The Journal of Open Source Software

Spiner: Performance Portable Routines for Generic, Tabulated, Multi-Dimensional Data

Jonah M. Miller *[1,2], **Daniel Holladay**[2,3], **Chad D. Meyer**[4], **Joshua C. Dolence**[1,2], **Sriram Swaminarayan**[3], **Christopher M. Mauney**[2,5], and **Karen Tsai**[3]

1 CCS-2, Computational Physics and Methods, Los Alamos National Laboratory, Los Alamos, NM **2** Center for Theoretical Astrophysics, Los Alamos National Laboratory, Los Alamos, NM **3** CCS-7, Applied Computer Science, Los Alamos National Laboratory, Los ALamos, NM **4** XCP-4, Continuum Models and Numerical Methods, Los Alamos National Laboratory, Los ALamos, NM **5** HPC-ENV, HPC Environments, Los Alamo National Laboratory, Los Alamos, NM

DOI: 10.21105/joss.04367

Software

- Review
- Repository
- Archive

Editor: Dan Foreman-Mackey

Reviewers:

- @lgarrison
- @jzrake

Submitted: 17 March 2022
Published: 05 July 2022

Summary

We present `Spiner`, a new, performance-portable library for working with tabulated data. Spiner provides efficient routines for multi-dimensional interpolation and indexing on both CPUs and GPUs—as well as more exotic hardware—including interwoven interpolation and indexing access patterns, as needed for radiation transport. Importantly, `Spiner` defines a data format, based on `HDF5`, that couples the tabulated data to the information required to interpolate it, which `Spiner` can read and move to GPU.

Statement of Need

As Moore's law comes to an end, more and more performance comes from specialized hardware, such as GPUs. A key tool in the toolbox for many scientific codes is tabulated data. Fluid and continuum dynamics codes often encapsulate the equation of state as data tabulated in density and temperature, for example as published in the `Sesame` database (Lyon & Johnson, 1992) or the stellar collapse database (O'Connor & Ott, 2010b), first presented in O'Connor & Ott (2010a). Radiation transport, such as that performed by Skinner et al. (2019) and Miller et al. (2019) uses emissivity and absorption opacity on tables such as those computed in Sullivan et al. (2016). As continuum dynamics is required for a variety of applications, such as astrophysics, geophysics, climate science, vehicle engineering, and national security, utilizing a very large number of supercomputer cycles, providing tabulated data for these applications has the potential for significant impact.

These capabilities must be supported on *all* hardware a code may be run on, whether this is an NVIDIA GPU, an Intel CPU, or a next generation accelerator manufactured by one of any number of hardware vendors. To our knowledge there is no performance portable interpolation library on which these codes can rely, and there is a clear need, which we have developed `Spiner` to meet. `Spiner` is now used in the open-source and on-going `Singularity-EOS` (The Singularity Team, 2022a), `Singularity-Opac` (The Singularity Team, 2022b), and Phoebus (The Phoebus Team, 2022) `projects`, which have separate code papers in preparation.

*jonahm@lanl.gov

State of the Field

Interpolation is a common problem, implemented countless times across software projects, and a core part of any introductory text on scientific computing (Press et al., 2007). In graphics applications, interpolation is so ubiquitous that hardware primitives are provided by GPUs. These hardware intrinsics are, however, severely limited for scientific application. For example, on NVIDIA GPUs, the values to be interpolated must be single precision floating point, and the interpolation coefficients themselves are only half-precision, which is often insufficient to capture the high precision required for scientific applications. As GPUs are inherently vector devices, hardware interpolation is also vectorized in nature. However, downstream applications may be easier to reason about if scalar operations are available. For example, equation of state lookups often require root finds on interpolated data, and this can be easier to implement as a scalar operation, even if the final operation is vectorized over warps. Texture interpolation also does not support multi-dimensional mixed indexing/interpolation operations where, say, three indices of a four-dimensional array are interpolated and one is merely indexed into. Interpolation in the inner loop of a root-finding operation can be quite computationally expensive. For example, in the open-source general relativistic neutrino radiation hydrodynamics code nubhlight (Miller et al., 2019), these operations cover approximately ten percent of the runtime. For simulations involving only fluid dynamics and no expensive six-dimensional Boltzmann solve as required for radiation, the fraction of a timestep can be significantly larger.

Moreover, relying on hardware intrinsics is not a *portable* solution. A software interpolation library can, if written with care, work on not only the current generation of accelerators, but also on general purpose CPUs and the next generation of hardware as well.

Unfortunately, a performance-portable implementation not tuned to a specific use-case or embedded in a larger project is (to our knowledge) not available in the literature. A common problem in performance-portable computing is the management of performance-portable data structures. Libraries, such as `Kokkos` (Trott et al., 2022), often provide this functionality.

Here we present `Spiner`, a performance-portable library for working with tabulated data, thus meeting the needs of simulation codes on emerging hardware. `Spiner` provides data structures for working with tabulated data on both CPU and GPU, routines for interpolating on and indexing into tabulated data, and a file format that couples data to the information required to interpolate it. `Spiner` therefore fills a gap in available open software, providing a needed service for performance-portable simulation codes.

Interpolation is far more ubiquitous than its application in continuum dynamics and radiation transport, and we expect `Spiner` will find applications in the broader space of applications, such as image resampling. However, the team built `Spiner` with simulations in mind.

Design Principles and Salient Features

We built `Spiner` with several design goals in mind. First and foremost, interpolation must be fast, sufficiently fast that interpolation operations are not the rate-limiting operation in a larger calculation. Second, `Spiner` must be lightweight. It should contain exactly enough features to be useful for relevant science applications. Similarly, `Spiner` must be sufficiently low-level and flexible that it can support all performance portability strategies and access patterns required of it. That said, not all needs will be known at conception, and needs change over time. Thus `Spiner` must be extendable. Finally, `Spiner` should be well-documented with a modern, easy-to-use build system. We believe we have achieved these goals.

To ensure `Spiner` is lightweight and performant, it is header-only. To ensure performance portability, we rely on the `Kokkos` (Trott et al., 2022) library to provide performance portable data structures and parallel dispatch. However, we recognize that another performance portability paradigm may be desired. Hence, we developed a separate library, `Ports-of-Call`,

which we open-sourced at lanl/ports-of-call on GitHub (The Ports-of-Call Team, 2022). `Ports-of-Call` is a very thin abstraction around low-level device calls. It provides preprocessor macros to enable or disable Kokkos, an arbirtrary-dimensional array data structure, and hooks to add the same functionality for other backends such as pure CPU, OpenMP (Chandra et al., 2001), or Cuda (NVIDIA et al., 2020). `Ports-of-Call` is only a few hundred lines long. Both Spiner and Ports-of-Call are well documented, with Sphinx documentation provided automatically by github pages and github actions. They both also have modern build systems, with Cmake and Spack support. Unit tests are provided by `Catch2` (The Catch2 Collaboration, 2013).

The fundamental data structures are the `DataBox` and `RegularGrid1D`. The former relies heavily on the `PortableMDArray` data structure in `Ports-of-Call` for data storage, which provides an arbitrary-dimensional accessor to a contiguous block of data, as well as support for slicing, shallow copying, and resizing data. The latter contains information required to interpolate. The former contains both the data to interpolate, as well as multiple `RegularGrid1Ds`. Both objects know how to read from and write to an `HDF5` (The HDF Group, 2000) file, and the intent is that the `RegularGrid1D` is a hook that could be extended into a more sophisticated gridding or interpolation procedure. For example, one could implement a multi-level grid hierarchy or higher-order interpolation stencils. A `DataBox` can manage its own memory and can automatically allocate on host or device at runtime. Deep copies host-to-host and host-to-device are supported. To encourage good performance, no deep copies are ever implicitly performed—they must be explicitly requested. Consequently, `DataBoxes` have reference semantics. By deliberate choice, `DataBoxes` are not reference-counted and have trivial destructors. Instead, we provide an overload of `free` to free the data. However, `DataBoxes` can be managed via smart pointers—and we provide machinery to do so. We find this approach minimizes code complexity and carries most of the benefits of an automatically reference-counted data structure.

Performance And Accuracy

Since it is usually sufficient for the intended use-cases, only multilinear interpolation is supported, although as discussed above, hooks are present in the code for more sophisticated approaches. A convergence test is available in the test suite and shows excellent second-order convergence as expected. Performance on both CPUs and GPUs is also excellent. For example, the figure below benchmarks trilinear interpolation at double precision from a 64^3 grid on to a cubic grid of varying sizes. We test on a single Intel Xeon (Haswell) core, twenty cores accross two sockets (with a Kokkos OpenMP backend), and one Nvidia V100 GPU (with the Kokkos Cuda backend). On the V100, profiling tools report we achieve 15% of peak performance in terms of flops and 46% of peak in terms of memory bandwidth.

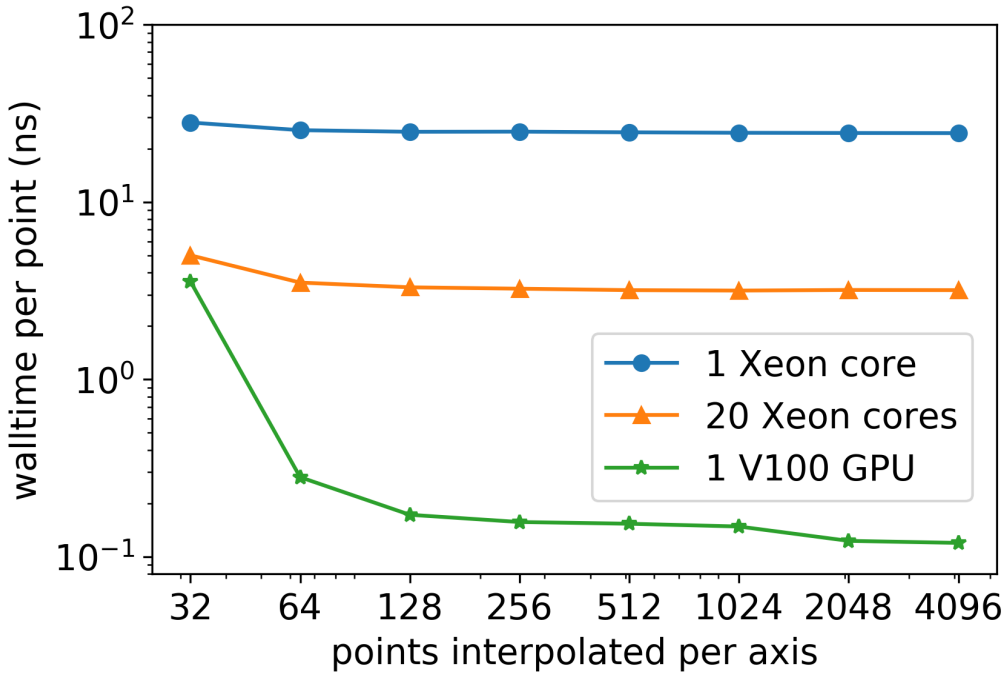

Figure 1: Performance of trilinear interpolation on one Haswell core, twenty Haswell cores, and a V100 GPU. Smaller is better. The rightmost point is over 68 billion interpolation operations.

Acknowledgements

This work was supported by the U.S. Department of Energy through the 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-22-22502.

References

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

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., Ryan, Ben. R., & Dolence, J. C. (2019). ν bhlight: Radiation GRMHD for Neutrino-driven Accretion Flows. *The Astrophysical Journal Supplement Series*, *241*(2), 30. https://doi.org/10.3847/1538-4365/ab09fc

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

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical recipes with source code CD-ROM 3rd edition: The art of scientific computing*. Cambridge University Press. ISBN: 9780521884075

Skinner, M. A., Dolence, J. C., Burrows, A., Radice, D., & Vartanyan, D. (2019). FORNAX: A flexible code for multiphysics astrophysical simulations. *The Astrophysical Journal Supplement Series*, *241*(1), 7. https://doi.org/10.3847/1538-4365/ab007f

Sullivan, C., O'Connor, E., Zegers, R. G. T., Grubb, T., & Austin, S. M. (2016). The sensitivity of core-collapse supernovae to nuclear electron capture. *The Astrophysical Journal*, *816*(1), 44. https://doi.org/10.3847/0004-637X/816/1/44

The Catch2 Collaboration. (2013). Catch2. In *GitHub repository*. GitHub. https://github.com/catchorg/Catch2

The HDF Group. (2000). *Hierarchical data format version 5*. http://www.hdfgroup.org/HDF5

The Phoebus Team. (2022). Phoebus: Phifty one ergs blows up a star. In *GitHub repository*. GitHub. https://github.com/lanl/phoebus

The Ports-of-Call Team. (2022). Ports-of-call. In *GitHub repository*. GitHub. https://github.com/lanl/ports-of-call

The Singularity Team. (2022a). Singularity-EOS: Performance portable equations of state. In *GitHub repository*. GitHub. https://github.com/lanl/singularity-eos

The Singularity Team. (2022b). Singularity-opac: Performance portable opacities. In *GitHub repository*. GitHub. https://github.com/lanl/singularity-opac

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