OpenFOAM: A C++ Library for Complex Physics Simulations

Hrvoje Jasak
h.jasak@wikki.co.uk

Wikki Ltd, United Kingdom
FSB, University of Zagreb, Croatia
Florence, 5th October 2007
Objective

- Present the developing functionality in OpenFOAM based on fundamental ideas of object orientation, layered software design and equation mimicking
- Review some recent development projects to encourage collaboration

Topics

1. Implementing complex physical models through equation mimicking
2. OpenFOAM: Object-oriented software for Computational Continuum Mechanics
3. Layered software development as a collaboration platform
4. Examples of collaborative development
   - Automatic mesh generation for complex geometries
   - Multi-region support and Fluid-Structure Interaction (FSI) solver
   - Turbomachinery simulations: General Grid Interface (GGI)
   - Internal combustion engine simulation: in-cylinder flows
5. Summary
Open Source Computational Continuum Mechanics

- Commercial CFD dominates the landscape: a complete code with sufficient efficiency, parallelism, mesh handling and pre- and post-utilities is a large project
- Proprietary implementation is the limiting factor: closed software architecture
- ...but does not satisfy all needs: especially research in numerics, physical modelling and collaborative development
- Complete CFD methodology is already in the public domain (research papers, model formulation, numerical schemes, linear equation solvers etc.)
- Objective: open source implementation of existing knowledge and an object-oriented platform for easy future development
  1. Completely open software platform using object-oriented design
  2. Extensive modelling capabilities in library form: component re-use
  3. Fast, robust and accurate numerical solver
  4. State of the art complex geometry handling
  5. Collaborative and project-driven model development
- ...but the mode of operation changes considerably
Flexible Handling of Arbitrary Equations Sets

- Natural language of continuum mechanics: partial differential equations
- Example: turbulence kinetic energy equation

\[
\frac{\partial k}{\partial t} + \nabla \cdot (u k) - \nabla \cdot [(\nu + \nu_t) \nabla k] = \nu_t \left[ \frac{1}{2} (\nabla u + \nabla u^T) \right]^2 - \frac{\epsilon_o}{k_o} k
\]

- Objective: Represent differential equations in their natural language

```c++
solve
{
    fvm::ddt(k)
    + fvm::div(phi, k)
    - fvm::laplacian(nu() + nut, k)
    == nut*magSqr(symm(fvc::grad(U)))
    - fvm::Sp(epsilon/k, k)
};
```

- Correspondence between the implementation and the original equation is clear
Object Orientation

Object-Oriented Software: Create a Language Suitable for the Problem

- Analysis of numerical simulation software through object orientation: “Recognise main objects from the numerical modelling viewpoint”
- Objects consist of data they encapsulate and functions which operate on the data

Example: Sparse Matrix Class

- **Data members**: protected and managed
  - Sparse addressing pattern (CR format, arrow format)
  - Diagonal coefficients, off-diagonal coefficients
- Operations on matrices or data members: **Public interface**
  - Matrix algebra operations: +, −, *, /,
  - Matrix-vector product, transpose, triple product, under-relaxation
- Actual data layout and functionality is important only internally: efficiency

Example: Linear Equation Solver

- Operate on a system of linear equations $Ax = b$ to obtain $x$
- It is irrelevant how the matrix was assembled or what shall be done with solution
- Ultimately, even the solver algorithm is not of interest: all we want is new $x$!
- Gauss-Seidel, AMG, direct solver: all answer to the same interface
Object Orientation

Basic Components
- Scalars, vectors and tensors with algebra
- Computational mesh; mesh motion, adaptive refinement, topological changes
- Fields (scalar, vector, tensor) and boundary conditions: Dirichlet, Neumann etc.
- Sparse matrix support with linear solver technology

Discretisation Classes
- Implemented as interpolation, differentiation and discretisation operators
- All discretisation methods use identical basic components, e.g. common mesh and matrix support. Better testing and more compact software implementation

Physical Modelling Libraries and Top-Level Solvers
- Libraries encapsulate interchangeable models answering to a common interfaces
- Models implement the interface functions, isolated with run-time selection
- Custom-written and optimised top-level solvers for class of physics

Utilities
- Common functionality is needed for most simulation methods
- Example: problem setup, mesh manipulation, data analysis, parallel execution
Common Interface for Model Classes

- Physical models grouped by functionality, e.g. material properties, viscosity models, turbulence models etc.
- Each model answers the interface of its class, but its implementation is separate and independent of other models
- The rest of software handles the model through generic interface: breaking the complexity of the interaction matrix

```cpp
class turbulenceModel
{
  virtual volTensorField R() const = 0;
  virtual fvVectorMatrix divR(
    volVectorField& U
  ) const = 0;
  virtual void correct() = 0;
};
```

- New turbulence model implementation: Spalart-Allmaras 1-equation model

```cpp
class SpalartAllmaras : public turbulenceModel
{
  // Model implementation lives here
};
```
Model-to-Model Interaction

Common Interface for Model Classes

- Model user only sees the virtual base class
- Example: steady-state momentum equation with turbulence

```cpp
autoPtr<turbulenceModel> turbulence
(  
turbulenceModel::New(U, phi, laminarTransport)
);

fvVectorMatrix UEqn
(
    fvm::ddt(rho, U)
    + fvm::div(phi, U)
    + turbulence->divR(U)
    ==
    - fvc::grad(p)
);
UEqn.solve();
```

- Implementation of a new model does not disturb existing models
- Consumer classes see no changes whatsoever, just a new model choice
Application Development in OpenFOAM

- Custom-written top-level solvers are written for each class of physics
- Solvers are optimised for efficiency and storage, re-using basic components
- Writing top-level code is very similar to manipulating the equations
- **Ultimate user-coding capabilities**: components can be re-used to handle most problems in computational continuum mechanics

Layered Development

- Design encourages code re-use: developing shared tools
- Classes and functional components developed and tested in isolation
  - Vectors, tensors and field algebra
  - Mesh handling, refinement, mesh motion, topological changes
  - Discretisation, boundary conditions
  - Matrices and linear solver technology
  - Physics by segment in library form
- Library level mesh, pre-, post- and data handling utilities
- Model-to-model interaction handled through common interfaces
- New components do not disturb existing code: fewer bugs
Automatic Mesh Generation

Automatic Meshing for Complex Geometry

- Mesh generation for complex geometry currently demands extensive user-interaction
- The FVM method is not limited to prescribed cell shape: polyhedral support
- Fewer cells per volume, minimal distortion and non-orthogonality, near-wall layers, richer cell-to-cell connectivity, flow alignment, resolution control, error-driven adaptivity
- Primarily, **reliable automatic meshing**
Conjugate Heat Transfer Problems

- OpenFOAM supports multi-region simulations, with possibility of separate addressing and physics for each mesh: multiple meshes, with local fields
- Some equations present only locally, while others span multiple meshes

```c++
coupledFvScalarMatrix TEqns(2);

TEqns.hook
(
   fvm::ddt(T) + fvm::div(phi, T)
   - fvm::laplacian(DT, T)
);

TEqns.hook
(
   fvm::ddt(Tsolid) - fvm::laplacian(DTsolid, Tsolid)
);

TEqns.solve();
```

- Coupled solver handles multiple matrices together in internal solver sweeps
Example: Conjugate Heat Transfer

- Coupling may be established geometrically: adjacent surface pairs
- Each variable is stored only on a mesh where it is active: (U, p, T)
- Choice of conjugate variables is completely arbitrary: e.g. catalytic reactions
- Coupling is established only per-variable: handling a general coupled complex physics problem rather than conjugate heat transfer problem specifically
Solution Techniques for Coupled Problems

- **Partitioned approach**: Picard iterations
  - Optimal for weakly coupled FSI problems
  - Separate mathematical model for fluid and solid continua
  - Shared or different discretisation method: FVM and FEM
  - Coupling achieved by enforcing the kinematic and dynamic conditions on the fluid-solid interface
  - Strong coupling by additional iteration loop over partial solvers (need a convergence acceleration method)

- **Monolithic approach**: Simultaneous solution
  - Appropriate when fluid-structure interaction is very strong
  - Good stability and convergence properties
  - In some cases may lead to ill-conditioned matrices or sub-optimal discretisation or solution procedure in fluid or solid region

Levels of Fluid-Structure Coupling

- Unified mathematical model: single equation set for fluid and structure
- Unified discretisation method and boundary coupling consistency
- Unified solution procedure: fluid + structure matrix solved in a single solver
Example: Fluid-Structure Interaction

Updated Lagrangian FVM Solver

- Incremental momentum equation in updated Lagrangian formulation

\[
\int_{V_u} \rho_u \frac{\partial \delta \mathbf{v}}{\partial t} \mathbf{v}_u = \int_{S_u} \mathbf{n}_u \cdot \left( \delta \Sigma_u + \Sigma_u \cdot \delta \mathbf{F}_u^T + \delta \Sigma_u \cdot \delta \mathbf{F}_u^T \right) \mathbf{S}_u
\]

- Incremental constitutive equation for St. Venant-Kirchhoff elastic solid

\[
\delta \Sigma_u = 2\mu \delta \mathbf{E}_u + \lambda \text{tr} (\delta \mathbf{E}_u) \mathbf{I}
\]

- Increment of Green-Lagrange strain tensor

\[
\delta \mathbf{E}_u = \frac{1}{2} \left[ \nabla \delta \mathbf{u} + (\nabla \delta \mathbf{u})^T + \nabla \delta \mathbf{u} \cdot (\nabla \delta \mathbf{u})^T \right]
\]

- Final form of momentum equation ready for discretisation

\[
\int_{V_u} \rho_u \frac{\partial \delta \mathbf{v}}{\partial t} \mathbf{v}_u - \int_{S_u} \mathbf{n}_u \cdot (2\mu + \lambda) \nabla \delta \mathbf{u} \mathbf{S}_u = \int_{S_u} \mathbf{n}_u \cdot \mathbf{q} \mathbf{S}_u
\]
Example: Fluid-Structure Interaction

ALE FVM Fluid Flow Solver

- Continuity and momentum equation for incompressible flow in ALE formulation
- Space conservation law; automatic mesh motion solver: Laplace equation
- Collocated 2nd order FVM for flow and 2nd order FEM for mesh motion

Solution Procedure

1. Update mesh according to displacement increment from the previous time step
2. Update second Piola-Kirchhoff stress tensor
3. Do until convergence
   - Calculated RHS of discretised momentum equation using last calculated displacement increment field
   - Solve momentum equation
4. Accumulate displacement vector and second Piola-Kirchhoff stress fields
5. On convergence, switch to the next time step

Data Transfer

- Data transfer and coupling significantly easier: both domains and equations sets are implemented in the same solver
- Data interpolation routines already available: patch-to-patch interpolation
Cantilevered Beam Vibration in an Oscillating Flow Field

Effect of Under-Relaxation

<table>
<thead>
<tr>
<th>( \rho_s/\rho_f )</th>
<th>Number of outer iterations per time-step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed under-relaxation</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>60</td>
</tr>
</tbody>
</table>

Example: Fluid-Structure Interaction
Turbomachinery Simulations

General Grid Interface

- Sliding interface mesh modifier couples two mesh components in relative motion to create a continuous mesh: topological change
- In turbo-machinery simulations, similar coupling problem appears: sliding rotor-stator interface, non-matching cyclics
- Treatment of coupled boundaries topology change: **General Grid Interfaces**
  - Coupled path treatment implicit in discretisation and solvers
  - Coupling addressing calculated using patch-to-patch interpolation
  - Special handling for patch evaluation and operator discretisation
  - Special turbo-specific steady-state discretisation: **mixing plane**
Summary

Project Status Summary

- OpenFOAM is a free software, available to all at no charge: GNU Public License
- Object-oriented approach facilitates model implementation
- Equation mimicking opens new grounds in Computational Continuum Mechanics
- Extensive capabilities already implemented
- Open design for easy user customisation
- Solvers are validated in detail and match efficiency of commercial codes
- Number of contributors is rapidly increasing: notable out-of-core developments and new physical models

New Capabilities in Development Pipeline

- Automatic mesh generation for complex geometry
- In-cylinder simulations for internal combustion engines: Gasolene Direct Injection (GDI) with piston bowl penetration and extreme valve cant
- Suite of multi-region multi-physics (FSI) solvers in a single software
- Turbomachinery-specific features developed by the Turbo Working Group