NeoFOAM - A Collaborative Effort for a Modern OpenFOAM

The NeoFOAM Authors

Jan 2025

Motivation

Considerations for a Software Lifecycle

OpenFOAM is one of the most significant and widely utilized open-source Computational Fluid Dynamics (CFD) tools in both academic and industrial domains. Nonetheless, the foundational design decisions that shaped OpenFOAM were established quite some time ago. Meanwhile, software development methodologies have undergone significant transformations. This includes adopting more distributed and community-focused development models and incorporating automated workflows and modular software design. Established software libraries frequently encounter the dilemma of determining whether to integrate evolutionary updates or to implement a revolutionary advancement. The latter is generally imperative when integrating recent advancements proves to be prohibitive in terms of cost. The authors believe that OpenFOAM requires a revolutionary design overhaul for reasons that are detailed in this manuscript.

Changes in the (HPC) Hardware Landscape

Besides the aforementioned changes in software development practices, the high-performance computing (HPC) hardware ecosystem has undergone substantial transformations in recent years. This shift has been characterized by a transition from a dominating presence of x86-based CPUs supplied by a limited number of vendors to a scenario where graphics processing units (GPUs) constitute the predominant source of computational power in modern HPC clusters. Moreover, the landscape is now marked by intensified competition among multiple vendors striving for market share.

Developments in Research Software Engineering

- collaborative platforms like GitHub and GitLab allow a more distributed and collaborative development

- these platforms also provide automated workflows to ensure good coding practices and testing

The NeoFOAM Architecture

In the following the key aspects of the NeoFOAM architecture are discussed.

The Executor Class

NeoFOAM uses the MPI+X approach for parallelism, where X is the execution space used for device parallelism. The Executor class uses Kokkos [@Trott2022], provides an interface for memory management, and specifies where to execute the operations:

- SerialExecutor: run on the CPU with MPI
- CPUExecutor: run on the CPU with either OpenMP or C++ Threads in Combination and MPI
- GPUExecutor: run on the GPU with MPI

The Field Class

The Field class is the basic container class and is the central elements for implementing a platform portable CFD framework. Fields should allow to perform basic algebraic operations such as binary operations like the addition or subtraction of two fields, or scalar operations like the multiplication of a field with a scalar.

Domain Specific Language (DSL)

The concept of a Domain Specific Language (DSL) allows to simplify the process of implementing and solving equations in a given programming language like C++. Engineers can express equations in a concise and readable form, closely resembling their mathematical representation, while no or little knowledge of the numerical schemes and implementation is required. This approach allows engineers to focus on the physics of the problem rather than the numerical implementation and helps in reducing the time and effort required to develop and maintain complex simulations.

This approach is readable and easy to understand for engineers familiar with OpenFOAM. However, it has several limitations due to its implementation in OpenFOAM:

- the solution system is always a sparse matrix
- individual terms are eagerly evaluated, resulting in unnecessary use of temporaries
- the sparse matrix is always an LDU matrix that is typicalle not supported by third-party linear solver libraries

- Only cell-centred discretisation is supported

NeoFOAM DSL tries to address these issues by providing:

- lazy evaluation of the equations terms. This allows for better optimisation of the resulting equation and can reduce the number of required temporaries.
- a more modular design
- Support for standard matrix formats like COO and CSR, to simplify the use of external linear solvers

Future work

LibTooling based OpenFOAM refactoring

We are aiming for a high level of compatibility with OpenFOAM. However, we don't expect binary or ABI compatibility. This means that NeoFOAM won't produce a libfiniteVolume.so and libOpenFOAM.so which could serve as a plugin replacement for existing libfiniteVolume.so and libOpenFOAM.so. Instead, we aim for source compatibility, i.e. the possibility to compile application OpenFOAM code like pimpleFoam and others against the NeoFOAM libraries. This approach is demonstrated in the FoamAdapter repository. However, this means that a substantial part of the existing OpenFOAM code based would need to be refactored, for example by changing from
the fvScalarField and fvVectorField to the corresponding NeoFOAM::VolumeField classes, among others.
Because of the size of the OpenFOAM code base this should ideally be performed by an automated approach, such as via Clangs libTooling [https://clang.llvm.org/docs/LibTooling.html].

Package Manager

OpenFOAM is used widely and many third-party projects exists. However no central repository collecting these packages exists. Adding a package manager for NeoFOAM would allow to focus development on the core aspects of NeoFOAM and would simplify the addition of functionality by the community. Similar approaches exists typically for programming languages for example - pip - cargo - dockerhub

References