Discrete Adjoint of an implicit coupled solver based on foam-extend using Algorithmic Differentiation

Arindam Sen
Markus Towara
Uwe Naumann

Software and Tools for Computation Engineering
RWTH, Aachen University

16th EuroAD workshop
December 8-9, 2014, Jena, Germany
CFD@STCE

- OpenFOAM:
  - Arindam Sen
  - Markus Towara
- ACE+ and GPDE:
  - Zahrasadat Dastouri
- dco developers:
  - Johannes Lotz (dco/c++)
  - Klaus Leppkes (dco/c++)
  - Jan Riehme (dco fortran)
Contents

Introduction

adjointpUcoupledFoam

Results

Adjoint chtMultiRegionSimpleFoam
Contents

Introduction

adjointpUCoupledFoam

Results

Adjoint chtMultiRegionSimpleFoam
Adjoints based CFD Optimization: Approaches

- Discrete Approach by Overloading vis-a-vis continuous approach:
  - Elegance and adaptability
  - Gradients up to machine accuracy, No non-differentiability issues, robustness.
  - No tedious analytical derivations
  - Large runtimes for medium-large scale problems, Unsteady problems in particular
  - Scalability

- Strategies: Checkpointing, Analytical treatment of the Linear solver\(^1\)

- May not be enough, Bottleneck!

\(^1\) M. Giles, Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation, *Advances in Algorithmic Differentiation*, Springer 2008
foam-extend: Introduction and features

- Foam-extend: It is a fork for OpenFOAM retaining a lot of its structural features with additional extensions.
- foam-extend-3.1: Finite Volume based CFD code, Open Source under GPL, implemented in C++
- Key features include:
  - Mixing plane boundary for turbomachinery applications
  - Density-based coupled Roe flux solver
  - Dynamic mesh capability
  - GPU support - cuda solvers
- Focus: Pressure based coupled p/U solver and it’s adjoint.
pUcoupledFoam

- Faster convergence in terms of iterations and runtime.
- Lower under-relaxation
- Increased memory cost (Further concern for adjoint problems)
- Loose coupling causes diminished performance, mostly non-symmetric.

Incompressible steady-state solver with coupled p-U using Rhie-Chow interpolation.
Uses block-matrix implementation, Tensor matrix coefficients instead of scalar, Templated coefficients compared to the basic lduMatrix.
Directly solves multiple equations and couplings in one matrix system.

2. Klas Jareteg, Vuko Vukelić, Hrvoje Jasak, pUCoupledFoam - an open source coupled incompressible pressure-velocity solver based on foam-extend, 9th OpenFOAM workshop, Zagreb, Croatia, 06.2014
pUCoupledFoam

- + Faster convergence in terms of iterations and runtime.
- + Lower under-relaxation
- - Increased memory cost (Further concern for adjoint problems)
- - Loose coupling causes diminished performance, mostly non-symmetric.
- Incompressible steady-state solver with coupled p-U using Rhie-Chow interpolation.
- Uses block-matrix implementation, Tensor matrix coefficients instead of scalar, Templated coefficients compared to the basic lduMatrix.
- Directly solves multiple equations and couplings in one matrix system

---

Governing Equation and Objective Function

Incompressible steady-state Navier-Stokes equations with resistance term $\alpha^3$ in momentum equation:

\[(v \cdot \nabla)v = -\nabla p + \nabla \cdot (2\nu D(v)) - \alpha v \quad (momentum)\]  
\[\nabla \cdot v = 0 \quad (continuity)\]

Cost Function, \(J = \int p + 0.5v^2 \, d\tau\)

\[\frac{\partial J}{\partial \alpha} = ?\]  

Contents

Introduction

adjointpUcoupledFoam

Results

Adjoint chtMultiRegionSimpleFoam
Overloading with dco

- put dco into src/foam
- Overloading operators, basic mathematical functions and data types
- include dco.hpp and link all the dco files
- replace the custom Foam datatype, scalar with the generic dco adjoint datatype (dco::ga1s<double>)
- recompile the source

- Doesn’t work blackbox!
- Additional steps:
  1. Adapt to the overloading tool, for example, remove unions
  2. Remove explicit namespacing, for example, Foam::sqrt becomes sqrt
  3. Casting to deal with assignment operations, for example, assignment of dco type to int/float
  4. and others...
Overloading with dco

- put dco into src/foam
- Overloading operators, basic mathematical functions and data types
- include dco.hpp and link all the dco files
- replace the custom Foam datatype, scalar with the generic dco adjoint datatype (dco::ga1s<double>)
- recompile the source
- Doesn’t work blackbox!
- Additional steps:
  1. Adapt to the overloading tool, for example, remove unions
  2. Remove explicit namespacing, for example, Foam::sqrt becomes sqrt
  3. Casting to deal with assignment operations, for example, assignment of dco type to int/float
  4. and others...
Overloading with dco

- put dco into src/foam
- Overloading operators, basic mathematical functions and data types
- include dco.hpp and link all the dco files
- replace the custom Foam datatype, scalar with the generic dco adjoint datatype (dco::ga1s<double>)
- recompile the source
- **Doesn’t work blackbox!**
- Additional steps:
  1. Adapt to the overloading tool, for example, remove unions
  2. Remove explicit namespacing, for example, Foam::sqrt becomes sqrt
  3. Casting to deal with assignment operations, for example, assignment of dco type to int/float
  4. and others...
adjointpUcoupledFoam using dco/c++

The following steps are undertaken to obtain the Discrete adjoint version of a solver in foam:

   ```
   fvVectorMatrix UEqn
   (    fvm::div(phi, U) + turbulence->divDevReff(U) +
   fvm::Sp(alpha, U) );
   ```

2. Memory allocation for the tape.
   ```
   dco::ga1s<double>::global_tape =
   dco::ga1s<double>::tape_t::create();
   ```

3. Initializing the cost functions
   ```
   scalar J = 0;
   ```

4. Registering the individual entries of alpha as inputs
   ```
   for(int i=0; i<N; i++)
   dco::ga1s<double>::global_tape->register_variable(alpha[i]);
   ```
adjointpUcoupledFoam using dco/c++

The following steps are undertaken to obtain the Discrete adjoint version of a solver in foam:

   ```cpp
   fvVectorMatrix UEqn
   ( fvm::div(phi, U) + turbulence->divDevReff(U) +
   fvm::Sp(alpha, U) );
   ```

2. Memory allocation for the tape.
   ```cpp
   dco::ga1s<double>::global_tape =
   dco::ga1s<double>::tape_t::create();
   ```

3. Initializing the cost functions
   ```cpp
   scalar J = 0;
   ```

4. Registering the individual entries of alpha as inputs
   ```cpp
   for(int i=0; i<N; i++)
   dco::ga1s<double>::global_tape->register_variable(alpha[i]);
   ```
The Discrete adjoint version of solver

5. Evaluation of the cost function $(J = \int_T p + 0.5v^2 \, d\tau)$
   
   ```
   forAll(pressurePatches(), prpI)
   label patchI = mesh.boundaryMesh().findPatchID(pressurePatches()[prpI]);
   const fvPatch patch = mesh.boundary()[patchI];
   J += gSum(p.boundaryField()[patchI]*patch.magSf());
   ```

6. Calculate sensitivities of $J$ with respect to inputs $\alpha$ and reverse interpretation of tape.
   ```
   dco::ga1s<double>::set(J, 1.0, -1);
   dco::ga1s<double>::global_tape->interpret_adjoint();
   ```

7. Retrieving the calculated sensitivities.
   ```
   for(int i = 0; i < N; i++)
   dco::ga1s<double>::get (alpha[i], sens[i], -1);
   ```
The Discrete adjoint version of solver

5. Evaluation of the cost function \( J = \int_{\tau} p + 0.5v^2 \, d\tau \)
   
   ```
   forAll(pressurePatches(), prpI)
   label patchI = mesh.boundaryMesh().findPatchID(
       pressurePatches()[prpI]);
   const fvPatch patch = mesh.boundary()[patchI];
   J += gSum(p.boundaryField()[patchI]*patch.magSf());
   ```

6. Calculate sensitivities of \( J \) with respect to inputs \( \alpha \) and reverse interpretation of tape.
   ```
   dco::ga1s<double>::set(J, 1.0, -1);
   dco::ga1s<double>::global_tape->interpret_adjoint();
   ```

7. Retrieving the calculated sensitivities.
   ```
   for(int i = 0; i < N; i++)
   dco::ga1s<double>::get (alpha[i], sens[i], -1);
   ```
Contents

Introduction

adjointpUcoupledFoam

Results

Adjoint chtMultiRegionSimpleFoam
Test Case: backward facing step

- Comparison of adjointpUcoupledFoam with adjointsimpleFoam
- No. of cells: 4800

Figure: Velocity profile and Mesh
Test Case: backward facing step

Figure: Sensitivities capped at 0, potential regions where optimization could block cells.

adjointShapeOptimizationFoam by Engys, OpenFOAM (v2.3.0)

adjointsimpleFoam based on OpenFOAM (v2.3.x)

adjointpUcoupledFoam based on foam-extend (v3.1)
### Table: Memory and Runtime

<table>
<thead>
<tr>
<th>Solver</th>
<th>Primal RAM</th>
<th>Primal Time</th>
<th>Adjoint RAM</th>
<th>Adjoint Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoupled</td>
<td>139 MB</td>
<td>70s</td>
<td>906 MB</td>
<td>778s</td>
</tr>
<tr>
<td>Coupled</td>
<td>175 MB</td>
<td>11s</td>
<td>13GB</td>
<td>79s</td>
</tr>
</tbody>
</table>
Challenges

- Memory requirements are huge!
- Almost 90% of time spent on Linear Solvers
- Two options: Checkpoint Linear Solver (Overcompensation in time!), Analytical Treatment of Linear solver
Basics

• Used for Conjugate heat transfer between solid and liquid.
• Handles secondary solid or fluid region which can be coupled thermally with the main fluid region.
• Steady state version of chtMultiRegionFoam that combines heatConductionFoam and buoyantFoam
• Initial work of Fabian Key
• Test cases support: Volkswagen
Governing Equation and Objective Function

Steady-state Navier-Stokes equations for fluid domain with resistance term $\alpha$ in momentum equation:

$$\frac{\partial (\rho v_j)}{\partial x_j} = 0$$  \hspace{1cm} (4)

$$\frac{\partial (\rho v_j v_i)}{\partial x_j} - \frac{\partial}{\partial x_j} \left[ (\mu + \mu_t) \left( \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} - \frac{2}{3} \delta^j_i \frac{\partial v_k}{\partial x_k} \right) \right]$$

$$+ \frac{\partial p}{\partial x_i} - \frac{\partial (\rho g_k x_k)}{\partial x_i} - \alpha v_i = 0$$  \hspace{1cm} (5)

$$\frac{\partial (\rho v_j h)}{\partial x_j} + \frac{1}{2} \frac{\partial (\rho v_j v^2_k)}{\partial x_j} - \frac{\partial}{\partial x_j} \left[ (a + a_t) \frac{\partial h}{\partial x_j} \right] = 0$$  \hspace{1cm} (6)

Heat equation for solid domain:

$$- \frac{\partial}{\partial x_j} \left( k \frac{\partial T}{\partial x_j} \right) = 0,$$  \hspace{1cm} (7)
Primal Solution

Figure: Pressure and Temperature Field after 9000 Steps

Objective function: Power Loss

\[
J = - \int_{\Gamma} \left( p + \frac{1}{2} \rho v_j^2 \right) v_i n_i \, d\Gamma
\]  

with \( \Gamma = \text{inlet} \cup \text{outlet} \)
Results

Figure: Sensitivity Distribution (negative sensitivities indicate growth of material)

- Run for 1100 steps with 40 equidistant checkpoints
- Number of mesh points: 150000(Fluid) + 15000(Solid)
- Runtime: 12 min (Primal), 105 min (Adjoint)
- Size of checkpoint database: 250 MB
- Maximal tape size: 22 GB

Future work: Other potential cost functions like Cooling efficiency
Acknowledgements

This work has been conducted within the About Flow project on “Adjoint-based optimisation of industrial and unsteady flows”.

http://aboutflow.sems.qmul.ac.uk

About Flow has received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under Grant Agreement No. 317006.