Block coupled matrix solvers in foam-extend-3 and more

Teaching within: *CFD with OpenSource software* (TME050)

Klas Jareteg

klas.jareteg@chalmers.se
Chalmers University of Technology

2014-09-15

Disclaimer:

- DISCLAIMER: This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD® trade marks. Following the trademark policy.

- DISCLAIMER: The ideas and code in this presentation and all appended files are distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE

Plan and outline

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- `pUCoupledFoam` - Introduction
- `pUCoupledFoam` - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

Learning objectives

At the end of this lesson you should:

- be acquainted with the block coupled format in foam-extend-3.1
- better understand the basics of the pressure-velocity implementation in OpenFOAM
- have some hands on experience with `pUCoupledFoam`
- know some basic git concepts and commands
- better understand templating and object orientation in C++

Conventions/Environments/Code version

- Environments used:

Code 0.1:

Further study of the code

More information 0.1:

For further reading on the subject

Tips 0.1:

More (or less) general tips on OpenFOAM® or related

- Code used: commit 756e3c (part of branch nextRelease, foam-extend-3.0)

Earlier presentations/lectures on related topic

- Henrik Rusche and Hrvoje Jasak. *Implicit solution techniques for coupled multi-field problems – Block Solution, Coupled Matrices*. June 2010
- Ivor Clifford. *Block-Coupled Simulations Using OpenFOAM*. June 2011
- K. Jareteg, V. Vukovic, and H. Jasak. *puCoupledFoam - an open source coupled incompressible pressure-velocity solver based on foam-extend*. 2014. URL: http://www.openfoamworkshop.org/download/OFW09_P_0105.pdf
- K. Jareteg and I. Clifford. *Block coupled matrix solvers in foam-extend-3*. 9th OpenFOAM Workshop, Zagreb. 2014
- K. Jareteg. *Coupled solvers and more - Lecture within CFD with open source 2013 (TME050)*. Lecture slides TME050. 2013. URL: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2013/KlasJareteg_CoupledSolvers_20130917.pdf
- K. Jareteg. "Block coupled calculations in OpenFOAM: A coupled incompressible flow solver". Project work within: CFD with OpenSource software, 2012, Chalmers. Oct. 2012

Block matrix basics

- Block matrix basics
  - General idea
  - Implementation in foam-extend-3
  - Templates in C++ and OpenFOAM
- Pressure and velocity solver
  - `pUCoupledFoam` - Introduction
  - `pUCoupledFoam` - Hands on
- Miscallaneous
  - Explicit pressure-velocity coupling
  - Mesh and matrix formats
  - Git
- Python scripting

Coupled systems

Coupling on many levels:

- Model level (example: couple a turbulence model to your steady state solver)
- Equation level (example: couple the pressure equation to the velocity equation)
- Matrix level (example: `GGI` and `regionCoupling`)

Differ between:

- **explicit coupling:** solve one matrix equation for each variable, use fixed values for all other unknowns (example: velocity components in standard `simpleFoam`)
- **implicit coupling:** directly solve the linear couplings between variables by including multiple equations in the same matrix system

More information 1.1:

Previous trainings on coupled systems:

- "Implicit solution techniques for coupled multi-field problems – Block Solution, Coupled Matrices", OFW5, Henrik Rusche and Hrvoje Jasak

Why block matrices?

Coupled problems often encountered:

- Pressure-velocity
- Multiphase flow
- Solid mechanics
- Heat transfer in a porous medium
- Multiple energy neutronics
- ...

Allows to solve coupled problems implicitly:

- Faster convergence as compared to iterative, explicit methodologies
- Alternative algorithms, not necessarily iterating between equations
- Potentially greater stability for stiff problems

Explicit solver formulations

Examples:

- Velocity components in `simpleFoam` and `pisoFoam`
- Turbulence and momentum equations in `simpleFoam` and `pisoFoam`

Advantages:

- Requires less memory than implicit coupling
- Potentially easier to implement (treating one equation at a time)

Disadvantages:

- Dependencies resolved by Picard or fixed point iterations
- Often slow convergence for strongly coupled problems and potentially divergence for stiff problems. One has to resort to under-relaxation and/or semi-implicit algorithms

Implicit solver formulation

Advantages:

- Increasing the convergence rate, fewer iterations
- Potentially necessary for the problem to converge
- Lower necessary under-relaxation

Disadvantages:

- Increased memory cost, each matrix coefficient a tensor instead of a scalar
- Potentially increased CPU time for weakly coupled problems
- If one equation is non-symmetric and the other symmetric, the block matrix must be non-symmetric

Theory of the block matrix solver

Finite-volume discretization of block coupled equation set

- (u_x, u_y, u_z) in cell P is dependent on $(u_x, u_y, u_z)_P$ in cell P and $(u_x, u_y, u_z)_N$ in cell N
- Off-diagonal entries only for cells sharing a face
- Resulting discretization for cell P

$$\mathbf{a}_P \vec{u}_P + \sum_N \mathbf{a}_N \vec{u}_N = \vec{b}$$

- In current framework $\mathbf{a}\vec{u}$ is written as a tensorial product

$$\mathbf{a}\vec{u} = \begin{bmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$$

- Assemble the sparse linear system

$$[\mathbf{A}]\,[\mathbf{u}] = [\mathbf{b}]$$

Levels of block coupling

Three special cases

- Segregated - no coupling between variables
  - Diagonal and off-diagonal coefficients are in the form of diagonal or spherical tensors

$$\mathbf{a}_P\vec{u}_P+\sum_N \mathbf{a}_N\vec{u}_N = \begin{bmatrix} a_{xx} & & \\ & a_{yy} & \\ & & a_{zz} \end{bmatrix}_P \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}_P + \sum_N \begin{bmatrix} a_{xx} & & \\ & a_{yy} & \\ & & a_{zz} \end{bmatrix}_N \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}_N$$

- Point-implicit - coupling between variables in the owner cell (eg. chemical reactions)
  - Diagonal coefficient is a full tensor

$$\mathbf{a}_P\vec{u}_P+\sum_N \mathbf{a}_N\vec{u}_N = \begin{bmatrix} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{bmatrix}_P \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}_P + \sum_N \begin{bmatrix} a_{xx} & & \\ & a_{yy} & \\ & & a_{zz} \end{bmatrix}_N \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}_N$$

Levels of block coupling

- Fully coupled - coupling between variables in both owner and neighbour cells (eg. stress analysis, adjoint convection)

$$\mathbf{a}_P\vec{u}_P + \sum_N \mathbf{a}_N\vec{u}_N = \left[\begin{array}{ccc} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{array}\right]_P \left[\begin{array}{c} u_x \\ u_y \\ u_z \end{array}\right]_P + \sum_N \left[\begin{array}{ccc} a_{xx} & a_{xy} & a_{xz} \\ a_{yx} & a_{yy} & a_{yz} \\ a_{zx} & a_{zy} & a_{zz} \end{array}\right]_N \left[\begin{array}{c} u_x \\ u_y \\ u_z \end{array}\right]_N$$

- In spirit of generic programming, the aim is to support all levels of coupling using the same underlying functionality
- The size of the block-coupled system is arbitrary (1×1, 2×2, 3×3, , $N\times N$)

Linear solver algorithms

- Sparseness pattern of block matrix is unchanged from scalar matrix
  - Sparseness pattern is mesh dependent
- Iterative solution algorithms use simple operations
  - Vector-matrix multiplication
  - Gauss-Seidel sweep
  - Matrix decomposition
- All readily generalize for matrix with tensor coefficients
  - Simply define primitive operations for N×N coefficients and N-length vectors (coefficient-vector multiplication, coefficient inversion, dot product, etc.)

Block matrix basics

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- `pUCoupledFoam` - Introduction
- `pUCoupledFoam` - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

Implementation in foam-extend-3

- Matrix class `BlockLduMatrix` implemented to handle block matrices in a general, templated manner (Primary development by: Hrvoje Jasak)
- Sparsity pattern preserved, still lower, upper and diagonal:

Code 1.1: $FOAM_SRC/foam/matrices/blockLduMatrix/BlockLduMatrix/BlockLduMatrix.H

```
//- Diagonal coefficients
CoeffField<Type>* diagPtr_;

//- Upper triangle coefficients.  Also used for symmetric matrix
CoeffField<Type>* upperPtr_;

//- Lower triangle coefficients
CoeffField<Type> *lowerPtr_;
```

- Allows `lduMatrix` and `lduMatrixAdressing` to be re-used:

Code 1.2: $FOAM_SRC/foam/matrices/blockLduMatrix/BlockLduMatrix/BlockLduMatrix.H

```
// LDU mesh reference
const lduMesh& lduMesh_;
```

- Compared to the standard `lduMatrix`, the coefficients are now templated

Get to know the code

Tips 1.1:

- Learn to find your way around the code:
  - `grep keyword `find -iname "*.C"``
  - Doxygen (pre-generated: `http://www.openfoam.org/docs/cpp/`)
- Get acquainted with the general code structure:
  - Study the structure of the `src`-directory
  - Try to understand where the matrix classes are found
- When you are writing your own solvers study the available utilities:
  - find how to read variables from dicts scalars, booleans and lists
  - find out how to add an argument to the argument list

Templating in `BlockLduMatrix`

- Coefficient fields are template on `Type`, example diagonal coefficients:

Code 1.3: $FOAM_SRC/foam/matrices/blockLduMatrix/BlockLduMatrix/BlockLduMatrix.H

```
typedef CoeffField<Type> TypeCoeffField;

//- Return diagonal coefficients
const TypeCoeffField& diag() const;
```

- Templating allows different coupled problems to be handled in same structure
- Optimized performance by potential specific implementations for each `Type`
- Types specified includes: `vector2`, `vector3`, `vector4`, `scalar`, `vector`. Also larger vectors can be included by adding a new type (e.g `vector12`)

More information 1.2:

See $FOAM_SRC/foam/primitives/VectorN/ for implementation of e.g. `vector2`

- Note that the structure works also for `scalar` and that `BlockLduMatrix<scalar>` and `fvScalarMatrix` will give equivalent performance

Block matrix basics

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- `pUCoupledFoam` - Introduction
- `pUCoupledFoam` - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

Tips 1.2:

Tips on templates

- Templated functions and classes can operate with generic types.
- Templates are generated at compile time (compare to virtual functions)
- Allows reusing algorithms and classes which are common to many specific types

Example: `List`

- A list could be used different type of contents $\rightarrow$ generic class needed
- `ListI.H`: included already in the header file
- Compilation done for each specific type (remember: generated during compile-time)

Example: `BlockLduMatrix`

- Allow matrix coefficients to be of generic size
- Each `<Type>` must have operators needed defined
- Compilation done for each specific type (remember generated during compile-time)

More information 1.3:

- Read the basics (and more):
  - `http://www.cplusplus.com/doc/tutorial/templates/`
  - Effective C++: 50 Specific Ways to Improve Your Programs and Designs
  - C++ Templates: The Complete Guide (Vandervoorde)

- Look at existing code to see how the templating is implemented, used and compiled ("code explains code")

Pressure and velocity solver

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- pUCoupledFoam - Introduction
- pUCoupledFoam - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

pUCoupledFoam

- New coupled solver released with foam-extend-3.1
- Incompressible pressure-velocity coupled solver, coupled alternative to simpleFoam
- Solver based on an explicit use of Rhie-Chow interpolation

More information 2.1:

Presentation during session "Block coupled":

- "pUCoupledFoam - an open source coupled incompressible pressure-velocity solver based on foam-extend" Klas Jareteg, Vuko Vukcevic, Hrvoje Jasak, 9th OpenFOAM Workshop, Zagreb, 2014

`pUCoupledFoam` - Implicit formulation I

- Navier-Stokes, incompressible, steady-state:

$$\nabla \cdot (\mathbf{U}) = 0 \tag{1}$$

$$\nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla(\nu \nabla \mathbf{U}) = -\frac{1}{\rho}\nabla p \tag{2}$$

- Semi-discretized form:

$$\sum_{\text{faces}} \mathbf{U}_{\text{f}} \cdot \mathbf{S}_{\text{f}} = 0 \tag{3}$$

$$\sum_{\text{faces}} \left[\mathbf{U}\mathbf{U} - \nu \nabla \mathbf{U}\right]_{\text{f}} \cdot \mathbf{S}_{\text{f}} = -\sum_{\text{faces}} P_{\text{f}} \mathbf{S}_{\text{f}} \tag{4}$$

- Modified pressure:

$$\frac{p}{\rho} = P \tag{5}$$

pUCoupledFoam - Implicit formulation II

- Rhie-Chow in continuity equation:

$$\sum_{\text{faces}} \left[\overline{\mathbf{U}_{\mathrm{f}}} - \overline{\mathbf{D}_{\mathrm{f}}} \left(\nabla P_{\mathrm{f}} - \overline{\nabla P_{\mathrm{f}}} \right) \right] \cdot \mathbf{S}_{\mathrm{f}} = 0 \tag{6}$$

where the second and third term introduces Rhie-Chow interpolation, corresponding to the difference between the pressure gradient and the interpolated gradient.

pUCoupledFoam - Coupled equations

- Solution variable of length 4:

$$x^P = \begin{bmatrix} u^P \\ v^P \\ w^P \\ P^P \end{bmatrix} \tag{7}$$

Code 2.1: $FOAM_APP/solvers/coupled/pUCoupledFoam/createFields.H

```
volVector4Field Up
(
    IOobject
    (
       "Up",
       runTime.timeName(),
       mesh,
       IOobject::NO_READ,
       IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedVector4("zero", dimless, vector4::zero)
);
```

- `volVectorField` gives storage for solution variable and allows for coupled boundaries to be used (cyclic, processor, ...)

pUCoupledFoam - Coupled equation discretization I

- Discretizing the momentum equation:

$$\sum_{\text{faces}} [\mathbf{U}\mathbf{U} - \nu\nabla\mathbf{U}]_{\mathrm{f}} \cdot \mathbf{S}_{\mathrm{f}} = -\sum_{\text{faces}} P_{\mathrm{f}}\mathbf{S}_{\mathrm{f}} \tag{8}$$

Code 2.2: $FOAM_APP/solvers/coupled/pUCoupledFoam/UEqn.H

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

- Note that the implicit gradient of the pressure is handled separately:

Code 2.3: $FOAM_APP/solvers/coupled/pUCoupledFoam/calculateCouplingMatrices.H

```
blockVectorMatrix pInU(fvm::grad(p));
```

- Implicit gradient (`fvm::grad`) new operator.

pUCoupledFoam - Coupled equation discretization II

- Continuity equation discretized as:

$$\sum_{\text{faces}} -\overline{\mathbf{D}_\mathrm{f}}\nabla P_\mathrm{f} \cdot \mathbf{S}_\mathrm{f} + \sum_{\text{faces}} \overline{\mathrm{f}} \cdot \mathbf{S}_\mathrm{f} = \sum_{\text{faces}} -\overline{\mathbf{D}_\mathrm{f}}\,\overline{\nabla P_\mathrm{f}} \cdot \mathbf{S}_\mathrm{f} \tag{9}$$

Code 2.4: $FOAM_APP/solvers/coupled/pUCoupledFoam/pEqn.H

```
fvScalarMatrix pEqn
(
  - fvm::laplacian(rUAf, p)
 ==
  - fvc::div(presSource)
);
```

- Implicit divergence (`fvm::div`) new operator

Code 2.5: $FOAM_APP/solvers/coupled/pUCoupledFoam/calculateCouplingMatrices.H

```
blockVectorMatrix UInp(fvm::div(U));
```

- Implicit contributions added using new block matrix functions:

Code 2.6: $FOAM_APP/solvers/coupled/pUCoupledFoam/pUCoupledFoam.C

```
// of U and p in the system. This could be better. VV, 30/April/2014
blockMatrixTools::insertBlockCoupling(3, 0, UInp, U, A, b, false);
```

- System solved and solution retrieved:

Code 2.7: $FOAM_APP/solvers/coupled/pUCoupledFoam/pUCoupledFoam.C

```

    // Solve the block matrix
    BlockSolverPerformance<vector4> solverPerf =
        BlockLduSolver<vector4>::New
        (
            word("Up"),
            A,
            mesh.solutionDict().solver("Up")
        )->solve(Up, b);

    solverPerf.print();

    // Retrieve solution
    blockMatrixTools::retrieveSolution(0, U.internalField(), Up);
    blockMatrixTools::retrieveSolution(3, p.internalField(), Up);

    U.correctBoundaryConditions();
```

More information 2.2:

Use of `tmp`

- `tmp` is used to minimize the computational effort in the code
- In general C++ will create objects in local scope, return a copy and destroy the remaining object
- This is undesired for large objects which gives lots of data transfer
- To avoid the local object to be out of scope the `tmp` container is used
- Example in operators returning a discretized equation

Source and more info: `http://openfoamwiki.net/index.php/OpenFOAM_guide/tmp`

Code 2.8: $FOAM_SRC/finiteVolume/finiteVolume/divSchemes/gaussDivScheme/gaussDivScheme.C

```
template<class Type>
tmp<blockVectorMatrix> gaussDivScheme<Type>::fvmDiv
(
    const GeometricField<Type, fvPatchField, volMesh>& vf
) const
{
    tmp<surfaceScalarField> tweights = this->tinterpScheme_().weights(vf);
    const scalarField& wIn = tweights().internalField();

    const fvMesh& mesh = vf.mesh();

    tmp<blockVectorMatrix> tbm
    (
        new blockVectorMatrix
        (
           mesh
        )
    );
    blockVectorMatrix& bm = tbm();
```

pUCoupledFoam - Benchmarking I

Case 1: Backward facing step

- Structured mesh, 4800 cells
- Comparison of simpleFoam and pUCoupledFoam
- Under-relaxation in pUCoupledFoam: none in pressure, 0.995 in U

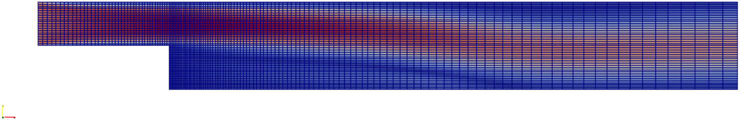


Figure: Geometry and velocity solution for back facing step case

- Major performance increase, both considering number of iterations and elapsed time
- Convergence per iteration is same for both matrix solvers using pUCoupledFoam

pUCoupledFoam - Benchmarking II

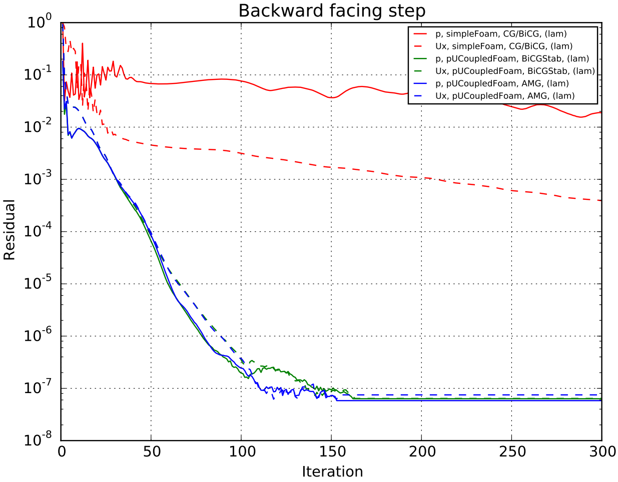


Figure: Performance of `simpleFoam` compared to `pUCoupledFoam`.

pUCoupledFoam - Benchmarking III

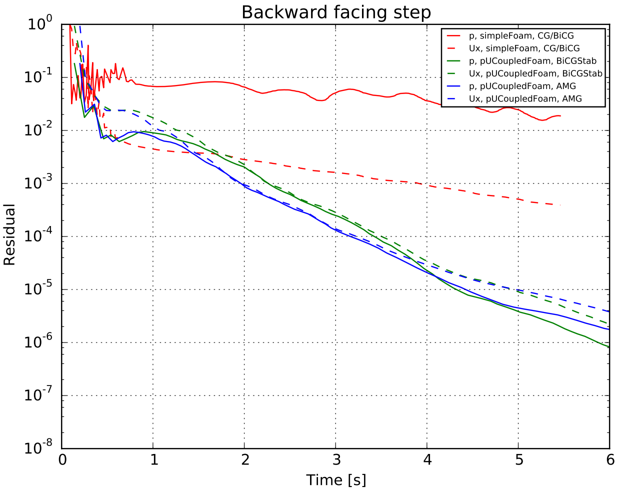


Figure: Performance of simpleFoam compared to pUCoupledFoam.

pUCoupledFoam - Benchmarking IV

Case 2: Munk M3 airfoil in 2D

- Unstructured mesh, 36410 cells
- Comparison of `simpleFoam` and `pUCoupledFoam`
- Under-relaxation in `pUCoupledFoam`: none in pressure, 0.85 in U

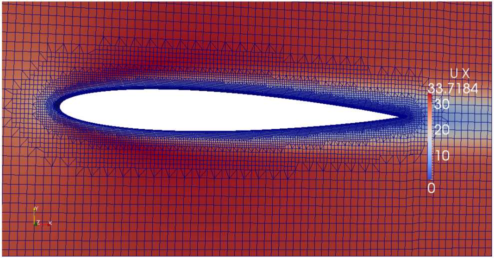


Figure: Geometry (zoomed) and pressure solution for Munk M3 airfoil case.

- Major performance increase, both considering number of iterations and elapsed time
- Better performance for the AMG solver

pUCoupledFoam - Benchmarking V

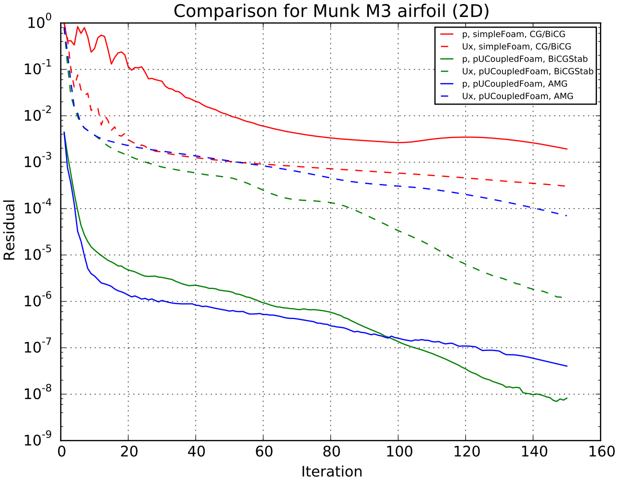


Figure: Performance of `simpleFoam` compared to `pUCoupledFoam`.

pUCoupledFoam - Benchmarking VI

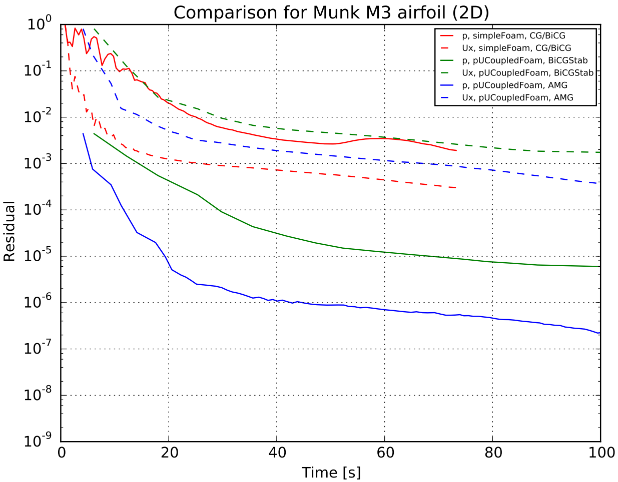


Figure: Performance of `simpleFoam` compared to `pUCoupledFoam`.

Pressure and velocity solver

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- pUCoupledFoam - Introduction
- pUCoupledFoam - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

Test cases

Existing tutorial cases for `pUCoupledFoam`:

- `cavity`
- `backwardFacingStepLaminar`
- `backwardFacingStepTurbulent`

Interesting to compare to `simpleFoam`:

- Compare the results (pressure, velocity, ...)
- Compare the convergence rates
- Compare the running times

... a script is needed!

`pyBenchmark` - a test utility example

Python script to benchmark the coupled solver, abilities:

- Read a configuration file to setup all cases
- Run cases and extract elapsed time and iteration counts
- Generate Matplotlib figures of the performance of the compared cases
- Run the separate cases and plots in subprocesses, parallelizing the script
- Wrapper around PyFoam

pyBenchmark - config I

Based on a config file parsed using ConfigParser:

- **General:** Listing the different cases to be run. The cases refer to one section each. Also listing the different plots to be generated.

Code 2.9: $FOAM_APP/Code/benchmark.cfg

```
[General]
cases: cavity
root: compared_cases
plots: cavity_per_time cavity_per_iteration
```

- **Cases:** One section per case. Each case including the path to the template of the case, whether blockMesh should be run and which solver settings should be tested.

Code 2.10: $FOAM_APP/Code/benchmark.cfg

```
[cavity]
solvers: pU_BiCG pU_AMG simpleFoam_BiCG
template: templates/cavity
blockMesh: True
```

pyBenchmark - config II

- **Solvers:** Specifying the information for the solver to be used (under-relaxation, matrix solvers, etc.)

Code 2.11: $FOAM_APP/Code/benchmark.cfg

```
[pU_BiCG]
solver: pUCoupledFoam
label: BiCGStab
fields: Up
Up: solver    BiCGStab
    tolerance 1e-07
    relTol    0.001
    maxIter   500
    minIter   0
    preconditioner Cholesky
underrelax: p None U 0.995 k 0.95 epsilon 0.95
```

pyBenchmark - config III

- **Plots:** One section for each plot to be generated. Each plot is based on a case section and a set of solvers compared.

Code 2.12: $FOAM_APP/Code/benchmark.cfg

```
[cavity_per_time]
type: residuals
xaxis: t linear
xlabel: Time [s]
ylabel: Residual
yaxis: log
residuals: p Ux
title: Comparison for cavity case
labels: variable solver matrixsolver
outputname: plots/cavity_per_time
outputtypes: pdf
cases:  cavity cavity cavity
solvers: simpleFoam_BiCG pU_BiCG pU_AMG
```

pyBenchmark - getting and running the script

Checkout via git (username="block", password="pUCoupledFoam"):

```
git clone http://klas.nephy.chalmers.se/code/PyBenchmark .
```

Run the scripts from the `example` directory:

```
./runBenchmark.py -h
Usage: Run benchmarking of different solvers

Options:
  -h, --help            show this help message and exit
  -d, --debug           Debug from logger
  -c CONFIGFILE, --configfile=CONFIGFILE
                        Config file
  -p, --plot            Plot according to config file
  -r, --run             Run the benchmarking cases
  -s, --setup           Setup the benchmarking cases
```

Miscallaneous

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- pUCoupledFoam - Introduction
- pUCoupledFoam - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

Incompressible flow

Acknowledgement for description: Professor Hrvoje Jasak

- For low Mach numbers the density and pressure decouple.
- General Navier-Stokes equations simplify to:

$$\nabla \cdot (\mathbf{U}) = 0 \tag{10}$$

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U}\mathbf{U}) - \nabla(\nu \nabla \mathbf{U}) = -\frac{1}{\rho}\nabla p \tag{11}$$

- Non-linearity in the equation ($\nabla \cdot (\mathbf{U}\mathbf{U})$) resolved by iteration
- Continuity equation requiring the flow to be divergence free
- No explicit pressure dependence for the divergence free criterion. Pressure equation must be derived.

Incompressible flow - equation coupling I

- Pressure equation retrieved from the continuity equation.
- Start by a semi-discretized form of the momentum equation:

$$a_P \mathbf{U}_P = \mathbf{H}(\mathbf{U}) - \nabla P \tag{12}$$

where:

$$\mathbf{H}(\mathbf{U}) = \sum_N a_N^{\mathbf{U}} \mathbf{U}_N \tag{13}$$

and rearranged to:

$$\mathbf{U}_P = (a_P^{\mathbf{U}})^{-1} \mathbf{H}(\mathbf{U}) - (a_P^{\mathbf{U}})^{-1} \nabla P \tag{14}$$

Incompressible flow - equation coupling II

- Eq. (14) is then substituted in to the continuity equation:

$$\nabla \cdot ((a_P^{\mathbf{U}})^{-1} \nabla P) = \nabla \cdot ((a_P^{\mathbf{U}})^{-1} \mathbf{H}(\mathbf{U})) \tag{15}$$

- Gives two equations: momentum and pressure equation
- Pressure equation will assure a divergence free flux, and consequently the face fluxes ($F = \mathbf{S}_\mathrm{f} \cdot \mathbf{U}$) must be reconstructed from the solution of the pressure equation:

$$F = -(a_P^{\mathbf{U}})^{-1} \mathbf{S}_\mathrm{f} \cdot \nabla P + (a_P^{\mathbf{U}})^{-1} \mathbf{S}_\mathrm{f} \cdot \mathbf{H}(\mathbf{U}) \tag{16}$$

SIMPLE algorithm is primarily used for steady state problems:

1. Guess the pressure field
2. Solve momentum equation using the guessed pressure field (eq. 14)
3. Compute the pressure based on the predicted velocity field (eq. 15)
4. Compute conservative face flux (eq. 16)
5. Iterate

In reality, under relaxation must be used to converge the problem

More information 3.1:

Study the source code of `simpleFoam`:

- Try to recognize the above equations in the code

Rhie-Chow correction

- Rhie and Chow introduced a correction in order to be able to use collocated grids
- This is used also in OpenFOAM, but not in an explicit manner
- The Rhie-Chow like correction will occur as a difference to how the gradient and Laplacian terms in eq. (15) are discretized.

More information 3.2:

Further explanation on the Rhie-Chow interpolation:

- Computational methods for fluid dynamics, Ferziger and Peric
- Description from an OpenFOAM point of view: Peng-Kärrholm: `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2007/rhiechow.pdf`

Miscallaneous

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- `pUCoupledFoam` - Introduction
- `pUCoupledFoam` - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

Matrix format in OpenFOAM I

Matrix:

- Sparse matrix system:
  - No zeros stored
  - Only neighbouring cells will give a contribution
- Basic format of the `lduMatrix`:
  - diagonal coefficients
  - upper coefficients
  - lower coefficients (not necessary for symmetric matrices)

More information 3.3:

Study the code for `lduMatrix`:

- find the diagonal, upper and lower fields

Lazy Evaluation in `lduMatrix`:

- Used to avoid calculation and transfer of unnecessary data
- Example `lduMatrix`:
  - Used for returning the upper part of the matrix (`upper()`)
  - If upper part does not exist it will be created
  - If it already exists it is simply returned
- To achieve lazy evaluation you will see pointers used in OpenFOAM

Matrix format in OpenFOAM II

`lduMatrix`

- Basic square sparse matrix
- Stored in three arrays: the diagonal, the upper and the lower part:

Code 3.1: $FOAM_SRC/foam/matrices/lduMatrix/lduMatrix/lduMatrix.H

```
//- Coefficients (not including interfaces)
scalarField *lowerPtr_, *diagPtr_, *upperPtr_;
```

- Diagonal elements: numbered as cell numbers
- Off-diagonal elements: are numbered according to faces.

Matrix format in OpenFOAM III

Sparsity of matrix:

$$\mathbf{A} = A_{i,j} \tag{17}$$

- i, j: contribution from cell j on cell i
- j, i: contribution from cell i on cell j
- $i > j$: upper elements
- $i < j$: lower elements
- $i = j$: diagonal elements

Matrix format in OpenFOAM IV

`fvMatrix`

- Specialization for finite volume
- Adds source and reference to field
- Helper functions:

Code 3.2: $FOAM_APP/solvers/incompressible/simpleFoam/pEqn.H

```
volScalarField AU = UEqn().A();
U = UEqn().H()/AU;
UEqn.clear();
phi = fvc::interpolate(U) & mesh.Sf();
adjustPhi(phi, U, p);
```

Miscallaneous

Block matrix basics
- General idea
- Implementation in foam-extend-3
- Templates in C++ and OpenFOAM

Pressure and velocity solver
- `pUCoupledFoam` - Introduction
- `pUCoupledFoam` - Hands on

Miscallaneous
- Explicit pressure-velocity coupling
- Mesh and matrix formats
- Git

Python scripting

Git

- Version control system[1] - meant to manage changes and different versions of codes
- Distributed - each directory is a fully functioning repository without connection to any servers
- Multiple protocols - code can be pushed and pulled over HTTP, FTP, ssh ...

[1] Many more version control systems exists, e.g. Subversion and Mercurial

Git - Hands on I

Basics:

- Initialize a repository in the current folder:

```
git init
```

- Check the current status of the repository:

```
git status
```

- Add a file to the revision control:

```
git add filename
```

- Now again check the status:

```
git status
```

- In order to commit the changes:

```
git commit -m "Message that will be stored along with the commit"
```

- List the currents commits using log:

```
git log
```

Git - Hands on II

Branches:

- When developing multiple things or when multiple persons are working on the same code it can be convenient to use branches.
- To create a branch:

```
git branch name_new_branch
```

- List the available branches:

```
git branch
```

- Switch between branches by:

```
git checkout name_new_branch
```

- Branches can be merged so that developments of different branches are brought together.

Ignore file:

- Avoid including compiled files and binary files in the revision tree.
- Add a `.gitignore` file. The files and endings listed in the file will be ignored. Example:

```
# Skip all the files ending with .o (object files)
*.o

# Skip all dependency files
*.dep
```

- When looking at the status of the repository the above files will be ignored.

Git - Information and software

Some documentation:

- Git - Documentation: `http://git-scm.com/doc` (entire book available at: `https://github.s3.amazonaws.com/media/progit.en.pdf`)
- Code School - Try Git: `http://try.github.io/levels/1/challenges/1`
- ... google!

Examples of software:

- Meld - merging tool, can be used to merge different branches and commits (`http://meldmerge.org/`)
- Giggle - example of a GUI for git (`https://wiki.gnome.org/Apps/giggle`)

What is a script language?

- Interpreted language, not usually needed to compile
- Aimed for rapid execution and development
- Examples: Python, Perl, Tcl ...

Why using a script language?

- Automatization of sequences of commands
- Easy to perform data and file preprocessing
- Substitute for more expansive software
- Rapid development

How to run a script language?

- Interactive mode; line-by-line
- Script mode; run a set of commands written in a file

Python basics

- Interpreted language, no compilation by the user
- Run in interactive mode or using scripts
- Dynamically typed language: type of a variable set during runtime

```
foo = "1"
bar = 5
```

- Strongly typed language: change of type requires explicit conversion

```
>>> foo=1
>>> bar="a"
>>> foobar=foo+bar
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python syntax I

- Commented lines start with "#"
- Loops and conditional statements controlled by indentation

```
if 1==1:
    print "Yes, 1=1"
print "Will always be written"
```

- Three important data types:
  - Lists:

```
>>> foo = [1,"a"]
>>> bar = [1, 2, 3, 4]
>>> print foo[0]
1
>>> print bar[:]
[1, 2, 3, 4]
>>> print bar[1:2]
[2]
>>> print bar[-1]
4
>>> bar.append(4)
>>> print bar
[1, 2, 3, 4, 4]
```

Python syntax II

- Tuples:

```
>>> foo = (1,2,3)
>>> print "Test %d use %d of tuple %d" % foo
Test 1 use 2 of tuple 3
```

- Dictionaries:

```
>>> test = {}
>>> test['value']=4
>>> test['name']="test"
>>> print test
{'name': 'test', 'value': 4}
```

Python modules I

Auxiliary code can be included from modules. Examples:

- `os`: Operating system interface. Example:

```
import os

# Run a command
os.system("run command")
```

- `shutil`: High-level file operations

```
import shutil

# Copy some files
shutil.copytree('template','runfolder')
```

Case study: Running a set of simulations I

- Multiple OpenFOAM runs with different parameters
- Example: edits in `fvSolution`:
  - Make a copy of your dictionary.
  - Insert keywords for the entries to be changed
  - Let the script change the keywords and run the application

```python
#!/usr/bin/python

import os
import shutil

presweeps = [2,4]
cycles = ['W','V']

for p in presweeps:
    for c in cycles:
        os.system('rm -rf runfolder')
        shutil.copytree('template','runfolder')

        os.chdir('runfolder')
        os.system("sed -i 's/PRESWEEPS/%d/' system/fvSolution"%p)
        os.system("sed -i 's/CYCLETYPE/%s/' system/fvSolution"%c)
        os.system("mpirun -np 8 steadyNavalFoam -parallel > log.steadyNavalFoam")

        os.chdir('..')
```

Case study: Extract convergence results I

- Run cases as in previous example and additionally extract some running time

```
#!/usr/bin/python

import os
import shutil

presweeps = [2,4]
cycles = ['W','V']

for p in presweeps:
    for c in cycles:
        os.system('rm -rf runfolder')
        shutil.copytree('template','runfolder')

        os.chdir('runfolder')
        os.system("sed -i 's/PRESWEEPS/%d/' system/fvSolution"%p)
        os.system("sed -i 's/CYCLETYPE/%s/' system/fvSolution"%c)
        os.system("mpirun -np 8 steadyNavalFoam -parallel > log.steadyNavalFoam")
        f = open('log.steadyNavalFoam','r')
        for line in f:
            linsplit = line.rsplit()
            if len(linsplit>7):
                if ls[0]=="ExecutionTime":
                    exectime = float(ls[2])
                    clocktime = float(ls[6])
        f.close()
        print "Cycle=%s, presweeps=%d, execution time=%f, clocktime=%f"%(c,p,exectime,clocktime)
        os.chdir('..')
```

Case study: Setting up large cases I

```
#!/usr/bin/python
# Klas Jareteg
# 2013-08-30
# Desc:
#   Setting up the a case with a box

import os,sys,shutil
opj = os.path.join
from optparse import OptionParser
import subprocess

MESH = '/home/klas/OpenFOAM/klas-1.6-ext-git/run/krjPbe/2D/meshes/box/coarse/moderator.blockMesh'
FIELDS = '/home/klas/OpenFOAM/klas-1.6-ext-git/run/krjPbe/2D/meshes/box/coarse/0'

################################################################################
##########################    OPTIONS    #######################################
################################################################################

parser = OptionParser()
parser.add_option("-c", "--clean", dest="clean",
                  action="store_true", default=False)
parser.add_option("-s", "--setup", dest="setup",
                  action="store_true", default=False)
(options, args) = parser.parse_args()


################################################################################
##########################    CLEAN UP  ########################################
################################################################################

if options.clean:
    os.system('rm -fr 0')
    os.system('rm -fr [0-9]*')
```

Case study: Setting up large cases II

```
################################################################################
#########################   SETUP   #############################################
################################################################################

if options.setup:
    shutil.copy(MESH,'constant/polyMesh/blockMeshDict')

    p = subprocess.Popen(['blockMesh'],\
        stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    out, error = p.communicate()

    if error:
        print bcolors.FAIL + "ERROR: blockMesh failing" + bcolors.ENDC
        print bcolors.ENDC + "ERROR MESSAGE: %s"%error + bcolors.ENDC

    try:
        shutil.rmtree('0')
    except OSError:
        pass

    shutil.copytree(FIELDS,'0')
```

Plotting with Python - matplotlib

```
#!/usr/bin/python

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,1)
y = np.linspace(0,2)
y = y**2

plt.figure()
plt.plot(x,y)
plt.title('Test of matplotlib')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('Test.pdf',format='pdf')
```

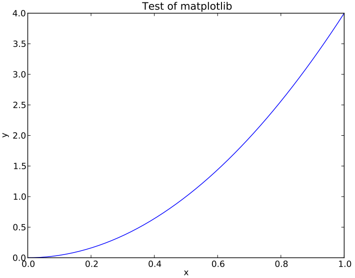


Figure: Example plot from matplotlib

More on plotting

- matplotlib (`http://matplotlib.org/`):
  - Plotting package with MATLAB equivalent syntax
  - Primarily 2D plots
- MayaVi2 (`http://code.enthought.com/projects/mayavi/`):
  - Plots 3D
  - Works with VTK, possible complement to ParaView

Read more

Python introduction material:

- Python tutorial: `http://docs.python.org/2/tutorial/`

Python and high performance computing:

- `http://www.c3se.chalmers.se/index.php/Python_and_High_Performance_Computing`

PyFoam

From documentation:

"This library was developed to control OpenFOAM-simulations with a decent (sorry Perl) scripting language to do parameter-variations and results analysis. It is an ongoing effort. I add features on an As-Needed basis but am open to suggestions."

Abilities:

- Parameter variation
- Manipulation directories
- Setting fields and boundary conditions
- Generate results and plots
- ....

`http://openfoamwiki.net/index.php/Contrib_PyFoam`

More modules

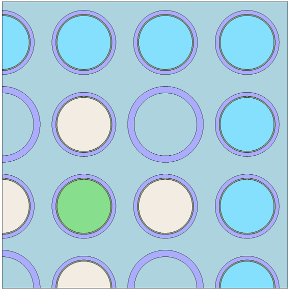
- `logging`: Flexible logging which could be used also for modules.
- `optparse`: Parser for command line options. Example from `http://docs.python.org/2/library/optparse.html`:

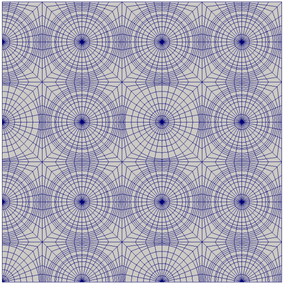
```
from optparse import OptionParser
[...]
parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="write report to FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose", default=True,
                  help="don't print status messages to stdout")

(options, args) = parser.parse_args()
```

- `numpy`: Scientific computing with Python. Information `http://wiki.scipy.org/Tentative_NumPy_Tutorial`
  - Array and matrix operations
  - Linear algebra

Case study: Meshing with Python I

- Library of objects and functions to read a config file and produce a set of meshes and fields





Case study: Meshing with Python II

Needed for simulation:

- All meshes ($16 \times 4+1+1=66$)
- All fields ($\approx$400)
- All coupled patches

Reasons to automatize:

- Changes in mesh configurations (mesh independence tests etc.)
- Change in geometrical configurations
- Change in field initial and boundary conditions
- ....

Case study: Meshing with Python III

Meshes and fields produced from a configuration file read by Python application:

```
[general]
dimensions: 3
convert: 0.01
time: 0

[GeneralAssembly]
name: Generalized assembly mesh
symmetry: 4
nx: 7
lattice:   guid pin0 guid pin0
           pin0 pin0 pin0 pin0
           guid pin0 guid pin0
           pin0 pin0 pin0 pin0

dphi: 8
pitch: 1.25
H: 1.0
dz: 1.0
gz: 1.0
ref: 0.0
ref_dz: 1.0
ref_gz: 1.0

moderatorfields: T p K k epsilon U G
modinnfields: T p K k epsilon U G
neutronicsmultiples: Phi Psi
fuefields: T rho K h p
clafields: T rho K h p
gapfields: T p_gap K k_gap epsilon_gap U_gap G

[pin0]
```

Case study: Meshing with Python IV

```
type: FuelPin
fue_ro: 0.41
fue_ri: 0.12
fue_dr: 4
....
```

Case study: Meshing with Python V

blockMeshDict

```
....
convertToMeters 0.010000;

vertices
(
    (0.000000 0.000000 0.000000)
    (0.070711 0.070711 0.000000)
    (0.055557 0.083147 0.000000)
....
    (4.375000 4.167612 0.000000)
    (4.375000 4.167612 1.000000)
    (1000.000000 1000.000000 1000.000000)
);

blocks
(
    hex ( 0 1 2 2 5 6 7 7 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 2 10 10 5 7 13 13 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 10 16 16 5 13 19 19 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 16 22 22 5 19 25 25 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 166 172 172 5 169 175 175 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 172 178 178 5 175 181 181 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )
    hex ( 0 178 184 184 5 181 187 187 ) ( 1 1 1 ) simpleGrading (1.000000 1.000000 1.000000 )

....
```

Case study: Meshing with Python VI

Summary:

- Using `blockMesh` for structured meshes with many regions
- Need for a script in order to be able to reproduce fast and easy
- Object oriented library written in Python