The Journal of Open Source Software

DDC: The Discrete Domain Computation library

Thomas Padioleau [1*], **Julien Bigot** [1*], **and Emily Bourne** [2]

1 Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, 91191, Gif-sur-Yvette, France **2** SCITAS, EPFL, CH-1015 Lausanne, Switzerland * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Mauricio Pacha Vargas Sepulveda

Reviewers:

- @lwshanbd
- @cfguzman

Submitted: 21 May 2025
Published: unpublished

DDC: The Discrete Domain Computation library

Summary

The Discrete Domain Computation (DDC) library is a C++ library designed to provide high-performance, strongly-typed labelled multidimensional arrays. Inspired by Python's Xarray and built on top of performance-portable libraries like Kokkos, DDC enables expressive, safe, and efficient numerical computations. It provides a coherent ecosystem to work with labelled dimensions from data structures to algorithms. Additionally, DDC extends functionality through modules such as FFT (based on kokkos-fft), splines, and with a bridge to the PDI library. The library is actively used to modernize legacy scientific codes, such as the Fortran-based Gysela plasma simulation code (Grandgirard et al., 2016).

Statement of need

The use of multidimensional arrays is widespread across various fields, particularly in scientific computing, where they serve as fundamental data containers. A primary motivation for their use is their potential to improve computational performance by leveraging problem-specific structures. For instance, when solving a partial differential equation that results in a stencil problem, computations typically achieve higher efficiency on a structured mesh compared to an unstructured mesh. This advantage primarily stems from a better usage of memory, with predictable accesses, and better cache utilization.

Many programming languages commonly used in scientific computing support multidimensional arrays in different ways. Fortran, a longstanding choice in the field, and Julia, a more recent language, both natively support these data structures. In contrast, the Python ecosystem relies on the popular NumPy library's `numpy.Array` (Harris et al., 2020). Meanwhile, C++23 introduced `std::mdspan` to the standard library. This container was inspired by `Kokkos::View` from the Kokkos library which also serves as the foundation of DDC.

Despite their importance, multidimensional arrays introduce several practical challenges. In a sense, they encourage the usage of implicit information in the source code. A frequent source of errors is the inadvertent swapping of indices when accessing elements. Such errors can be difficult to detect, especially given the common convention of using single-letter variable names like `i` and `j` for indexing. Another challenge in medium to large codebases is the lack of semantic clarity in function signatures when using raw multidimensional arrays. When array dimensions carry specific meanings, this information is not explicitly represented in the source code, leaving it up to the user to ensure that dimensions are ordered correctly according to implicit expectations. For example it is quite usual to use the same index for multiple interpretations: looping over mesh cells identified by `i` and interpreting `i+1` as the face to the right. Another example is slicing that removes dimensions, this can shift the positions of remaining dimensions, altering the correspondence between axis indices and their semantic meanings.

Solutions have been proposed to address these issues. For example in Python, the Xarray (Hoyer & Hamman, 2017) library allows users to label dimensions that can then be used to perform computations. Following a similar approach, the "Discrete Domain Computation" (DDC) library aims to bring equivalent functionality to the C++ ecosystem. It uses a zero overhead abstraction approach, i.e. with labels fixed at compile-time, on top of different performant portable libraries, such as: Kokkos (Trott et al., 2022), Kokkos Kernels (Rajamanickam et al., 2021), kokkos-fft (The kokkos-fft team, n.d.) and Ginkgo (Anzt et al., 2020). Labelling at compile time is achieved by strongly typing dimensions, an approach similar to that used in units libraries such as mp-units (Pusz et al., 2024), which strongly type quantities rather than dimensions.

The library is actively used to modernize the Fortran-based Gysela plasma simulation code (Bourne et al., n.d.). This simulation code relies heavily on high-dimensional arrays. While the data stored in the arrays has 7 dimensions, each dimension can have multiple representations, including Fourier, spline, Cartesian, and various curvilinear meshes. The legacy Fortran implementation, used to manipulate multi-dimensional arrays that stored slices of all the possible dimensions with very limited information about which dimensions were actually represented to enforce correctness at the API level. DDC enables a more explicit, strongly-typed representation of these arrays, ensuring at compile-time that function calls respect the expected dimensions. This reduces indexing errors and improves code maintainability, particularly in large-scale scientific software.

DDC Core key features

The DDC library is a C++ library designed for expressive and safe handling of multidimensional data. Its core component provides flexible data containers along with algorithms built on top of the performance portable Kokkos library. The library is fully compatible with Kokkos and does not attempt to hide it, allowing users to leverage Kokkos' full capabilities while benefiting from DDC's strongly-typed, labelled multidimensional arrays when and where it makes sense.

Strongly-typed labelled indices

DDC employs strongly-typed multi-indices to label dimensions and access data. It introduces two types of multi-indices to access the container's data:

- `DiscreteVector` multi-indices:
  - strongly-typed labelled integers,
  - provide a multidimensional array access semantics,
  - always as fast access as raw multidimensional array,
- `DiscreteElement` multi-indices:
  - strongly-typed labelled keys or opaque identifiers,
  - provide an associative access semantics, as keys in a map container,
  - potentially slower access, depending on the type of set of `DiscreteElement`.

In a DDC container, `DiscreteElement` indices represent absolute positions, while `DiscreteVector` indices are always relative to the beginning of the container.

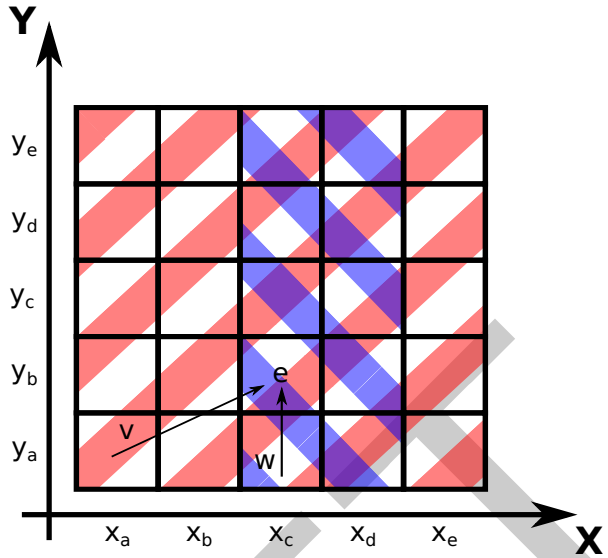

Figure 1: Example of two sets of `DiscreteElement`.

For example consider Figure 1 that illustrates a two-dimensional data chunk with axes `X` and `Y`. Here `chunk_r` is a container defined over the red area and `chunk_b` is a slice of `chunk_r` over the blue area. Let us define

- `DiscreteElement<X, Y> e(x_c, y_b)`,
- `DiscreteVector<X, Y> v(2, 1)`,
- `DiscreteVector<X, Y> w(0, 1)`.

In this case, the following expressions all refer to the same memory location:

- `chunk_r(e)`,
- `chunk_b(e)`,
- `chunk_r(v)`,
- `chunk_b(w)`.

This highlights the fact that `DiscreteElement` provides a globally consistent indexing mechanism, while `DiscreteVector` is context-dependent and relative to the container's origin.

Sets of `DiscreteElement`

The semantics of DDC containers associates data to a set of `DiscreteElement` indices. Let us note that the set of all possible `DiscreteElement` has a total order that is typically established once and for all at program initialization. Thus to be able to construct a DDC container one must provide a multidimensional set of `DiscreteElement` indices, only these indices can be later used to access the container's data.

The library provides several ways to group `DiscreteElement` into sets, each represented as a Cartesian product of per-dimension sets. These sets offer a lookup function to retrieve the position of a multi-index relative to the front of the set. The performance of container data access depends significantly on the compile-time properties of the set used.

Multidimensional algorithms

Finally, DDC offers multidimensional algorithms to manipulate the containers and associated `DiscreteElement` indices such as parallel loops and reductions. The parallel versions are based on Kokkos providing performance portability. DDC also provides transform-based algorithms such as discrete Fourier transforms (via a Kokkos-fft wrapper) and spline transforms, enabling

conversions between sampled data and coefficients in Fourier or B-spline bases over labeled dimensions.

Acknowledgements

We acknowledge the financial support of the Cross-Disciplinary Program on "Numerical simulation" of CEA, the French Alternative Energies and Atomic Energy Commission. This work has received support by the CExA Moonshot project of the CEA cexa-project. We acknowledge contributions from the Maison de la Simulation. We also thank the developers and contributors of the DDC project for their efforts in making numerical modeling more accessible and efficient.

References

Anzt, H., Cojean, T., Chen, Y.-C., Flegar, G., Göbel, F., Grützmacher, T., Nayak, P., Ribizel, T., & Tsai, Y.-H. (2020). Ginkgo: A high performance numerical linear algebra library. *Journal of Open Source Software*, *5*(52), 2260. https://doi.org/10.21105/joss.02260

Bourne, E., Grandgirard, V., Asahi, Y., Bigot, J., Donnel, P., Hoffmann, A., Kara, A., Krah, P., Legouix, B., Malaboeuf, E., Midou, D., Munschy, Y., Peybernes, M., Protais, M., Obrejan, K., Padioleau, T., & Vidal, P. (n.d.). *Gyselalib++* (Version 0.1.0). https://github.com/gyselax/gyselalibxx

Grandgirard, V., Abiteboul, J., Bigot, J., Cartier-Michaud, T., Crouseilles, N., Dif-Pradalier, G., Ehrlacher, Ch., Esteve, D., Garbet, X., Ghendrih, Ph., Latu, G., Mehrenberger, M., Norscini, C., Passeron, Ch., Rozar, F., Sarazin, Y., Sonnendrücker, E., Strugarek, A., & Zarzoso, D. (2016). A 5D gyrokinetic full-f global semi-lagrangian code for flux-driven ion turbulence simulations. *Computer Physics Communications*, *207*, 35–68. https://doi.org/10.1016/j.cpc.2016.05.007

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hoyer, S., & Hamman, J. (2017). Xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.148

Pusz, M., Guerrero Peña, J. E., Hogg, C., & The mp-units project team. (2024). *mp-units* (Version 2.4.0). https://github.com/mpusz/mp-units

Rajamanickam, S., Acer, S., Berger-Vergiat, L., Dang, V., Ellingwood, N., Harvey, E., Kelley, B., Trott, C. R., Wilke, J., & Yamazaki, I. (2021). Kokkos kernels: Performance portable sparse/dense linear algebra and graph kernels. *arXiv Preprint arXiv:2103.11991*.

The kokkos-fft team. (n.d.). *kokkos-fft* (Version 0.3.0). https://github.com/kokkos/kokkos-fft

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