**Title of the training course:**
Programming CFD in OpenFOAM

**Course description:**
Advanced course on programming in OpenFOAM with C++. Starting from a brief review of the C++ syntax, the numerical algorithms required to discretize and solve the Navier-Stokes equations will be presented with a particular emphasis on the implementation details by referring to the OpenFOAM source code. Each module will be followed by a hands-on where the learner can familiarize himself with the OpenFOAM source code.

**Language:** English

**Date:** 26-28 November 2025

**Training objectives:**
Improve the programming skills in OpenFOAM and learn more about the code architecture, data structure and algorithm implementation details. An emphasis will be placed on the discretization of the main differential operators (e.g. div, grad, laplacian …) and on the solution process of a linear system. The learner will be able at the end of the course to add a new implementation or modify the existing one of a differential operator or a solver.

**Target audience:**
Developers, CFD engineers & researchers, PhD students (in Engineering).

**Prerequisites and content levels**
Advanced experience with C/C++ and OpenFOAM. No GPU programming knowledge is required.

**Duration:** three days

**Trainer(s)/Instructor(s):** Simone Bnà (CINECA), Alessio Piccolo (CINECA), Giuseppe Giaquinto (CINECA), Tommaso Zanelli (CINECA)

**Course organizer contacts:**
Francesca Gebbia, Simone Bnà

**Official course URL(s) and subscriptions**:
https://eventi.cineca.it/en/hpc/programming-cfd-openfoam/bologna-20251126

**Agenda**

Day 1

- **09:30 – 10.00 HPC introduction (TZ)**: ssh connection, vscode configuration (vi/nano for advanced user), module load, login on reservation nodes

**Hands-on**: connection to the cluster, run vscode or check installation (locally)

- **10:00 – 11:15 Getting started with C++ (GG)**: creating a C++ program, compilation (wmake), macro, function overloading, scope, namespace, header files, class, inheritance, alias, template programming, factory method pattern

Break 15 min

- **11:30– 12:30 Core classes (container) (GG)**: introduction, primitives, List, Field, ptrList, FieldField, operations on Fields (reduce, sum, average, scan (few notes on parallel aspects))

- **12:30– 13:00 Hands-on:** AXPY implementation

13:00-14:30 Lunch

- **14:30 - 15:15 Memory management (GG)**: dynamic memory management and smart pointers: autoPtr and tmp

- **15:15 – 16:00 Case initialization (AP)**: setRootCaseList, create Time, I/O registry

Break 15 min

- **16:15 - 17:15 Data construction (AP)**: fvMesh, Dimensioned Field, GeometricField, boundaryField, fvPatchFields

- **17.15 - 18:00 Interfaces (AP)**: Interfacing with derived classes, virtual constructors, run-time selection

- **18.00 - 18:30 Hands-on**: boundary condition implementation with run-time selection.

---

Day 2

- **10:00 - 11:15 Data access (TZ)**: data access in loops (face addressing, loop over faces, loop over cells)

Break 15 min

- **11:30 - 12:30 FV equations (TZ)**: lduMatrix, fvMatrix

- **12:30 - 13:00 Hands-on**: Frobenius norm of a matrix, visualization of the sparsity pattern of a fvMatrix

13:00-14:30 Lunch

- **14:30 – 16:00 Derivatives and algebra (TZ):** fvm, fvc, Laplace operator, Div operator, Time-derivative operator, Grad operator

Break 15 min

- **16:15 – 18:00 Hands-on**: write a custom grad operator using loop over faces instead of a loop over cells

---

Day 3

- **09:30 – 10:45 Solution algorithms for incompressible Navier-Stokes (SB)**: Navier-Stokes, segregated vs coupled: splitting U-p, PISO, SIMPLE, PIMPLE

Break 15 min

- **11:00 - 12:30 Solvers (SB)**
    - **Solvers (I)**: lduMatrix, lduMatrix::solver, Amul (few notes on parallel aspects), PCG, preconditioning

    - **Solvers (II)**: csrMatrix, ldu2csr conversion, AmgX4Foam and PETSc4Foam design aspects

12:30:14:00 lunch break

- **14:00 - 15:00 OpenFOAM GPU porting (SB):** OpenFOAM design limitations, SPUMA design concepts
- **15:00 - 18:00 Hands-on**: convert a pressure matrix (Laplace Operator) from LDU to CSR; write your own PCG solver using a user-defined CSR-based Amul and custom grad Scheme.