v_y}{\partial y^2} + \frac{\partial^2 v_y}{\partial z^2} \right)$$

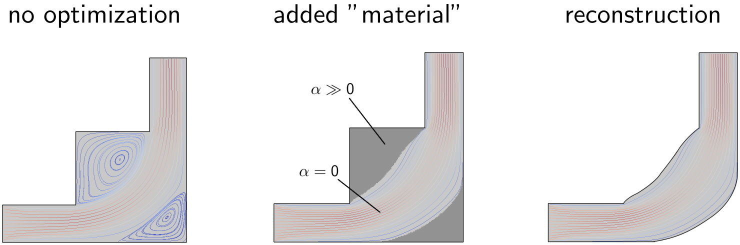
$$v_x \frac{\partial v_z}{\partial x} + v_y \frac{\partial v_z}{\partial y} + v_z \frac{\partial v_z}{\partial z} = -\frac{\partial p'}{\partial z} + \nu \left(\frac{\partial^2 v_z}{\partial x^2} + \frac{\partial^2 v_z}{\partial y^2} + \frac{\partial^2 v_z}{\partial z^2} \right)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0$$

(Decoupled) Solution of the partial differential equations (SIMPLE-Algorithm):

- discretize / linearize momentum conservation equations
- solve momentum equations for velocity v, assume pressure p as known
- obtained velocity field fulfills momentum equation but not mass conservation equation as the pressure field was guessed and not correct
- discretize mass conservation equation
- use mass conservation equation to correct pressure field
- use new pressure field to correct velocity field
- Loop...

Topology Optimization



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:

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

- 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...

How to get derivatives?

- we coupled our operator-overloading tool `dco/c++` with OpenFOAM
- allows us to compute (arbitrary) first order derivatives in adjoint mode
- process descriped in [2]

[2] A Discrete Adjoint Model for OpenFOAM Proceedings of the International Conference on Computational Science, ICCS 2013

Outline

OpenFOAM Introduction

From OpenFOAM 1.7 to 2.3

Features of discrete adjoint OpenFOAM

- Started work on discrete OpenFOAM in late 2011
- up until now discrete adjoint OpenFOAM was using legacy version 1.7 of OpenFOAM (ca. 2010)
- at this time source was provided as tarball
- many newer cases are incompatible / need adjusting to run with OpenFOAM 1.7
- introduced support for OpenFOAM 2.3.x (2014) from scratch, working on official Git-Repo from Git-Hub
- merge for future releases of OpenFOAM should hopefully be way easier

OpenFOAM a1s mode

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

```
namespace Foam
{

  typedef double doubleScalar;
  ...

}
```

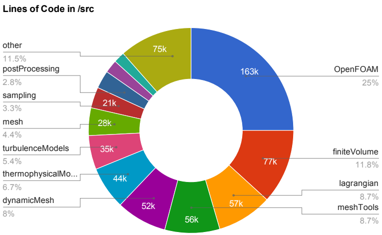
with:

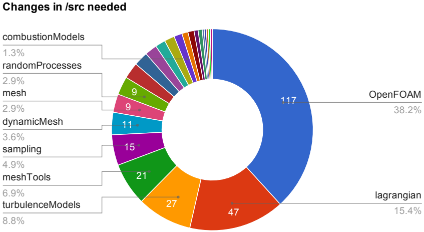
```
#include "dco.hpp"
namespace Foam
{

  typedef dco::a1s::type doubleScalar;
  ...

}
```

in Reality

- some changes have to be made in the OpenFOAM code:
  - unions don't support active datatypes
  - some casts are missing and have to be done by hand
    - i.e. `int i = 2.5` is fine, `int i = dco::a1s::type(2.5)` is not
  - some function macros (pow,max,min) have to be adapted
  - some templates have to be instantiated by hand (i.e. for double)

OpenFOAM LOC Analysis





Outline

OpenFOAM Introduction

From OpenFOAM 1.7 to 2.3

Features of discrete adjoint OpenFOAM

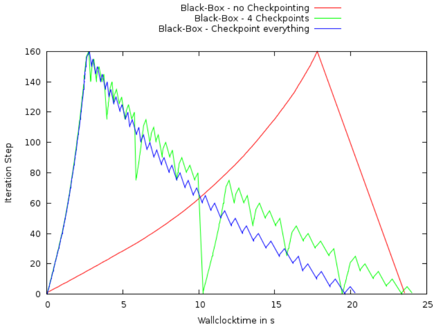
discrete Adjoint solvers

- incompressible steady flow
  - based on `simpleFOAM`
  - checkpointing support
  - optimization with steepest descent
  - reverse accumulation work in progress
- incompressible unsteady flow
  - based on `pisoFOAM`
  - checkpointing support
  - optimization with steepest descent (unsteady cost function, steady solution)
- compressible steady flow with coupled heat transfer
  - based on `chtMultiRegionSimpleFOAM`
  - Work in progress

Checkpointing Interface

- supports revolve (offline) and equidistant checkpointing
- if checkpoint size $\ll$ tape size both shemes perform similar

First Results



- Black-Box approach severely bound by memory bandwith! (Need to allocate around 20 GB)
- By doing Checkpointing we can actually get faster...

How to improve?

- Knowledge and Profiling reveals that most of calculation time and tape memory is spent in (iterative) linear solvers
- Analytical insight allows us to treat the adjoints of linear solvers analytically

Lemma

For a Linear Equation Sytem $Ax = b$ *we can calculate the adjoints for A and b by:* [3]

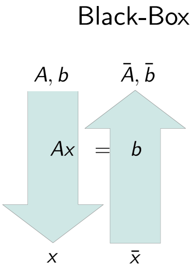
$\bar{b} = A^{-T}\bar{x} \rightarrow A^T\bar{b} = \bar{x}$

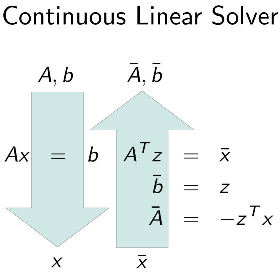
$\bar{A} = -\bar{b} \cdot x^T$

- This gives us an additional Linear Equation System which we have to solve during the gathering of the adjoints

[3] M. Giles, Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation, *Advances in Automatic Differentiation 2008*

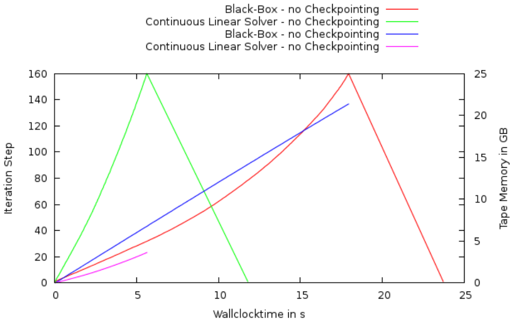
Treatment of Linear Solvers





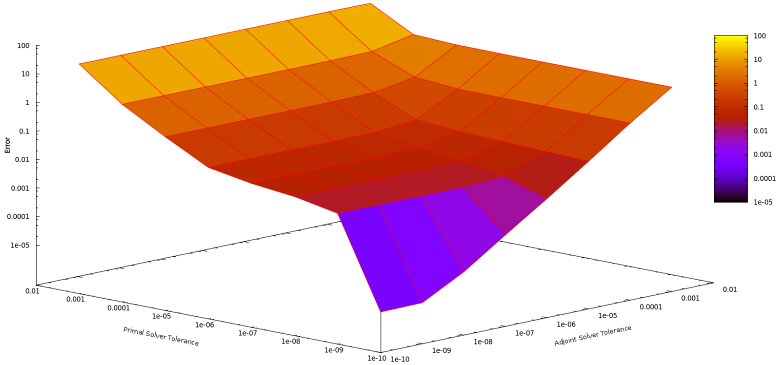
- when we encounter an linear solver during the augmented forward run we can stop taping
- when we encounter the gap in the tape during the interpretation we have to fill in the gap by hand
- have to remember A and x!

Results



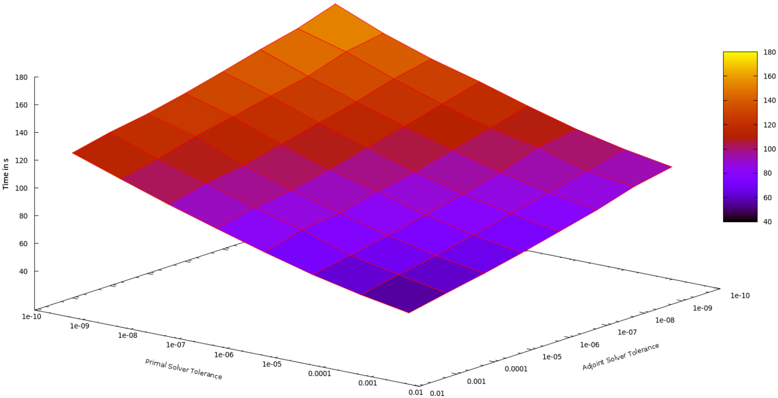
- we are seeing a nice improvement in both runtime and memory usage

Error



Error
100
10
1
0.1
0.01
0.001
0.0001
1e-05
0.01
0.001
0.0001
1e-05
1e-06
1e-07
1e-08
1e-09
1e-10
Primal Solver Tolerance
1e-10
1e-09
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
Adjoint Solver Tolerance

Runtime



Time in s
180
160
140
120
100
80
60
40
1e-10
1e-09
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
Primal Solver Tolerance
Adjoint Solver Tolerance

Summary and Outlook

- Black-Box discrete adjoint methods: Stringent memory requirements, not so fast.
- Improvements via Checkpointing and Linear solver treatment.
- further improvements possible for steady cases through reverse accumulation (implemented $\rightarrow$ EuroAD)
- parallelisation with (A)MPI still to be done

Thank you!
Questions?