


















# Singularity-EOS: Performance Portable Equations of State and Mixed Cell Closures

Jonah M. Miller <sup>1,2</sup>✉, Daniel A. Holladay <sup>2,3</sup>, Jeffrey H. Peterson <sup>4</sup>, Christopher M. Mauney <sup>2,5</sup>, Richard Berger <sup>3</sup>, Anna Pietarila Graham<sup>5</sup>, Karen C. Tsai <sup>3</sup>, Brandon Barker <sup>1,2,6,7</sup>, Alexander Holas <sup>2,3,8</sup>, Ann E. Mattsson <sup>9</sup>, Mariam Gogilashvili <sup>1,2,7,10</sup>, Joshua C. Dolence <sup>1,2</sup>, Chad D. Meyer <sup>11</sup>, Sriram Swaminarayan <sup>3</sup>, and Christoph Junghans <sup>3</sup>

1 CCS-2, Computational Physics and Methods, Los Alamos National Laboratory, USA 2 Center for Theoretical Astrophysics, Los Alamos National Laboratory, Los Alamos, NM 3 CCS-7, Applied Computer Science, Los Alamos National Laboratory, USA 4 XCP-2, Eulerian Codes, Los Alamos National Laboratory, USA 5 HPC-ENV, HPC Environments, Los Alamos National Laboratory, USA 6 Department of Physics and Astronomy, Michigan State University, USA 7 Center for Nonlinear Studies, Los Alamos National Laboratory, USA 8 Heidelberg Institute for Theoretical Studies, Germany 9 XCP-5, Materials and Physical Data, Los Alamos National Laboratory, USA 10 Department of Physics, Florida State University, USA 11 XCP-4, Continuum Models and Numerical Methods, Los Alamos National Laboratory, USA ✉ Corresponding author

DOI: [10.21105/joss.06805](https://doi.org/10.21105/joss.06805)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Kyle Niemeyer](#) 

## Reviewers:

- [@parikshitbajpai](#)
- [@guadabsb15](#)

Submitted: 19 April 2024

Published: 07 November 2024

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

## Summary

We present Singularity-EOS, a new performance-portable library for equations of state and related capabilities. Singularity-EOS provides a large set of analytic equations of state, such as the Gruneisen equation of state, and tabulated equation of state data under a unified interface. It also provides support capabilities around these equations of state, such as Python wrappers, solvers for finding pressure-temperature equilibrium between multiple equations of state, and a unique *modifier* framework, allowing the user to transform a base equation of state, for example by shifting or scaling the specific internal energy. All capabilities are performance portable, meaning they compile and run on both CPU and GPU for a wide variety of architectures.

## Statement of need and State of the Field

When expressed mathematically for continuous materials, the laws of conservation of mass, energy, and momentum form the Navier-Stokes equations of fluid dynamics. In the limit of zero molecular viscosity, they become the Euler equations. These laws have been used to describe phenomena as disparate as flow of air over an airplane wing, bacterial motion in fluids, and the cataclysmic deaths of stars. However, the fluid equations are not complete, and the system must be *closed* by a description of the material at a sub-continuum (e.g., molecular or atomic) scale. This closure is commonly called the *equation of state* (EOS).

Equations of state vary from the simple ideal gas law, to sophisticated multi-phase descriptions of the lattice structure of ice or wood, to models of quark-gluon plasma and nuclear pasta at ultra high densities. A common form to write an equation of state is as a pair of relations:

$$P = P(\rho, T, \vec{\lambda}) \text{ and } \varepsilon = \varepsilon(\rho, T, \vec{\lambda}),$$

which relate the pressure  $P$  and specific internal energy  $\varepsilon$  to density  $\rho$ , temperature  $T$ , and potentially some unknown set of additional quantities  $\vec{\lambda}$ . However, other representations are

possible, and in common parlance an EOS is the collection of knowledge needed to reconstruct some intrinsic thermodynamic quantities from others. For example, the speed of sound through a material or the specific heat capacity, which are thermodynamic derivatives of the pressure and the specific internal energy respectively, can both be determined by the EOS.

In multi-material fluid dynamics simulations, one often will end up with a so-called *mixed cell*, where two materials exist within the same simulation zone. This can be an artifact of the numerical representation; for example a steel bar and the surrounding air may end up sharing a finite volume cell if the boundaries of the cell do not align exactly with the surface of the steel bar, or, it may represent physical reality; for example, air is a mixture of nitrogen and oxygen gases, as well as water vapor. Regardless of the nature of the mixed cell, one must somehow provide, to the fluid code, what the material properties of the cell are as a whole. This is called a *mixed cell closure*. One such closure is *pressure-temperature equilibrium* (PTE), where all materials in the cell are assumed to be at the same pressure and temperature.

Typically fluid dynamics codes each develop an EOS package individually to meet a given problem's needs. Databases of tabulated equations of state, such as the Sesame (Lyon & Johnson, 1992) and Stellar Collapse (O'Connor & Ott, 2010b) databases often come with tabulated data readers, for example, the EOSPAC library (Pimentel, 2021) and Stellar Collapse library (O'Connor & Ott, 2010a). However, these libraries typically do not include analytic equations of state or provide a unified API. They also don't provide extra equation-of-state capabilities, such as equilibrium solvers or production hardening. With a few exceptions, these libraries are also typically not GPU-capable.

We present Singularity-EOS, which aims to be a "one stop shop" for EOS models for fluid and continuum dynamics codes. It provides a unified interface for both analytic and tabulated equations of state. It also provides useful surrounding capabilities, such as Python wrappers, *modifiers*, which allow the user to transform a given EOS, and solvers which can find the state in which multiple EOS's are in PTE. To support usability, the library is extensively documented and tested and supports builds through both `cmake` and `Spack` (Gamblin et al., 2015).

Singularity-EOS leverages the Kokkos (Trott et al., 2022) library for performance portability, meaning the code can run on both CPUs and GPUs, as well as other accelerators. This fills an important need, as modern super computing capabilities increasingly rely on GPUs for performance. Singularity-EOS is now used in the ongoing open-source *Phoebus* project which has a separate code paper in-preparation.

## Design Principles and Feature Highlights

Here we enumerate several design principles underlying Singularity-EOS, and highlight a few feature of the library.

### Flexibility in loop patterns

Singularity-EOS provides both scalar and vector APIs, allowing the user to make EOS calls on both single points in thermodynamic space, and on collections of points. The vector calls may be more performant (as they may vectorize), however care is made to ensure both APIs operate at acceptable performance, to accommodate different code structures downstream.

### Flexibility in memory layout

The vector calls in Singularity-EOS use an *accessor* API and (with a few exceptions) accept any C++ object that has a `operator[]` function defined. This allows users to lay out their memory as they see fit and use Singularity-EOS even on strided or sparsely allocated memory.

## Expose APIs to aid performance

Many equations of state are most naturally represented as functions of density and temperature. However, fluid codes require pressure as a function of density and internal energy. Extracting this often requires computing a root to invert the relation

$$\varepsilon = \varepsilon(\rho, T).$$

In these cases, we expose an initial guess for temperature, which helps the solution rapidly converge. Similarly, the performance of a sequence of EOS calls may depend on the ordering of the calls. For example, if both temperature and pressure are required from an equation of state that requires inversion, requesting pressure first will be less performant than requesting temperature first, as the former requires two root finds, and the latter requires only one. To enable this, we expose a function `FillEos`, in which the user may request multiple quantities at once, and the code uses ordering knowledge to compute them as performantly as possible.

## Performance-portable polymorphism

Accelerators present new challenges to standard object-oriented programming. In particular, not all compiler stacks (such as Sycl (Reyes et al., 2020) or OpenMP (Chandra et al., 2001) Target Offload (OpenMP Architecture Review Board, 2013)) support relocatable device code, which is required for standard C++ polymorphism. Even in programming models, such as CUDA (NVIDIA et al., 2020), which do support relocatable device code, polymorphism can be slower than naively expected, and the user-level API can be cumbersome, requiring operations such as `placement new`. To sidestep these issues, we use the C++ language feature `std::variant` to implement a polymorphism mechanism that works on device.

## Modifiers

A given code may need to modify an EOS model to make it suitable for a given application. For example, the zero-point of the energy may need to be shifted, a porosity model may need to be added, or the unit system may need to be changed. We implement this with a system of *modifiers*, which can be applied on top of an EOS in a generic way. Modifiers may also be chained.

## Fast log-lookups

To span the required orders of magnitude, equations of state are often tabulated on log-spaced grids. Logarithms and exponentials are, however, expensive operations and the performance of lookups can suffer. We instead use the not-quite-transcendental lookups described in (Miller et al., 2022) to significantly enhance performance of log-like lookups.

## Extensibility via modular parts and plugins

Singularity-EOS is designed to be extensible. The `std::variant`-based polymorphism, combined with modifiers, as described above, already provides significant flexibility. However, downstream codes may wish to add functionality to the library. This may be implemented in several ways. **First**, as Singularity-EOS is open source, contributions from downstream developers are welcome. **Second**, a C++ code that depends on Singularity-EOS may implement their own models and include them in a local variant object. Singularity-EOS provides tooling to build variants up iteratively. **Finally**, Singularity-EOS provides a flexible plugin infrastructure that allows downstream users to add capability to the core library locally by telling the build system to include a locally downloaded plugin. This final capability allows downstream users to share code with each other, even when committing that code to Singularity-EOS proper is not possible due to, e.g., licensing issues.

## Acknowledgements

This work was supported through the Laboratory Directed Research and Development program, the Center for Space and Earth Sciences, and the Center for Nonlinear Studies under project numbers 20240477CR-SES and 20220564ECR at Los Alamos National Laboratory (LANL). LANL is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001). This research used resources provided by the Darwin testbed at LANL which is funded by the Computational Systems and Software Environments subprogram of LANL's Advanced Simulation and Computing program (NNSA/DOE). This work is approved for unlimited release with report number LA-UR-24-23364.

## References

- Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., & McDonald, J. (2001). *Parallel programming in OpenMP*. Morgan kaufmann.
- Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., Supinski, B. R. de, & Futral, S. (2015). The spack package manager: Bringing order to HPC software chaos. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. <https://doi.org/10.1145/2807591.2807623>
- Lyon, S. P., & Johnson, J. D. (1992). *Sesame: The los alamos national laboratory equation of state database* (LA-UR-92-3407). Los Alamos National Laboratory.
- Miller, J. M., Dolence, J. C., & Holladay, D. (2022). Not-Quite Transcendental Functions and their Applications. *arXiv e-Prints*, arXiv:2206.08957. <https://doi.org/10.48550/arXiv.2206.08957>
- NVIDIA, Vingelmann, P., & Fitzek, F. H. P. (2020). *CUDA, release: 10.2.89*. <https://developer.nvidia.com/cuda-toolkit>
- O'Connor, E., & Ott, C. D. (2010a). A new open-source code for spherically symmetric stellar collapse to neutron stars and black holes. *Classical and Quantum Gravity*, 27(11), 114103. <https://doi.org/10.1088/0264-9381/27/11/114103>
- O'Connor, E., & Ott, C. D. (2010b). *Stellar collapse: microphysics*. <https://stellarcollapse.org/equationofstate>
- OpenMP Architecture Review Board. (2013). *OpenMP application program interface version 4.0*. <https://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf>
- Pimentel, D. A. (2021). *EOSPAC user's manual: v.6.5*. Los Alamos National Lab. (LANL), Los Alamos, NM (United States). <https://doi.org/10.2172/1765849>
- Reyes, R., Brown, G., Burns, R., & Wong, M. (2020). SYCL 2020: More than meets the eye. *Proceedings of the International Workshop on OpenCL*. <https://doi.org/10.1145/3388333.3388649>
- Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D., Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., & Wilke, J. (2022). Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4), 805–817. <https://doi.org/10.1109/TPDS.2021.3097283>