The Journal of Open Source Software

XCALibre.jl: xxx

Humberto Medina[1¶], Chris Ellis[1], Tom Mazin[1], Oscar Osborne[1], Timothy Ward[1], Stephen Ambrose[1], Svetlana Aleksandrova[2], Benjamin Rothwell[1], and Carol Eastwick[1]

1 The University of Nottingham, UK **2** The University of Leicester, UK ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Understanding the behaviour of fluid flow, such as air over a wing, water in a pipeline, or fuel in an engine is crucial in many engineering applications, from designing aircraft and automotive components to optimising energy systems, etc. Computational Fluid Dynamics (CFD) enables engineers to model real-world conditions, optimise designs, and predict performance under a wide range of scenarios, and it has become a vital part of the modern engineering design process for creating efficient, safe, and sustainable designs. As engineers seek to develop and optimise new designs, particularly in fields where there is a drive to push the current state-of-the-art or physical limits of existing design solutions, often, new CFD methodologies or physical models are required. Therefore, extendable and flexible CFD frameworks are needed, for example, to allow seamless integration with machine learning models. In this paper, the features of the first release of the Julia package XCALibre.jl are presented. Designed with extensibility in mind, XCALibre.jl is aiming to facilitate the rapid prototyping of new fluid models and to easily integrate with Julia's powerful ecosystem, enabling access to optimisation libraries and machine learning frameworks to enhance its functionality and expand its application potential, whilst offering multi-threaded performance CPUs and GPU acceleration.

Statement of need

Given the importance of fluid flow simulation in engineering applications, it is not surprising that there is a wealth of CFD solvers available, both open-source and commercially available. Well established open-source codes include: OpenFOAM, SU2, CODE_SATURN, Gerris, etc. It is a testament to the open-source philosophy, and their developers, that some of these codes offer almost feature parity with commercial codes. However, the more feature-rich open-source codes have large codebases and, for performance reasons, have been implemented in statically compiled languages which makes it difficult to adapt and incorporate recent trends in scientific computing, for example, GPU computing and interfacing with machine learning frameworks, which is also the case for commercial codes (to a larger extent due to their closed source nature where interfaces to code internals can be quite rigid – although thanks to access to more resources commercial codes have been steadily ported to work on GPUs). As a result, the research community has been actively developing new CFD codes, which is evident within the Julia ecosystem. The Julia programming language offers a fresh approach to scientific computing, with the benefits of dynamism whilst retaining the performance of statically typed languages thanks to its just-in-time compilation approach (using LLVM compiler technology). Thus, Julia makes it easy to prototype and test new ideas whilst producing machine code that is performant. This simplicity-performance dualism has resulted in a remarkable growth in its ecosystem offering for scientific computing, which includes state-of-the-art packages for solving differential equations (`DifferentialEquations.jl`), building machine learning models (`Flux.jl`, `Knet.jl` and `Lux.jl`), optimisation frameworks (`JUMP.jl`, XXX and XXX, and

more), automatic differentiation (), etc. Likewise, CFD packages have also been developed, most notoriously: `Oceananigans.jl`, which provides tools for ocean modelling, `Trixi.jl` which provides high-order for solvers using the Discontinuous Garlekin method, and `Waterlilly.jl` which implements the immerse boundary method on structured grids using a staggered finite volume method. In this context, `XCALibre.jl` aims to complement and extend the Julia ecosystem by providing a cell-centred and unstructured finite volume CFD framework for the simulation of both incompressible and weakly compressible flows.

`Gala` is an Astropy-affiliated Python package for galactic dynamics. Python enables wrapping low-level languages (e.g., C) for speed without losing flexibility or ease-of-use in the user-interface. The API for `Gala` was designed to provide a class-based and user-friendly interface to fast (C or Cython-optimized) implementations of common operations such as gravitational potential and force evaluation, orbit integration, dynamical transformations, and chaos indicators for nonlinear dynamics. `Gala` also relies heavily on and interfaces well with the implementations of physical units and astronomical coordinate systems in the `Astropy` package (Astropy Collaboration, 2013) (`astropy.units` and `astropy.coordinates`).

`Gala` was designed to be used by both astronomical researchers and by students in courses on gravitational dynamics or astronomy. It has already been used in a number of scientific publications (Pearson et al., 2017) and has also been used in graduate courses on Galactic dynamics to, e.g., provide interactive visualizations of textbook material (Binney & Tremaine, 2008). The combination of speed, design, and support for Astropy functionality in `Gala` will enable exciting scientific explorations of forthcoming data releases from the *Gaia* mission (Gaia Collaboration, 2016) by students and experts alike.

Mathematics

Single dollars ($) are required for inline mathematics e.g. $f(x) = e^{\pi/x}$

Double dollars make self-standing equations:

$$\Theta(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ else} \end{cases}$$

You can also use plain LaTeXfor equations

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x) e^{i\omega x} dx \tag{1}$$

and refer to Equation 1 from text.

Citations

Citations to entries in paper.bib should be in rMarkdown format.

If you want to cite a software repository URL (e.g. something on GitHub without a preferred citation) then you can do it with the example BibTeX entry below for Smith et al. (2020).

For a quick reference, the following citation commands can be used: - `@author:2001` -> "Author et al. (2001)" - `[@author:2001]` -> "(Author et al., 2001)" - `[@author1:2001; @author2:2001]` -> "(Author1 et al., 2001; Author2 et al., 2002)"

Figures

Figures can be included like this: Caption for example figure. and referenced from text using section .

Figure sizes can be customized by adding an optional second parameter: Caption for example figure.

Acknowledgements

We acknowledge contributions from Brigitta Sipocz, Syrtis Major, and Semyeong Oh, and support from Kathryn Johnston during the genesis of this project.

References

Astropy Collaboration. (2013). Astropy: A community Python package for astronomy. *Astronomy and Astrophysics*, *558*. https://doi.org/10.1051/0004-6361/201322068

Binney, J., & Tremaine, S. (2008). *Galactic Dynamics: Second Edition*. Princeton University Press. http://adsabs.harvard.edu/abs/2008gady.book.....B

Gaia Collaboration. (2016). The Gaia mission. *Astronomy and Astrophysics*, *595*. https://doi.org/10.1051/0004-6361/201629272

Pearson, S., Price-Whelan, A. M., & Johnston, K. V. (2017). Gaps in Globular Cluster Streams: Pal 5 and the Galactic Bar. *ArXiv e-Prints*. http://adsabs.harvard.edu/abs/2017arXiv170304627P

Smith, A. M., Thaney, K., & Hahnel, M. (2020). Fidgit: An ungodly union of GitHub and figshare. In *GitHub repository*. GitHub. https://github.com/arfon/fidgit