

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

Jonah M. Miller^{1,2}✉, Daniel A. Holladay^{2,3}, Jeffrey H. Peterson⁴, Christopher M. Mauney^{2,5}, Richard Berger³, Anna Pietarila Graham⁵, Karen C. Tsai³, Brandon Barker^{1,2,6,7}, Alexander Holas^{2,3,8}, Ann E. Mattsson⁹, Mariam Gogilashvili^{1,2,7,10}, Joshua C. Dolence^{1,2}, Chad D. Meyer¹¹, Sriram Swaminarayan³, and Christoph Junghans³

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.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

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

Editor: [Kyle Niemeyer](#) ↗ 

Reviewers:

- [@parikshitbajpai](#)
- [@snikolov3](#)

Submitted: 19 April 2024

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC 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

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 descriptions 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 ε to density ρ , temperature T , and

39 potentially some unknown set of additional quantities $\vec{\lambda}$. However, other representations are
40 possible, and in common parlance an EOS is the collection of knowledge needed to reconstruct
41 some intrinsic thermodynamic quantities from others. For example, the speed of sound through
42 a material or the specific heat capacity, which are thermodynamic derivatives of the pressure
43 and the specific internal energy, are both determined by the EOS.

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

53 State of the Field

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

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

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

73 Design Principles and Feature Highlights

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

76 Flexibility in loop patterns

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

81 Flexibility in memory layout

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

85 Expose APIs to aid performance

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

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

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

96 Performance-portable polymorphism

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

105 Modifiers

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

111 Fast log-lookups

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

116 Extensibility via modular parts and plugins

117 Singularity-EOS is designed to be extensible. The `std::variant`-based polymorphism, com-
118 bined with modifiers, as described above, already provides significant flexibility. However,
119 downstream codes may wish to add functionality to the library. This may be implemented
120 in several ways. **First**, as Singularity-EOS is open source, contributions from downstream
121 developers are welcome. **Second**, a C++ code that depends on Singularity-EOS may implement

122 their own models and include them in a local variant object. Singularity-EOS provides tooling
123 to build variants up iteratively. **Finally**, Singularity-EOS provides a flexible plugin infrastructure
124 that allows downstream users to add capability to the core library locally by telling the build
125 system to include a locally downloaded plugin. This final capability allows downstream users
126 to share code with each other, even when committing that code to Singularity-EOS proper is
127 not possible due to, e.g., licensing issues.

128 Acknowledgements

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

138 Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., & McDonald, J. (2001). *Parallel
139 programming in OpenMP*. Morgan kaufmann.

140 Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., Supinski, B. R. de, &
141 Futral, S. (2015). The spack package manager: Bringing order to HPC software chaos.
142 *Proceedings of the International Conference for High Performance Computing, Networking,
143 Storage and Analysis*. <https://doi.org/10.1145/2807591.2807623>

144 Lyon, S. P., & Johnson, J. D. (1992). *Sesame: The los alamos national laboratory equation of
145 state database* (LA-UR-92-3407). Los Alamos National Laboratory.

146 Miller, J. M., Dolence, J. C., & Holladay, D. (2022). Not-Quite Transcendental Functions and
147 their Applications. *arXiv e-Prints*, arXiv:2206.08957. [https://doi.org/10.48550/arXiv.2206.
148 08957](https://doi.org/10.48550/arXiv.2206.08957)

149 NVIDIA, Vingelmann, P., & Fitzek, F. H. P. (2020). *CUDA, release: 10.2.89*. [https:
150 //developer.nvidia.com/cuda-toolkit](https://developer.nvidia.com/cuda-toolkit)

151 O'Connor, E., & Ott, C. D. (2010a). A new open-source code for spherically symmetric stellar
152 collapse to neutron stars and black holes. *Classical and Quantum Gravity*, 27(11), 114103.
153 <http://stacks.iop.org/0264-9381/27/i=11/a=114103>

154 O'Connor, E., & Ott, C. D. (2010b). *Stellar collapse: microphysics*. [https://stellarcollapse.
155 org/equationofstate](https://stellarcollapse.org/equationofstate)

156 Pimentel, D. A. (2021). *EOSPAC user's manual: v.6.5*. Los Alamos National Lab. (LANL),
157 Los Alamos, NM (United States).

158 Reyes, R., Brown, G., Burns, R., & Wong, M. (2020). SYCL 2020: More than meets the eye.
159 *Proceedings of the International Workshop on OpenCL*. [https://doi.org/10.1145/3388333.
160 3388649](https://doi.org/10.1145/3388333.3388649)

161 Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri,
162 R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D.,
163 Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., & Wilke, J.
164 (2022). Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions
165 on Parallel and Distributed Systems*, 33(4), 805–817. [https://doi.org/10.1109/TPDS.
166 2021.3097283](https://doi.org/10.1109/TPDS.2021.3097283)