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

- Kokkos [@Trott_Kokkos_3_Programming_2022] based approach

Executor

NeoFOAM uses the MPI+X approach for parallelism, where X is the execution space used for device parallelism. The Executor class uses Kokkos, 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

Field

DSL

Future work

LibTooling based OpenFOAM refactoring

Package Manager