The Journal of Open Source Software

Cabana: A Performance Portable Library for Particle-Based Simulations

Stuart Slattery[1¶], Samuel Temple Reeve[1], Christoph Junghans[2], Damien Lebrun-Grandié[1], Robert Bird[2], Guangye Chen[2], Shane Fogerty[2], Yuxing Qiu[3], Stephan Schulz[4], Aaron Scheinberg[5], Austin Isner[1], Kwitae Chong[1], Stan Moore[6], Timothy Germann[2], James Belak[7], and Susan Mniszewski[2]

1 Oak Ridge National Laboratory, Oak Ridge, TN, USA **2** Los Alamos National Laboratory, Los Alamos, NM, USA **3** University of California, Los Angeles, Los Angeles, CA, USA **4** Jülich Supercomputing Centre, Jülich, Germany **5** Jubilee Development, Cambridge, MA, USA **6** Sandia National Laboratories, Albuquerque, NM, USA **7** Lawrence Livermore National Laboratory, Livermore, CA, USA ¶ Corresponding author

DOI: 10.21105/joss.04115

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @atmyers
- @XzzX

Submitted: 28 January 2022
Published: 08 April 2022

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

Summary

Particle-based simulations are ubiquitous throughout many fields of computational science and engineering, spanning the atomistic level with molecular dynamics (MD), to mesoscale particle-in-cell (PIC) simulations for solid mechanics, device-scale modeling with PIC methods for plasma physics, and massive N-body cosmology simulations of galaxy structures, with many other methods in between (Hockney & Eastwood, 1989). While these methods use particles to represent significantly different entities with completely different physical models, many low-level details are shared including performant algorithms for short- and/or long-range particle interactions, multi-node particle communication patterns, and other data management tasks such as particle sorting and neighbor list construction.

Cabana is a performance portable library for particle-based simulations, developed as part of the Co-Design Center for Particle Applications (CoPA) within the Exascale Computing Project (ECP) (Alexander et al., 2020). The CoPA project and its full development scope, including ECP partner applications, algorithm development, and similar software libraries for quantum MD, is described in (Mniszewski et al., 2021). Cabana uses the Kokkos library for on-node parallelism (Edwards et al., 2014; Trott et al., 2022), enabling simulation on multi-core CPU and GPU architectures, and MPI for GPU-aware, multi-node communication. Cabana provides particle simulation capabilities on almost all current Kokkos backends, including serial execution, OpenMP (including OpenMP-Target for GPUs), CUDA (NVIDIA GPUs), HIP (AMD GPUs), and SYCL (Intel GPUs), providing a clear path for the coming generation of accelerator-based exascale hardware. Cabana builds on Kokkos by providing new particle data structures and particle algorithms resulting in a similar execution policy-based, node-level programming model that is intended to be used in addition to the core Kokkos library within an application. Cabana is designed as an application and physics agnostic, but particle-specific toolkit which can either be used to generate a new application, or to be used as needed in existing applications at various levels of invasiveness including through interfaces that wrap user memory in existing data structures.

Statement of need

For the most part, particle simulation codes targeting high performance computing have been developed specific to a given application area. Examples include `HACC` for cosmology (Habib et al., 2016), `LAMMPS` for atomic systems (Plimpton, 1995), and `XGC` for plasma physics (Ku et al., 2018) (all ECP partner applications for the CoPA project). In contrast, other areas of computational science have successfully developed motif-based libraries, e.g. `AMReX` for block structured adaptive mesh based simulations (Zhang et al., 2019), with many applications sharing the development effort for common needs. Co-designing software for a simulation "motif" such as particles is increasingly important as hardware for scientific simulations continues to evolve, becoming more heterogeneous, requiring more effort to extract performance, and otherwise likely requiring separate versions of a single application (or kernel) for each vendor-specific API to utilize the multitude of available accelerators. We note that there are some well designed and broad featured general particle simulation libraries: `AutoPas` automatically chooses underlying particle algorithms and parallel options using performance tuning (Gratl et al., 2021) and `OpenFPM` includes both particle and particle-grid capabilities (Incardona et al., 2019). However, the former does not currently support GPUs and the latter currently supports only `CUDA` (and does so internally, rather than through a separate library such as `Kokkos`). To address this need, our objective is to provide scalable software services applicable to high-performance scientific codes across numerous application domains through general particle algorithms and data structures that are performant on a variety of distributed memory and accelerated architectures in a single implementation.

Particle capability

`Cabana` provides particle data structures to optimize performance across hardware through an array-of-structs-of-arrays (AoSoA) concept. This directly extends the `Kokkos::View` (portable multidimensional arrays) with an additional dimension that creates small, statically-sized groups of particles. Intermediate between struct-of-arrays (SoA) and array-of-structs (AoS), the size of these groups may be changed depending on the compute hardware in use which makes the data layout configurable to achieve the performance of SoA in SIMD-like settings where coalescing is achievable and the memory locality of AoS when random access memory patterns dominate. This tunable layout was designed to enable optimal performance across multiple kernels and across different hardware for a given application.

The main algorithmic functionality of the library includes particle neighbor list generation and traversal, particle redistribution and halo communication for domain decomposition, and particle sorting. Particle sorting currently builds directly on `Kokkos` binning and sorting capabilities. Similarly, parallel iteration within `Cabana` algorithms directly uses `Kokkos` options for threaded parallelism. As with the AoSoA, `Cabana` extends the `Kokkos::parallel_for` (portable parallel execution) with SIMD-parallel capabilities for threading over the `AoSoA` data structures, as well as neighbor-parallel iteration over both central particles and their neighbors (potentially with many-body interactions and multiple levels of neighbors) with user-configurable options for serial or threaded execution as needed for application performance. Particle communication for multi-node simulation uses GPU-aware `MPI`, with capabilities for migrating particles from one unique owning rank to another, as well as to gather and scatter particle information from owning ranks to neighboring ranks.

Existing packages are also leveraged for accelerating complex operations. For example, an interface to `ArborX`, a library for performance portable geometric search also built on `Kokkos`, has been included for neighbor list creation which is more scalable for non-uniform particle distributions than other options in `Cabana` (Lebrun-Grandié et al., 2020).

Particle-grid capability

In addition to particle-specific algorithms, almost all particle-based codes use grids in some way. To support these applications Cabana also provides algorithms and data structures for many particle-grid motifs within the `Cajita` subpackage. Distributed, logically rectilinear grid data structures and high order, multidimensional spline kernels and spatial gradients are available, together with the requisite parallel communication to interpolate data between particles and grids. While this is most relevant to PIC methods and long-range MD algorithms, grid structures are useful even in simulations which are generally "mesh-free" (e.g. short-range MD), for accelerating neighbor list generation and multi-node spatial decomposition. In addition, support for sparse grids is in progress to enable only allocating and iterating over the grid where particles exist.

As with the core particle package, `Cajita` includes interfaces to separate libraries for complex particle-grid related motifs. This includes distributed, performance portable fast Fourier transforms (e.g. for long-range MD and N-body simulations) through the heFFTe library (Ayala et al., 2019)), preconditioners and linear solvers for structured grids through the HYPRE library (Falgout & Yang, 2002), as well as redistribution of the grid across MPI ranks for non-uniform particle distributions through the ALL load-balancing library (Halver et al., n.d.).

Exascale design patterns

Cabana has been designed for performance across both multi and many-core systems (CPU and GPU). Often, a focus on GPU performance also results in relatively good CPU performance (while the reverse is often not true). However, one notable exception is the relative efficiency of threaded atomic operations (hardware support for avoiding data race conditions) on each type of device. `Kokkos` provides support for this discrepancy through the `Kokkos::ScatterView`, where by default the GPU uses atomic memory and the CPU uses data duplication, which is being increasingly relied upon throughout Cabana. Cabana also primarily encourages a GPU-resident strategy: data is created and computed on the device without intermittent copies to the host to whatever degree possible. This is in contrast to a GPU-offload approach, although this strategy is still possible through the library.

Data structures across physics kernels

The AoSoA data structure enables not only flexibility for ideal layouts for different architectures, but also for complex physics applications with multiple kernels that are each optimized for a different data layout. This is common, for example, in molecular dynamics where the particle update (integration) kernel has regular memory access across all particles which can be effectively coalesced and therefore an SoA would be the optimal choice. Elsewhere in the time integration loop, the particle interaction kernel is based on random memory accesses across neighbors of each particle such that an AoS would be preferred for performance. Rather than pay the performance penalty of choosing one data layout (SoA or AoS) which is suboptimal for at least one kernel or transposing data to the optimal layout for each kernel, the AoSoA enables performance near the ideal for both kernels. Further discussion of the AoSoA is provided in (Mniszewski et al., 2021).

Separating memory and execution

In creating Cabana, optimal design patterns have emerged to build application functionality. The first such pattern is the separation of memory and execution spaces, which are general `Kokkos` concepts for where data resides and where parallel execution takes place: host (CPU) or device (GPU). The `Kokkos Device` combines both in one object, leading to one design strategy in the following code:

```
template<Device>
struct Foo
```

```
{
    View<typename Device::memory_space> _device_data;

    void bar()
    {
        parallel_for<typename Device::execution_space>( _device_data );
    }
};
```

The class is created such that the same memory and execution space must always be used to store and operate on the data. In contrast, Cabana has moved to the following design:

```
template<MemorySpace>
struct Foo
{
    View<MemorySpace> _device_data;

    template<class ExecutionSpace>
    void bar( const ExecutionSpace& exec_space )
    {
        static_assert( is_accessible_from<MemorySpace,
                                          ExecutionSpace>{}, "" );

        parallel_for( exec_space, _device_data );
    }
};
```

Here, the class data is stored in a specific memory space and, separately for each parallel execution, the user can choose any execution space that is compatible with that memory space. This greatly increases the flexibility of the class for using different parallel threading backends on a given device, e.g. both `OpenMP-Target` and vendor-specific backends. This also extends to easier adoption of newer execution options, such as `CUDA` streams, which can enable coarse-grained asynchronous tasking in applications. In addition, this makes the class amenable to both separate host or device computation as well as an offload model where a new overload that first copies the data to the class memory space is all that is required. In user code, multiple instances of this class may be used with different memory and execution spaces possible for each instance.

Enabling kernel fusion

A more specific design pattern that enables not only flexibility, but also significant performance improvements in some cases, is support for kernel fusion. As an example, below is a straightforward implementation for a simulation that needs particle-grid interpolation for multiple physical entities using `Cajita`:

```
// Create halo exchange pattern for an individual array.
auto halo = Cajita::createHalo( field, ... );

// Interpolate scalar gradient to the grid with kernel and MPI scatter.
auto val_1 = Cajita::createScalarGradientP2G( ... );
Cajita::p2g( val_1, ..., halo, ...);

// Interpolate tensor divergence to the grid with kernel and MPI scatter.
auto val_2 = Cajita::createTensorDivergenceP2G( ... );
Cajita::p2g( val_2, ..., halo, ... );
```

Often, the time to launch each kernel and communicate the data (in a distributed and accelerated computing setting) is significant compared to the time for the execution of the

parallel kernel itself which introduces significant latency costs. In the above case, each function call requires a separate parallel kernel and scatter communication kernel. The following reimplementation can, in some cases, improve performance considerably:

```
// Create fused halo exchange pattern.
auto fused_halo = Cajita::createHalo( ..., *field_1, *field_2 );

// Fused local interpolation of both properties.
Kokkos::parallel_for( exec_space, num_point, points,
    KOKKOS_LAMBDA( const Particle& p ){
        Cajita::SplineData<float,3,Cajita::Node> sd;
        Cajita::evaluateSpline( p.x );
        Cajita::P2G::gradient( sd, p.scalar_field, field_1 );
        Cajita::P2G::divergence( sd, p.tensor_field, field_2 ); });

// Fused MPI scatter.
fused_halo.scatter( exec_space, Cajita::ScatterReduce::Sum,
                    field_1, field_2 );
```

There are a number of benefits to this approach. First, the number of kernel launches in an accelerated setting and MPI communication calls have been reduced by a factor of 2, thus reducing latency. Second, identical quantities that would have been computed in each interpolation kernel, such as the spline interpolation data, can be reused for multiple interpolations to reduce total operation counts. Third, our experience shows that kernel fusion often allows for temporary variables that are needed across multiple kernels no longer need to be allocated in large global memory arrays and can instead become in-kernel, thread-local temporaries, significantly reducing memory costs. Finally, cache performance can be significantly improved due to global data reuse combined with the AoSoA data structure such that a single particle is accessed multiple times in a single kernel rather than a single time in multiple kernels.

Tutorial, proxy applications, and Fortran support

An extensive set of documentation, tests, and examples are available for Cabana including unit tests, tutorial examples, and performance testing across library functionality along with the GitHub wiki and doxygen API documentation. Continuous integration is used to ensure software quality, with testing across Kokkos backends and corresponding architectures. In addition, a Cabana Docker container is deployed, spack installation is available, and Cabana is a part of the Extreme-scale Scientific Software Stack (E4S) (E4S, 2021) to enable easy testing and use. For Fortran integration, a separate repository exemplifies using Cabana with Fortran applications (CoPA, 2021). Many proxy applications have also been developed using Cabana: CabanaMD for MD, CabanaPIC for plasma PIC, ExaMPM for the material point method (MPM), and HACCabana for N-body cosmology (CoPA, 2021). Proxy apps are relatively simple representations of the main physics in production applications and have proven useful within the Cabana development process for demonstrating library needs, capability, and performance. The proxy apps developed thus far also demonstrate the potential for rapid prototyping of particle codes on emerging hardware and interactions with hardware vendors.

Application adoption and future work

Cabana is designed for high-performance, large-scale particle simulations, with early adoption by the XGC plasma physics code (Mniszewski et al., 2021; Scheinberg et al., 2019), as well as the new PicassoMPM code for additive manufacturing (Belak et al., 2019), both a part of ECP. One important aspect of continuing work is consistent interaction with Cabana-based applications, contributing algorithms and data structures back to Cabana where they could be useful in other particle applications. Similarly, interaction with the Kokkos team is critical to keep

Cabana up-to-date with the latest architecture trends, but also to potentially contribute general parallel approaches or data structures from Cabana, where appropriate. Other continuing work includes tighter integration of the particle and particle-grid motifs, load balancing, additional input/output capabilities, and performance optimizations on early exascale systems.

Acknowledgments

The authors would like to thank Steve Plimpton for fruitful discussion on design and implementation.

This work was performed as part of the Co-design Center for Particle Applications, supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. DOE Office of Science and the NNSA.

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy (DOE). The publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.

This work was performed at Lawrence Livermore National Laboratory under U.S. Government Contract DE-AC52-07NA27344, Oak Ridge National Laboratory under U.S. Government Contract DE-AC05-00OR22725, Los Alamos National Laboratory, and at Sandia National Laboratories.

Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy (Contract No. 89233218NCA000001).

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract number DE-NA-0003525.

This research used resources of the Oak Ridge Leadership Computing Facility (OLCF), supported by DOE under contract DE-AC05-00OR22725.

References

Alexander, F., Almgren, A., Bell, J., Bhattacharjee, A., Chen, J., Colella, P., Daniel, D., DeSlippe, J., Diachin, L., Draeger, E., Dubey, A., Dunning, T., Evans, T., Foster, I., Francois, M., Germann, T., Gordon, M., Habib, S., Halappanavar, M., … Yelick, K. (2020). Exascale applications: Skin in the game. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *378*(2166), 20190056. https://doi.org/10.1098/rsta.2019.0056

Ayala, A., Tomov, S., Luo, X., Shaiek, H., Haidar, A., Bosilca, G., & Dongarra, J. (2019). *Impacts of multi-GPU MPI collective communications on large FFT computation*. https://doi.org/10.1109/ExaMPI49596.2019.00007

Belak, J., Turner, J., & Team, E. T. (2019). Exaam: Additive manufacturing process modeling at the fidelity of the microstructure. *APS March Meeting Abstracts*, *2019*, C22–010.

CoPA. (2021). *Co-design Center for Particle Applications: Libraries and proxy applications*. GitHub. http://github.com/ECP-CoPA

E4S. (2021). *The extreme-scale scientific software stack*. GitHub. https://github.com/E4S-Project

Edwards, H. C., Trott, C. R., & Sunderland, D. (2014). Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, *74*(12), 3202–3216. https://doi.org/10.1016/j.jpdc.2014.07.003

Falgout, R. D., & Yang, U. M. (2002). Hypre: A Library of High Performance Preconditioners. In P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, & J. J. Dongarra (Eds.), *Computational Science — ICCS 2002* (pp. 632–641). Springer. https://doi.org/10.1007/3-540-47789-6_66

Gratl, F. A., Seckler, S., Bungartz, H.-J., & Neumann, P. (2021). N ways to simulate short-range particle systems: Automated algorithm selection with the node-level library AutoPas. *Computer Physics Communications*, 108262. https://doi.org/10.1016/j.cpc.2021.108262

Habib, S., Pope, A., Finkel, H., Frontiere, N., Heitmann, K., Daniel, D., Fasel, P., Morozov, V., Zagaris, G., Peterka, T., Vishwanath, V., Lukić, Z., Sehrish, S., & Liao, W. (2016). HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astron.*, *42*, 49–65. https://doi.org/10.1016/j.newast.2015.06.003

Halver, R., Schulz, S., & Sutmann, G. (n.d.). *ALL - A loadbalancing library, C++ / Fortran library*. `https://gitlab.version.fz-juelich.de/SLMS/loadbalancing/-/releases`. http://slms.pages.jsc.fz-juelich.de/websites/all-website

Hockney, R. W., & Eastwood, J. W. (1989). *Computer Simulation Using Particles* (1st edition). CRC Press. ISBN: 978-0-85274-392-8

Incardona, P., Leo, A., Zaluzhnyi, Y., Ramaswamy, R., & Sbalzarini, I. F. (2019). OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers. *Computer Physics Communications*, *241*, 155–177. https://doi.org/10.1016/j.cpc.2019.03.007

Ku, S., Chang, C., Hager, R., Churchill, R., Tynan, G., Cziegler, I., Greenwald, M., Hughes, J., Parker, S., Adams, M., D'Azevedo, E., & Worley, P. (2018). A fast low-to-high confinement mode bifurcation dynamics in the boundary-plasma gyrokinetic code XGC1. *Physics of Plasmas*, *25*, 056107. https://doi.org/10.1063/1.5020792

Lebrun-Grandié, D., Prokopenko, A., Turcksin, B., & Slattery, S. R. (2020). ArborX: A performance portable geometric search library. *ACM Transactions on Mathematical Software (TOMS)*, *47*(1), 1–15. https://doi.org/10.1145/3412558

Mniszewski, S. M., Belak, J., Fattebert, J.-L., Negre, C. F., Slattery, S. R., Adedoyin, A. A., Bird, R. F., Chang, C., Chen, G., Ethier, S., Fogerty, S., Habib, S., Junghans, C., Lebrun-Grandié, D., Mohd-Yusof, J., Moore, S. G., Osei-Kuffuor, D., Plimpton, S. J., Pope, A., … Wall, M. E. (2021). Enabling particle applications for exascale computing platforms. *The International Journal of High Performance Computing Applications*, *0*(0), 10943420211022829. https://doi.org/10.1177/10943420211022829

Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, *117*(1), 1–19. https://doi.org/10.1006/jcph.1995.1039

Scheinberg, A., Chen, G., Ethier, S., Slattery, S., Bird, R., Worley, P., & Chang, C. (2019). Kokkos and Fortran in the exascale computing project plasma physics code XGC. *Proceedings of Sc19 Conference*.

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

Zhang, W., Almgren, A., Beckner, V., Bell, J., Blaschke, J., Chan, C., Day, M., Friesen, B., Gott, K., Graves, D., Katz, M., Myers, A., Nguyen, T., Nonaka, A., Rosso, M., Williams, S., & Zingale, M. (2019). AMReX: A framework for block-structured adaptive mesh refinement. *Journal of Open Source Software*, *4*(37), 1370. https://doi.org/10.21105/joss.01370