Block Solvers: Overview, Capabilities, Status and Challenges

Vuko Vukčević

Faculty of Mechanical Engineering and Naval Architecture

April 12, 2016

Objectives:

- Review the handling of complex coupled equations sets in `foam-extend` using block-coupled solution algorithms.
- Show the interface and implementation examples of coupled equation sets.

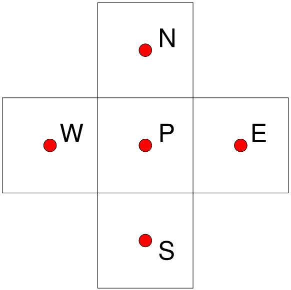
Topics:

1. Introduction on implicit coupling models.
2. Block-coupled solution for multiple variables on a single mesh.
3. Examples of coupled systems:
   - Simple multi-equation coupling: `blockCoupledScalarTransportFoam`.
   - Block-coupled (vector) finite volume operators `fvm`.
   - Block-coupled $p - \mathbf{u}$ algorithm: `pUCoupledFoam`.
   - Block-coupled eddy viscosity turbulence models.

- For cases of strong coupling between the components of a vector, the components can be solved as a **block variable**: (u_x, u_y, u_z) will appear as variables in the same linear system.
- In spite of the fact that the system is much larger, the coupling pattern still exists: components of $\mathbf{u}$ in cell P may be coupled to other components in the same point or to vector components in the neighbouring cell.
- With this in mind, we can still keep the sparse addressing defined by the mesh: if a variable is a vector, a tensorial diagonal coefficients couples the vector components in the same cell. A tensorial off-diagonal coefficient couples the components of $\mathbf{u}_P$ to all components of $\mathbf{u}_N$, which covers all possibilities.

- For **multi-variable block solution** like the compressible Navier-Stokes system above, the same trick is used: the cell variable consists of $(\rho, \rho\mathbf{u}, \rho E)$ and the coupling can be coupled by a 5×5 matrix coefficient.
- Important disadvantages of a block coupled system are:
  - Large linear system: several variables are handled together,
  - Different kinds of physics can be present, *e.g.* the transport-dominated (parabolic/hyperbolic) momentum equation and elliptic pressure equation. At matrix level, it is impossible to separate them, which makes the system more difficult to solve.

- Irrespective of the level of coupling, the FVM dictates that a cell value will depend only on the values in surrounding cells.



- We still have freedom to organise the matrix by ordering entries for various components of ϕ. Also, the matrix connectivity pattern may be changed by reordering the computational points.

- Example: block-coupled vector equation (u_x, u_y, u_z).
  - Per-variable organisation: first u_x for all cells, followed by u_y and u_z. Ordering of each sub-list matches the cell ordering.

$$\mathbf{A}_P = \begin{bmatrix} [u_x \leftrightarrow u_x] & [u_x \leftrightarrow u_y] & [u_x \leftrightarrow u_z] \\ [u_y \leftrightarrow u_x] & [u_y \leftrightarrow u_y] & [u_y \leftrightarrow u_z] \\ [u_z \leftrightarrow u_x] & [u_z \leftrightarrow u_y] & [u_z \leftrightarrow u_z] \end{bmatrix}$$

    Diagonal blocks, *e.g.* $[u_x \leftrightarrow u_x]$ have the size equal to the number of computational points and contain the coupling within the single component. All matrix coefficients are scalars. Off-diagonal block represent variable-to-variable coupling.
  - Per-cell organisation: (u_x, u_y, u_z) for each cell. A single numbering space for all cells, but each individual coefficient is more complex: contains complete coupling.
- Both choices have advantages and choice depends on software infrastructure and matrix assembly methods. In order to illustrate the nature of coupling, we shall choose per-cell organisation.

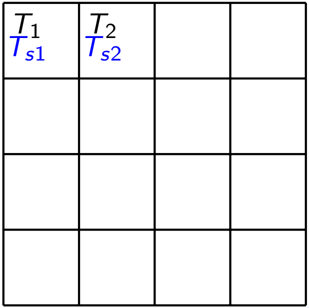
- Steady-state conjugate heat transfer between a porous medium and a fluid flowing through it. Assuming frozen flow field.
- (This corresponds to modelling of radiators in automotive under-hood simulations).
- **Fluid and solid domains are overlapping: same physical volume**
- Heat transfer in fluid with heat exchange to solid:

$$\nabla \bullet (\mathbf{u} T) - \nabla \bullet K (\nabla T) = \alpha (T_s - T)$$

- Heat transfer in solid with heat exchange to fluid:

$$-\nabla \bullet K_s (\nabla T_s) = \alpha (T - T_s)$$

- To simplify the picture, assume full overlap: both equations are solved on the complete mesh/domain.
- All cells share all field variables and allow coupling to.
  - Other variables in the same cells: **"source term coupling"**
  - Other variables in a neighbour cell: **"transport coupling"**
- ... but neighbourhood is still implied by the mesh: global sparseness pattern!



Trivial implementation: This is what FOAM was designed for!

```
fvScalarMatrix TEqn
(
    fvm::div(phi, T)
  - fvm::laplacian(DT, T)
 ==
    alpha*Ts - fvm::Sp(alpha, T)
);
TEqn.relax();
TEqn.solve();

fvScalarMatrix TsEqn
(
  - fvm::laplacian(DTs, Ts)
 ==
    alpha*T - fvm::Sp(alpha, Ts)
);
TsEqn.relax();
TsEqn.solve();
```

- The most important coupling is lagged: temperature fields T and T_s see each other in the source terms.
- Transport in each variable is handled implicitly.
- ...but the dominant term of T to T_s is lagged, delaying convergence!

Equation Coupling Idea: **Solve T and T_s in a Single Matrix**

- How to couple the variables implicitly: source term with temperature difference must be updated implicitly.
- A variable solved for is a pair of temperatures in each cell $\mathbf{x} = \begin{bmatrix} T \\ T_s \end{bmatrix}$.
- Matrix coefficients become tensors ... What does this look like?

Generalisation of Matrix Operations

- Global sparseness pattern is dependent on the mesh: matrix row contains an off-diagonal entry if two cells share a face.

Sparse Matrix of Dense Matrix Coefficients for Multi-Variable Block-Coupling.

$$\begin{bmatrix} \begin{bmatrix} a_{ff} & a_{fs} \\ a_{sf} & a_{ss} \end{bmatrix}_P & . & \begin{bmatrix} a_{ff} & a_{fs} \\ a_{sf} & a_{ss} \end{bmatrix}_U & \cdots \\ . & \begin{bmatrix} a_{ff} & a_{fs} \\ a_{sf} & a_{ss} \end{bmatrix}_P & . & \cdots \\ \begin{bmatrix} a_{ff} & a_{fs} \\ a_{sf} & a_{ss} \end{bmatrix}_L & . & \begin{bmatrix} a_{ff} & a_{fs} \\ a_{sf} & a_{ss} \end{bmatrix}_P & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \begin{bmatrix} T_1 \\ T_{s1} \end{bmatrix} \\ \begin{bmatrix} T_2 \\ T_{s2} \end{bmatrix} \\ \begin{bmatrix} T_3 \\ T_{s3} \end{bmatrix} \\ \vdots \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} b_1 \\ b_{s1} \end{bmatrix} \\ \begin{bmatrix} b_2 \\ b_{s2} \end{bmatrix} \\ \begin{bmatrix} b_3 \\ b_{s3} \end{bmatrix} \\ \vdots \end{bmatrix}$$

- Analysis: which equations are zero or non-zero?
- **Diagonal coefficient, off-diagonal entry**: source coupling.
- **Off-diagonal coefficient, off-diagonal entry**: cross-transport

Block-Coupled Solution Algorithm: Use matrices above, but do not call solve()

```
fvBlockMatrix<vector2> blockM(blockT);

// Insert equations into block Matrix
blockM.insertEquation(0, TEqn);
blockM.insertEquation(1, TsEqn);

// Add off-diagonal coupling terms
scalarField coupling(mesh.nCells(), -alpha.value());
blockM.insertEquationCoupling(0, 1, coupling);
blockM.insertEquationCoupling(1, 0, coupling);

// Update source coupling: coupling terms eliminated from
blockM.updateSourceCoupling();

blockM.solve();      // Block coupled solver call

// Retrieve solution
blockM.retrieveSolution(0, T.internalField());
blockM.retrieveSolution(1, Ts.internalField());
T.correctBoundaryConditions();
Ts.correctBoundaryConditions();
```

Coupling Coefficient Operators: Scalar Gradient (in a Vector Equation),

- **Couple Gauss gradient** (of a scalar), eg. ∇p

$$\int_{V_P} \nabla \phi \, dV = \oint_{\partial V_P} d\mathbf{s} \, \phi$$
$$= \sum_f \mathbf{s}_f \phi_f$$
$$\phi_f = f_x \phi_P + (1 - f_x) \phi_N$$

- $\mathbf{a}_N = f_x \, \mathbf{s}_f$ and $\mathbf{a}_P = -\sum_N \mathbf{a}_N$ are vector coefficient multiplying a scalar field
- **Coupled Gauss divergence** (of a vector), eg. $\nabla \bullet \mathbf{u} =$ transpose of a gradient operator
- Other block-coupled terms: work-in-progress
  - $\nabla \bullet \left[\gamma \left(\nabla \mathbf{u} \right)^T \right]$ for structural analysis solvers
  - $\mathbf{v} \bullet (\nabla \mathbf{u})^T$ for adjoint convection

Turbulent Steady Incompressible Flows: SIMPLE or Coupled.

- Equation set contains linear $p - \mathbf{u}$ and non-linear $\mathbf{u} - \mathbf{u}$ coupling.

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla_{\bullet}(\mathbf{u}\mathbf{u}) - \nabla_{\bullet}(\nu \nabla \mathbf{u}) = -\nabla p$$

$$\nabla_{\bullet}\mathbf{u} = 0$$

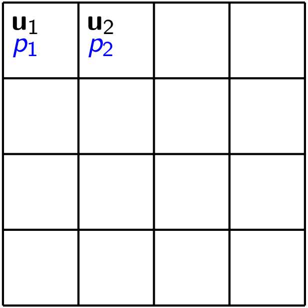
- Traditionally, this equation set is solved using the **segregated SIMPLE algorithm**.
  - Low memory peak: solution + single scalar matrix in peak storage.
  - p-U coupling is handled explicitly: loss of convergence (under-relaxation).
  - Number of iterations is substantial; not only due to non-linearity.
  - Convergence dependent on mesh size: SIMPLE slows down on large meshes.

- **Block-implicit $p - \mathbf{u}$ coupled solution**
  - Coupled solution significantly increases matrix size: 4 blocks instead of 1
  - ...but the linear $p - \mathbf{u}$ coupling is fully implicit!
  - Iteration sequence only needed to handle the non-linearity in the **u**-equation and the turbulence.
  - Net result: **significant convergence improvement** (steady or transient) at a cost of increase in memory usage: **reasonable performance compromise!**

SIMPLE-Based Segregated p-U Solver

```
// Momentum equation assembly and solution
fvVectorMatrix UEqn
(
    fvm::div(phi, U)
  + turbulence->divDevReff(U)
);
UEqn.relax();
solve(UEqn == -fvc::grad(p));

// Pressure equation assembly and solution
U = UEqn().H()/UEqn.A();
phi = fvc::interpolate(U) & mesh.Sf();
fvScalarMatrix pEqn
(
    fvm::laplacian(1/UEqn.A(), p) == fvc::div(phi)
);
pEqn.solve();
phi -= pEqn.flux();
p.relax();
```

Block-Coupled $\mathbf{u} - p$ System Matrix Structure



$$\begin{bmatrix} \begin{pmatrix} a_{P\,(\mathbf{u}\,\mathbf{u})} & a_{P\,(\mathbf{u}\,p)} \\ a_{P\,(p\,\mathbf{u})} & a_{P\,(p\,p)} \end{pmatrix} & \begin{pmatrix} a_{N\,(\mathbf{u}\,\mathbf{u})} & a_{N\,(\mathbf{u}\,p)} \\ a_{N\,(p\,\mathbf{u})} & a_{N\,(p\,p)} \end{pmatrix} & \cdots \\ . & \begin{pmatrix} a_{P\,(\mathbf{u}\,\mathbf{u})} & a_{P\,(\mathbf{u}\,p)} \\ a_{P\,(p\,\mathbf{u})} & a_{P\,(p\,p)} \end{pmatrix} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ p_1 \\ \mathbf{u}_2 \\ p_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} b_{\mathbf{u}\,1} \\ b_{p\,1} \\ b_{\mathbf{u}\,2} \\ b_{p\,2} \\ \vdots \end{bmatrix}$$

Coupled Implicit p-U Solver: pUCoupledFoam

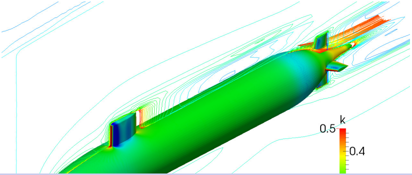
```
fvVectorMatrix UEqn
(
    fvm::div(phi, U)
  + turbulence->divDevReff(U)
);
fvScalarMatrix pEqn
(
  - fvm::laplacian(rUAf, p) == -fvc::div(fvc::grad(p))
);
BlockLduSystem<vector, vector> pInU(fvm::grad(p));
BlockLduSystem<vector, scalar> UInp(fvm::UDiv(U));

fvBlockMatrix<vector4> UpEqn(Up);
UpEqn.insertEquation(0, UEqn);
UpEqn.insertEquation(3, pEqn);
UpEqn.insertBlockCoupling(0, 3, pInU, true);
UpEqn.insertBlockCoupling(3, 0, UInp, false);

UpEqn.solve();
UpEqn.retrieveSolution(0, U.internalField());
UpEqn.retrieveSolution(3, p.internalField());
```

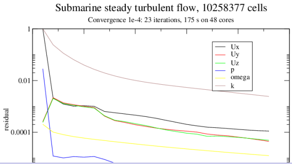
Open Source Implementation of the Coupled Implicit $p - \mathbf{u}$ Solver.

- Coupled solver reproduces the results of the segregated algorithm.
- Full integration with the `finiteVolume` library: re-use the existing and validated discretisation operators and boundary conditions without re-coding.

Coupled Solver Performance:

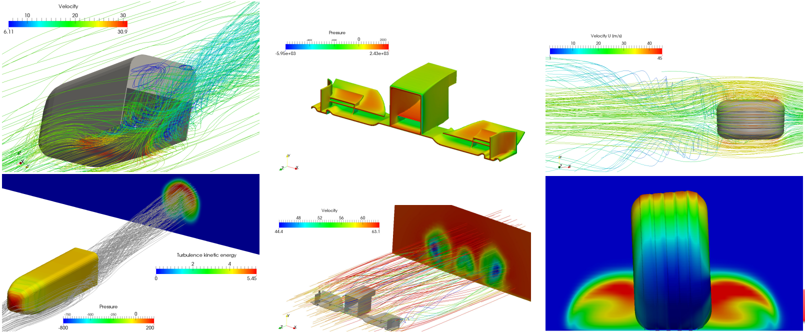
- Example: steady turbulent flow around a submarine, 10.2 million cells
- Turbulent $k - \omega$ SST model converges to 1e-4 in 23 iterations, 175 s on 48 cores





Who Needs a Coupled Solver?

- Improved speed-to-solution and reliability on difficult meshes.
- Robust discretisation: fewer settings.
- Improved parallel scaling: fewer processor communication calls.
- Already done: coupled turbulence model equations.



Block-Coupled Turbulence

Block-Coupled Turbulence Models: `coupledKEpsilon` and `coupledKOmegaSST`:

- Turbulence kinetic energy equation:

$$\frac{\partial k}{\partial t} + \nabla\bullet(\overline{\mathbf{u}} k) - \nabla\bullet(\nu_{eff} \nabla k) = G - \epsilon$$

$$G = \nu_t \left[\frac{1}{2}(\nabla\overline{\mathbf{u}} + \nabla\overline{\mathbf{u}}^T)\right]^2 = \mathbf{R} : \nabla U$$

- Dissipation of turbulence kinetic energy equation:

$$\frac{\partial \epsilon}{\partial t} + \nabla\bullet(\overline{\mathbf{u}}\epsilon) - \nabla\bullet((\nu_{eff})\nabla\epsilon] = C_1\, G\frac{\epsilon}{k} - C_2\frac{\epsilon^2}{k}$$

- Acknowledgment: Mr. Robert Keser (diploma thesis).

Summary

- **Multiple equations with strong coupling**: sparse-on-dense matrix class. Discretisation for "obvious" block-coupling; segregated discretisation is re-packed into block coefficients.

Block Matrix Tools in `foam-extend`:

- Dynamic coefficient expansion for general NxN coupling (used up to N=128),
- Support for coupled boundaries and parallelisation,
- Automated block matrix assembly and solution retrieval tools.

Coupled Solution for the $p - \mathbf{u}$ Equation Set:

- Implicit block-coupled `fvm::div(.)` and `fvm::grad(.)` operators, re-using existing FVM discretisation
- Efficient coupled $p - \mathbf{u}$ solver for steady-state simulations, excellent results!

Thanks for your attention.
Questions, please?