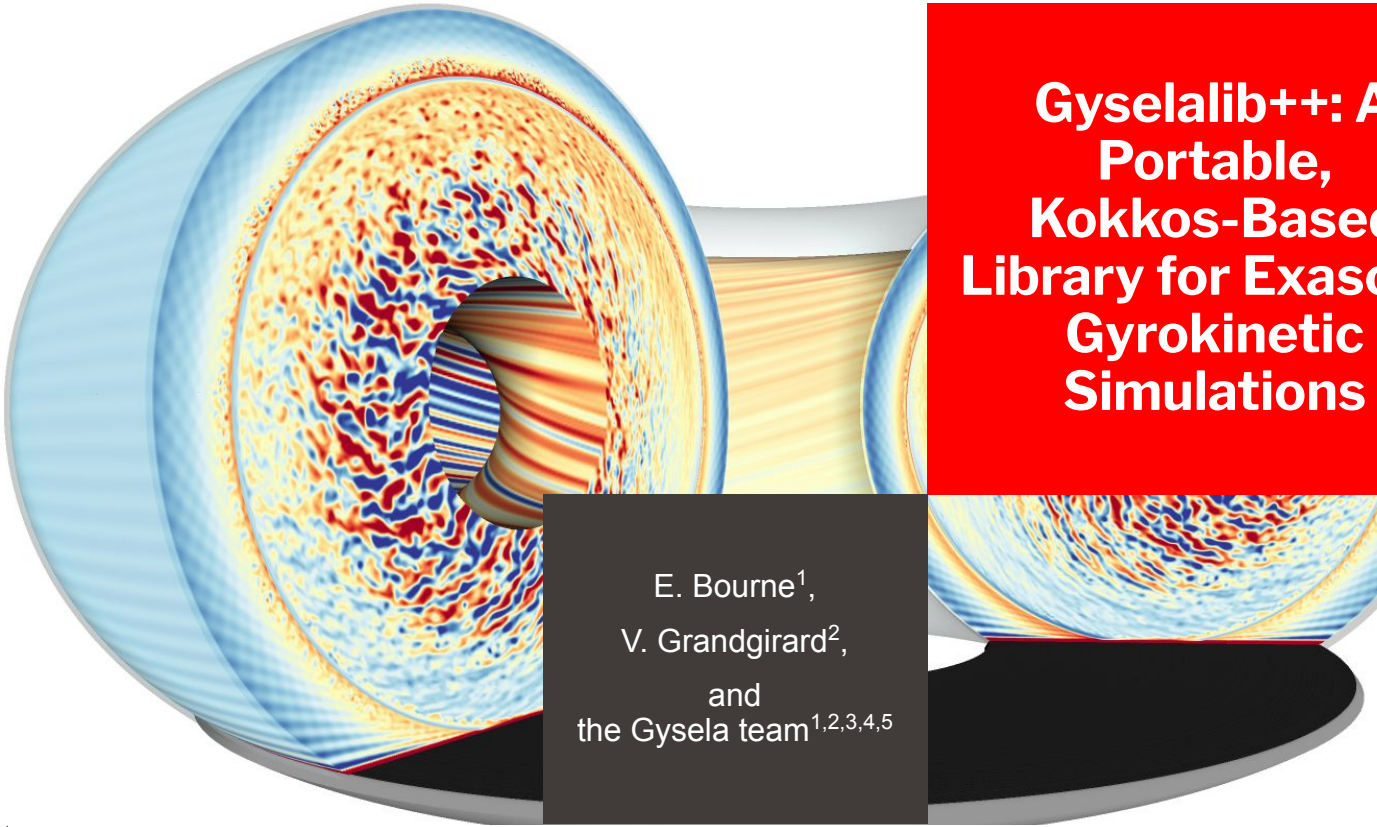
Gyselalib++: A Portable, Kokkos-Based Library for Exascale Gyrokinetic Simulations



E. Bourne[1],

V. Grandgirard[2],

and

the Gysela team[1,2,3,4,5]

[1] SCITAS, EPFL, CH-1015 Lausanne, Switzerland
[2] CEA, IRFM, 13108 Saint-Paul-lez-Durance Cedex, France
[3] Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, 91191, Gif-sur-Yvette, France
[4] Max-Planck-Institut für Plasmaphysik, Garching, Germany
[5] CINES, France

GYSELA: Exascale needs

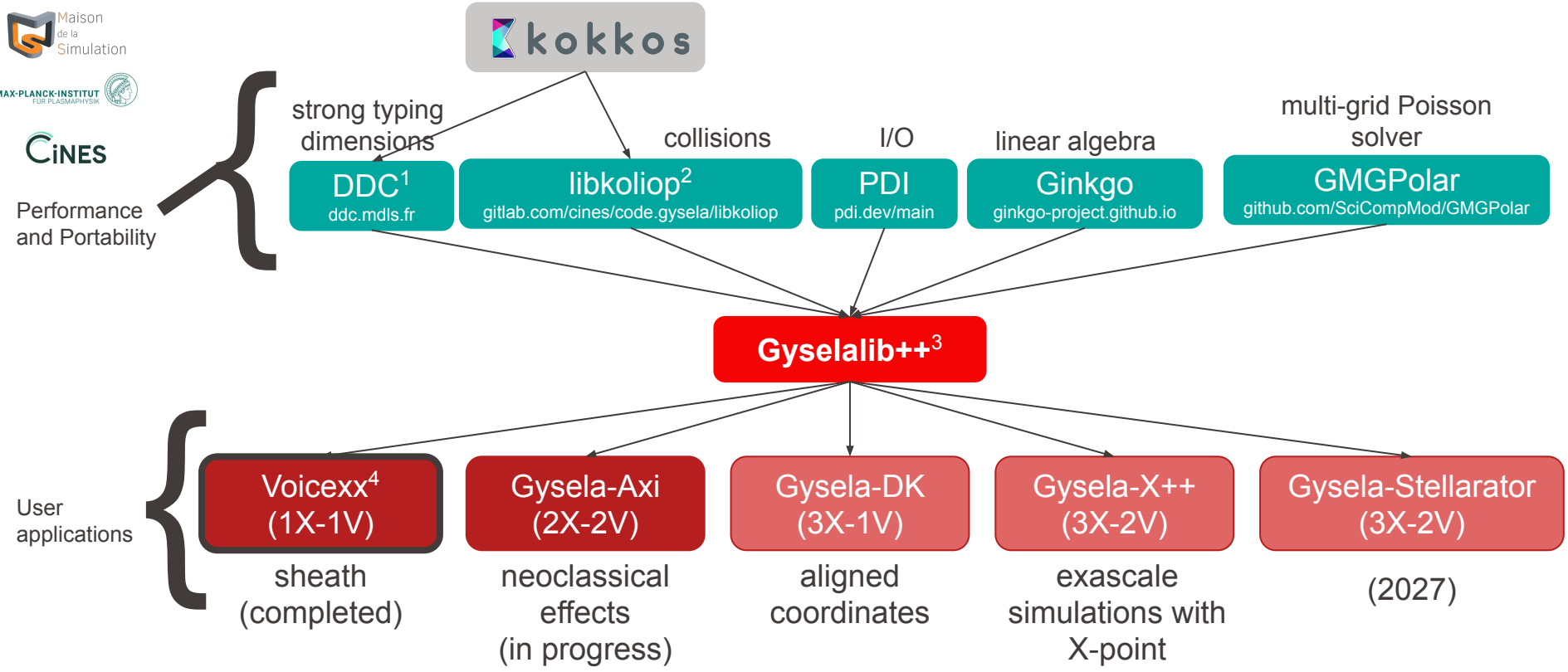
GYSELA (gyselax.github.io) :

- Non-linear 5D simulations (3D in space + 2D in velocity)
- Multi-scale problem in space and time
- Unique gyrokinetic code based on a semi-Lagrangian scheme modelling both core & edge plasmas
- Intensive use of petascale resources: ~ 150 Mhours / year
- Optimised up to 730k CPU cores
- Relative efficiency of 85% on more than 500k cores and 63% on 730k cores on CEA-HF (AMD EPYC 7763)
- Exascale needs for ITER plasma turbulence simulation with electromagnetic effects

Incremental upgrades of the code are no longer adequate to tackle upcoming challenges:

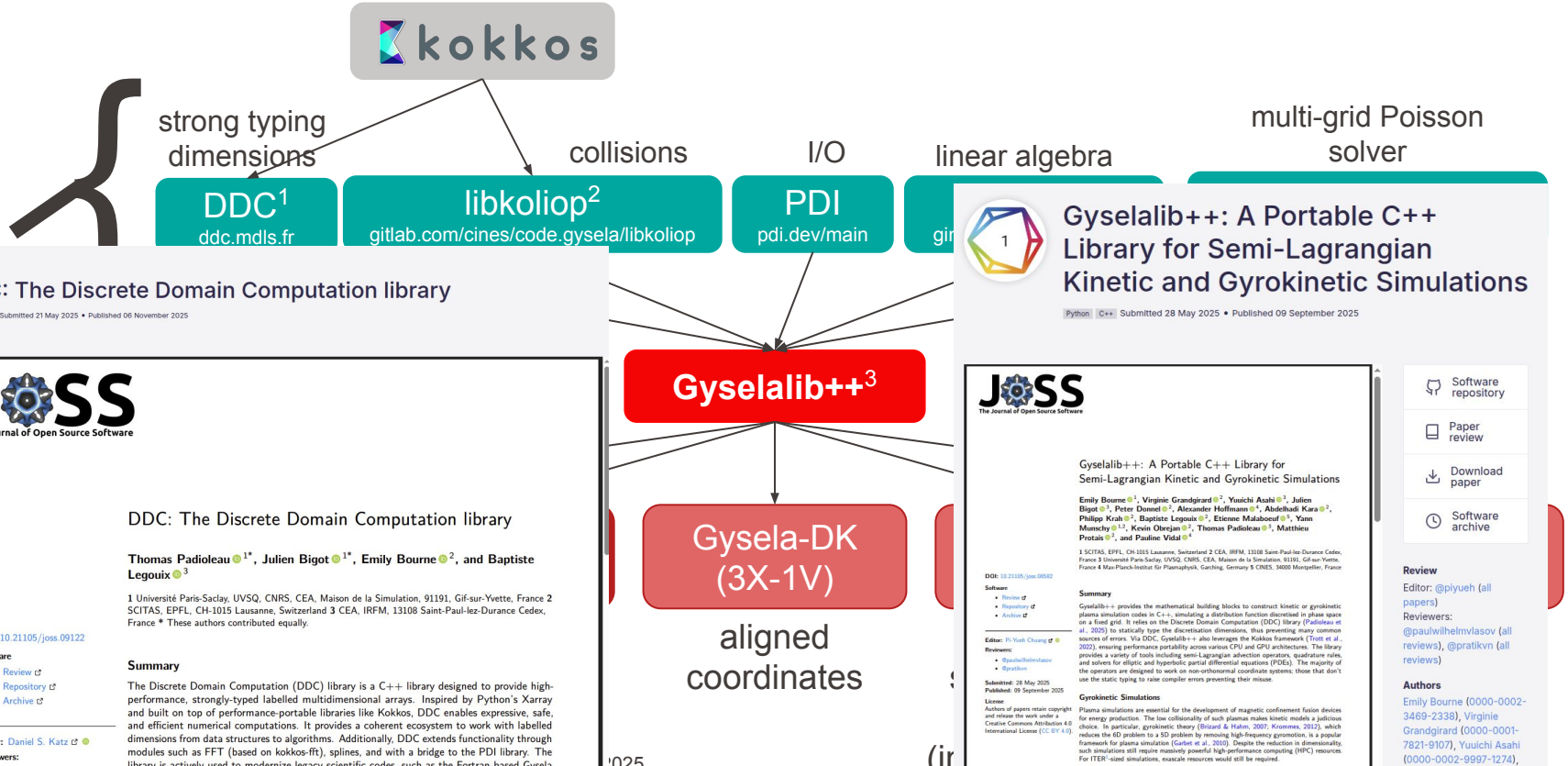
- Non-uniform points (multi-scale physics)
- X-point
- Effective use of GPUs

Gyselalib++ : A new modular library in C++



[1] T. Padioleau et al. *DDC: The Discrete Domain Computation library*, JOSS, 2025
[2] P. Donnel et al. *A multi-species collisional operator for full-F global gyrokinetics codes: Numerical aspects and verification with the GYSELA code*, CPC 2019
[3] E. Bourne et al. *Gyselalib++: A portable c++ library for semi-lagrangian kinetic and gyrokinetic simulations*, JOSS 2025
[4] E. Bourne et al. *Non-uniform splines for semi-Lagrangian kinetic simulations of the plasma sheath*, JCP 2023

Gyselalib++ : A new modular library in C++



DDC: The Discrete Domain Computation library

C++ Python Submitted 21 May 2025 • Published 06 November 2025

JOSS The Journal of Open Source Software

DDC: The Discrete Domain Computation library

Thomas Padioleau[1*], Julien Bigot[1*], Emily Bourne[2], and Baptiste Legouix[3]

1 Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, 91191, Gif-sur-Yvette, France 2 SCITAS, EPFL, CH-1015 Lausanne, Switzerland 3 CEA, IRFM, 13108 Saint-Paul-lez-Durance Cedex, France * These authors contributed equally.

DOI: 10.21105/joss.09122

Software
- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

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

Gyselalib++: A Portable C++ Library for Semi-Lagrangian Kinetic and Gyrokinetic Simulations

Python C++ Submitted 28 May 2025 • Published 09 September 2025

JOSS The Journal of Open Source Software

Gyselalib++: A Portable C++ Library for Semi-Lagrangian Kinetic and Gyrokinetic Simulations

Emily Bourne[1], Virginie Grandgirard[2], Yuuichi Asahi[3], Julien Bigot[3], Peter Donnel[2], Alexander Hoffmann[4], Abdelhadi Kara[2], Philipp Krah[2], Baptiste Legouix[2], Etienne Malaboeuf[5], Yann Munschy[1,2], Kevin Obrejan[2], Thomas Padioleau[3], Matthieu Protais[2], and Pauline Vidal[4]

1 SCITAS, EPFL, CH-1015 Lausanne, Switzerland 2 CEA, IRFM, 13108 Saint-Paul-lez-Durance Cedex, France 3 Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, 91191, Gif-sur-Yvette, France 4 Max-Planck-Institut für Plasmaphysik, Garching, Germany 5 CINES, 34000 Montpellier, France

DOI: 10.21105/joss.08582

Software
- Review
- Repository
- Archive

Editor: Pi-Yueh Chuang

Reviewers:
- @paulwilhelmvlasov
- @pratikvn

Submitted: 28 May 2025
Published: 09 September 2025

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

Summary

Gyselalib++ provides the mathematical building blocks to construct kinetic or gyrokinetic plasma simulation codes in C++, simulating a distribution function discretised in phase space on a fixed grid. It relies on the Discrete Domain Computation (DDC) library (Padioleau et al., 2025) to statically type the discretisation dimensions, thus preventing many common sources of errors. Via DDC, Gyselalib++ also leverages the Kokkos framework (Trott et al., 2022), ensuring performance portability across various CPU and GPU architectures. The library provides a variety of tools including semi-Lagrangian advection operators, quadrature rules, and solvers for elliptic and hyperbolic partial differential equations (PDEs). The majority of the operators are designed to work on non-orthonormal coordinate systems; those that don't use the static typing to raise compiler errors preventing their misuse.

Gyrokinetic Simulations

Plasma simulations are essential for the development of magnetic confinement fusion devices for energy production. The low collisionality of such plasmas makes kinetic models a judicious choice. In particular, gyrokinetic theory (Brizard & Hahn, 2007; Krommes, 2012), which reduces the 6D problem to a 5D problem by removing high-frequency gyromotion, is a popular framework for plasma simulation (Garbet et al., 2010). Despite the reduction in dimensionality, such simulations still require massively powerful high-performance computing (HPC) resources. For ITER-sized simulations, exascale resources would still be required.

The pre-existing GYSELA code (Grandgirard et al., 2016), written in Fortran, originally aimed to simulate plasma in the core region of a tokamak using semi-Lagrangian advection with a distribution function discretised in phase space on a uniform grid. This approach was shown to work well, and it allowed the study of many interesting physical phenomena (Dif-Pradalier et al., 2022; Estève et al., 2018; Sarazin et al., 2021). However, expanding this code to use more complex mathematical methods such as non-uniform points (vital for handling the different magnitudes of physical quantities in the core and edge regions), and increasingly complex geometries (such as D-shape geometries, geometries including both open and closed field lines,

Software repository
Paper review
Download paper
Software archive

Review
Editor: @piyueh (all papers)
Reviewers: @paulwilhelmvlasov (all reviews), @pratikvn (all reviews)

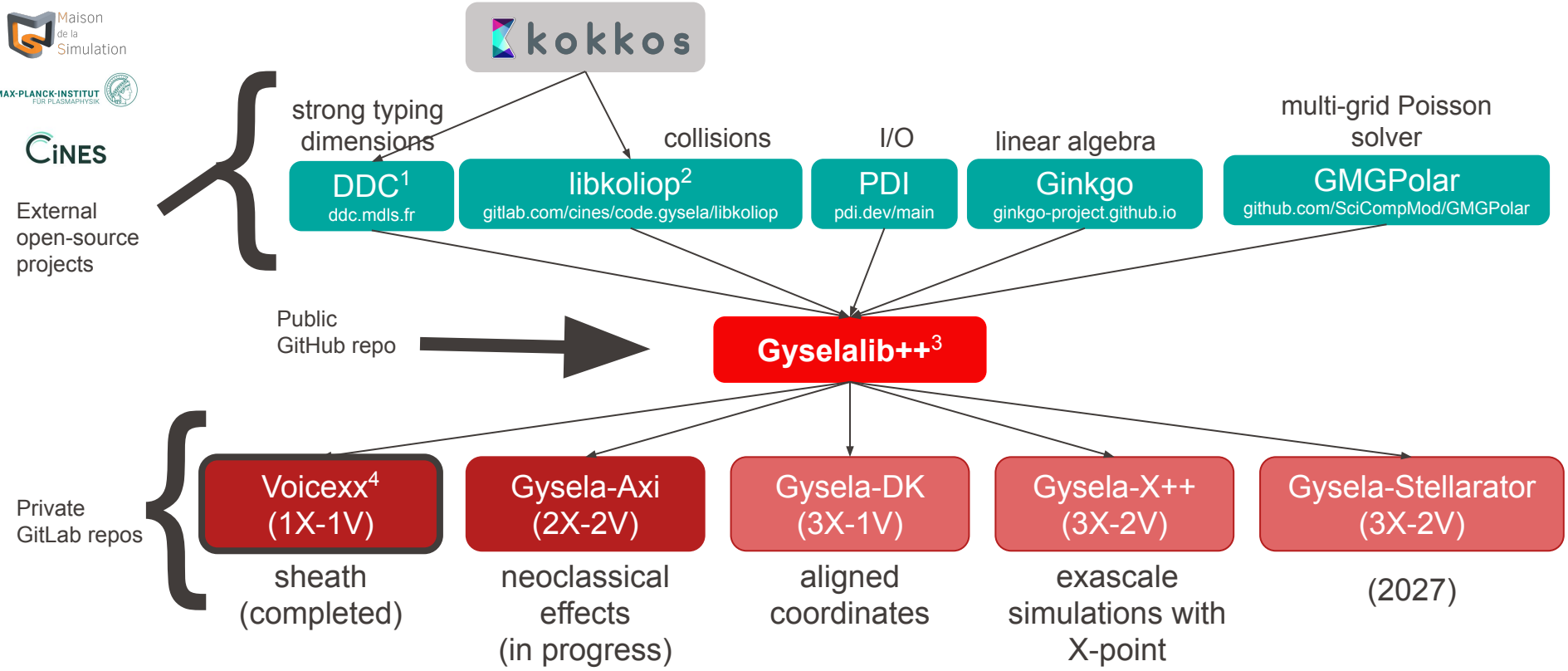
Authors
Emily Bourne (0000-0002-3469-2338), Virginie Grandgirard (0000-0001-7821-9107), Yuuichi Asahi (0000-0002-9997-1274), Julien Bigot (0000-0002-0015-4304), Peter Donnel (0000-0002-6669-416X), Alexander Hoffmann (0009-0005-0736-6122),

[2] P. Donnel et al. *A multi-species collisional operator for full-F global gyrokinetics codes: Numerical aspects and verifi*

[3] E. Bourne et al. *Gyselalib++: A portable c++ library for semi-lagrangian kinetic and gyrokinetic simulations*, JOSS 20

[4] E. Bourne et al. *Non-uniform splines for semi-Lagrangian kinetic simulations of the plasma sheath*, JCP 2023

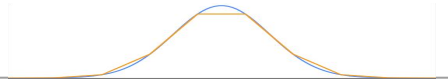
Gyselalib++ : A new modular library in C++



[1] T. Padioleau et al. *DDC: The Discrete Domain Computation library*, JOSS, 2025
[2] P. Donnel et al. *A multi-species collisional operator for full-F global gyrokinetics codes: Numerical aspects and verification with the GYSELA code*, CPC 2019
[3] E. Bourne et al. *Gyselalib++: A portable c++ library for semi-lagrangian kinetic and gyrokinetic simulations*, JOSS 2025
[4] E. Bourne et al. *Non-uniform splines for semi-Lagrangian kinetic simulations of the plasma sheath*, JCP 2023

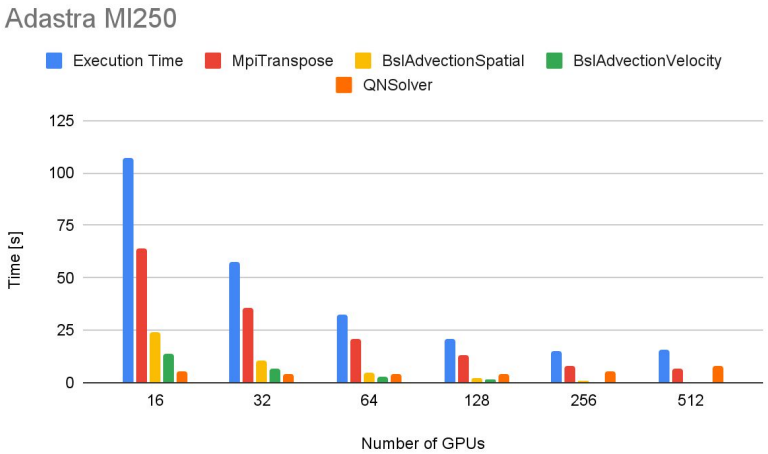
Gyselalib++: Mathematical Operators

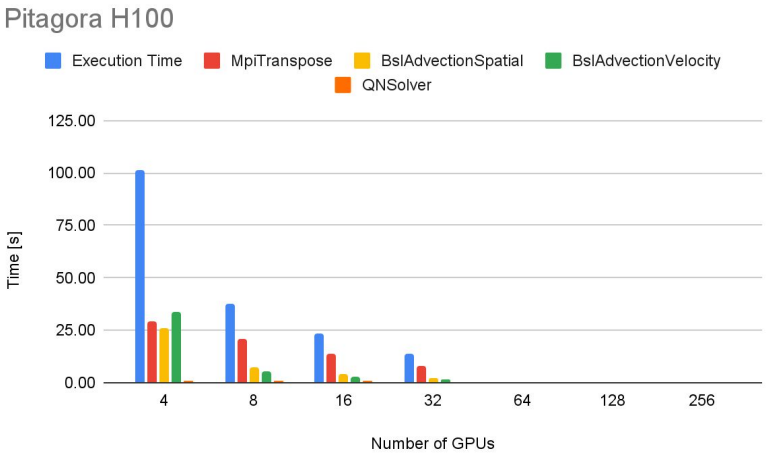
- Gyselalib++ is a library not a simulation. It provides the building blocks for both 5D gyrokinetic simulations and reduced models.
- Geometry specific tools can be added to complete the simulation

Operator Type	Variants		Example applications
Advection	• 1D Backward Semi-Lagrangian	• 2D Backward Semi-Lagrangian on a polar plane	$\partial_t f + A\partial_x f = 0$ $\partial_t f + \nabla_{r,\theta} f = 0$
Collision	• 5D multi-species collision operator valid in a gyrokinetic framework		$\frac{\partial F_a}{\partial t} = \sum_b C_{ab}(F_a, F_b)$
Interpolation	• Spline interpolation • Lagrange interpolation		
Poisson solver	• 1D FFT • 1D FEM	• 2D FEM on a polar plane	$\partial_x^2 \phi = \rho$ $-\nabla \cdot (\alpha \nabla \phi) + \beta\phi = \rho$
Quadrature	• Trapezoid • Simpson	• Defined from splines	$\int f(x)dx \approx \sum_i q_i f(x_i)$
Timestepper	• RK2 • RK3	• RK4 • Crank-Nicolson	$\partial_t y(t) = f(t, y(t))$

Performance : 2X-2V Landau damping

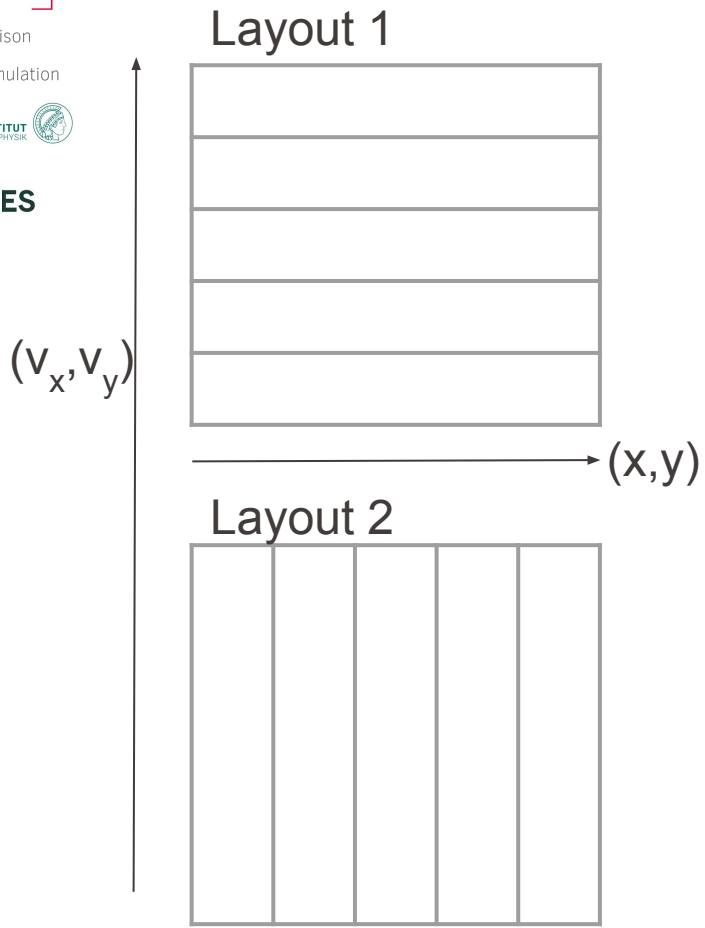
- Tests performed Mar. 2025 on Adastra AMD MI250 (CINES)
- Tests performed Jun. 2025 on Pitagora H100 (Cineca)
- **Preliminary** results - to be improved (256 x 256 x 256 x 256)





- MPI is a bottleneck

Gyselalib++ : MPI Scheme



- Spline interpolations are global operations
- Other dimensions are distributed

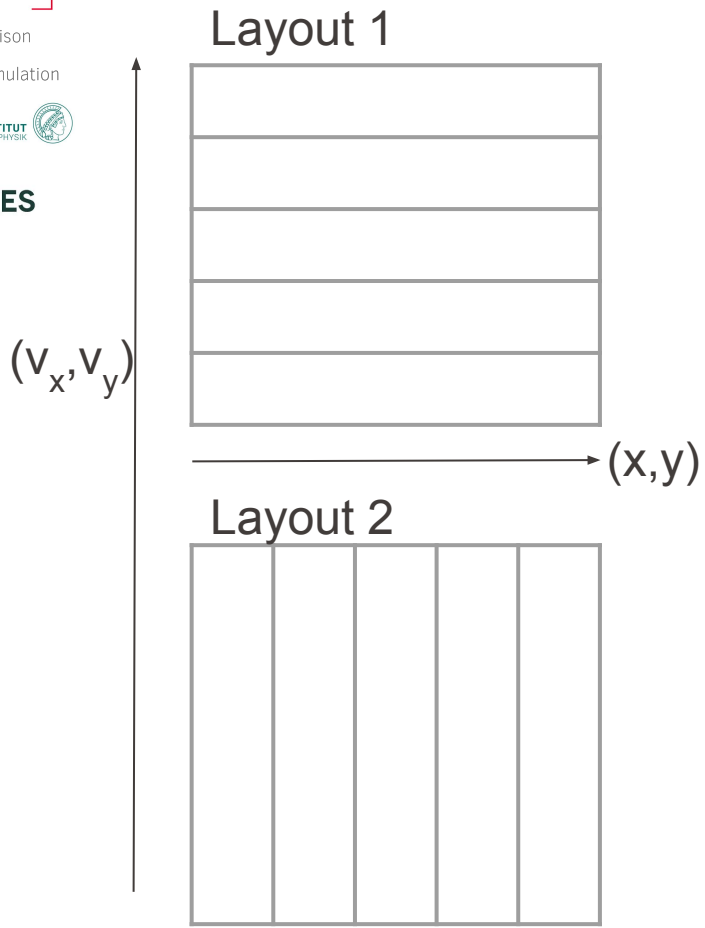
Advantages

- Operate on contiguous data
- Lots of parallelism
- Large data chunks

Disadvantages

- Costly alltoall operations
- Multiple MPI calls per time step

Gyselalib++ : MPI Scheme



Alternatives and trade-offs

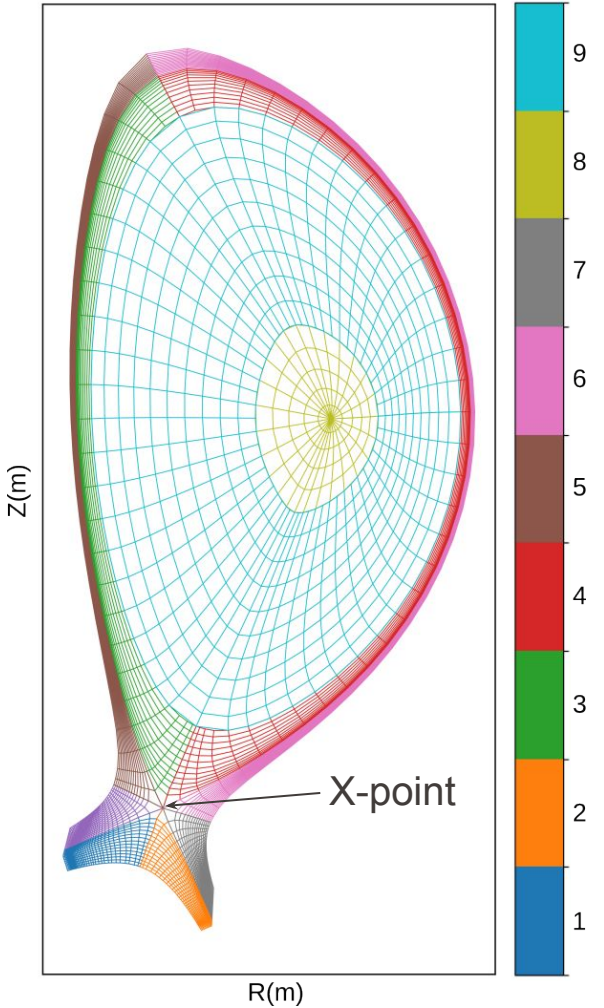
- Transpose 2 dims instead of all dims
  - More MPI calls but smaller communication groups (MPI_Cart)
- Lagrange interpolation?
  - Not stable for all cases?
  - Usable in some directions?
- Local splines?
  - Needed anyway for multi-patch methods for X-point
- Any other ideas?

Handling an X-point

Patches are required to manage the X-point geometry along field lines.

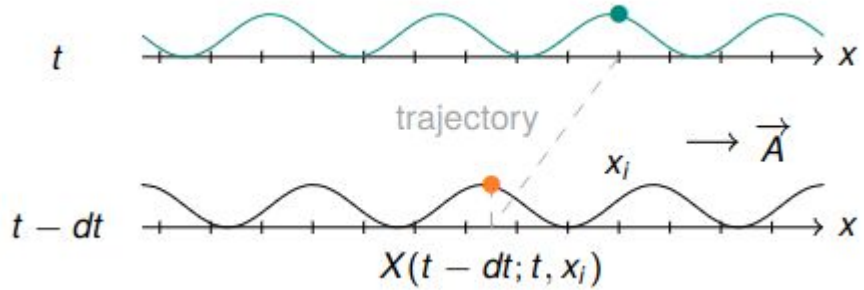
There are 2 main axes of research to handle this:

- Semi-Lagrangian Advection
  - P. Vidal (IPP) PhD 2022-2025
- Poisson solver
  - A. Hoffmann (IPP) PhD 2023-2026
    - Multi-patch (CONGA) FEEC solver
  - HyTeg A. Dasari (CERFACS) PhD 2024-2028
  - Without patches as a starting point:
    - GMGPolar coupling (CERFACS)
    - Porting of Zoni's polar spline solver

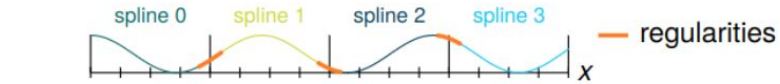


Semi-Lagrangian scheme on multi-patch

Semi-Lagrangian advection



Interpolate the function at t−dt at the feet of the characteristics: *X(t−dt; t, xi)*



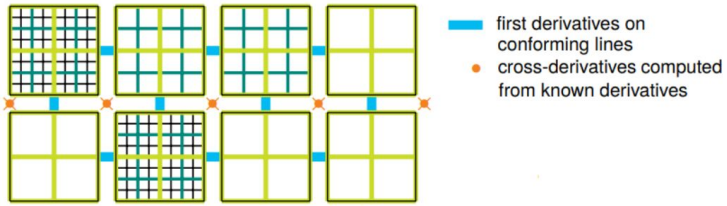
! $\mathcal{C}^0 \rightarrow$ instabilities.
▶ $\mathcal{C}^1 \rightarrow$ Hermite boundary conditions.

Spline reconstruction relies on reconstruction of derivatives

Semi-Lagrangian multi-patch advection

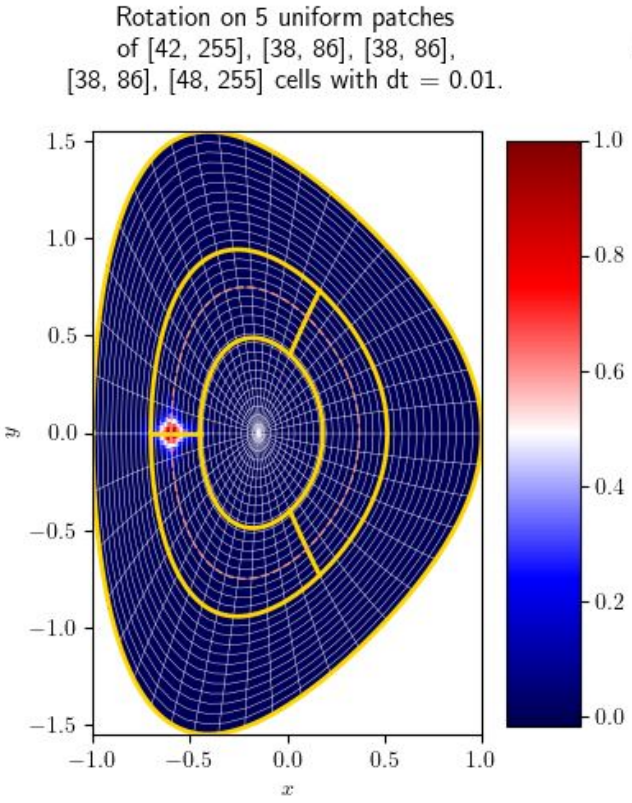
- Strang splitting is used to break the problem into 1D or 2D advections
- Patch-wise spline interpolation with treatment of interfaces

  P. Vidal et al. (2025) *Local cubic spline interpolation for Vlasov-type equations on a multi-patch geometry.* arXiv preprint arXiv:2505.22078



- Local spline interpolation is carried out by DDC
  - Batched LAPACK solvers upstreamed to Kokkos Kernels

    Asahi, Y. et al. (2024) *Development of performance portable spline solver for exa-scale plasma turbulence simulation.* SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE.
  - Ginkgo iterative solvers



Conclusion : Gyselalib++ a growing library

- Gyselalib++ already contains many of the basic mathematical operators required for plasma simulations
- DDC's static typing prevents many common bugs
- Gyselalib++ is already used in multiple simulations with a variety of dimensionalities
- The code is shown to be portable on multiple hardware types:
  - CPU
  - AMD GPU
  - Nvidia GPU
- Developments for more complex simulations are well under way:
  - Axi-symmetric
  - 5D
  - X-point
  - Stellarators

References

E. Bourne et al. (2025) *Gyselalib++: A Portable C++ Library for Semi-Lagrangian Kinetic and Gyrokinetic Simulations*. Journal of Open Source Software (JOSS)

T. Padioleau et al. (2025) *DDC: The Discrete Domain Computation library*. Journal of Open Source Software (JOSS)

P. Donnel et al.(2019) *A multi-species collisional operator for full-F global gyrokinetics codes: Numerical aspects and verification with the GYSELA code.* Computer Physics Communications (CPC)

E. Bourne et al. (2023) *Non-uniform splines for semi-Lagrangian kinetic simulations of the plasma sheath.* Journal of Computational Physics (JCP).

P. Vidal et al. (2025) *Local cubic spline interpolation for Vlasov-type equations on a multi-patch geometry.* arXiv preprint arXiv:2505.22078

Li, Y., et al. (2025). *Semi-Lagrangian methods of a plasma hybrid model with multi-species kinetic ions and massless electrons*. arXiv preprint arXiv:2504.04282

Asahi, Y. et al. (2025) *kokkos-fft: A shared-memory FFT for the Kokkos ecosystem*. Journal of Open Source Software (JOSS)

Mathematically Typed Data with DDC

- DDC (https://ddc.mdls.fr/) is a wrapper around **Kokkos** developed by Maison de la Simulation
- Kokkos allows us to have a single code base for multiple platforms (CPU/Nvidia GPU/AMD GPU/etc)
  - Strong interaction with the Kokkos development team (Moonshot CExA)
- DDC statically types the physical dimensions

  Example:

```
double integrate_over_vx(DConstFieldVx field, DConstFieldVx quad_coeffs) {
    return ddc::parallel_transform_reduce(Kokkos::DefaultExecutionSpace(),
        get_idx_range(integrated_field),
        0.0,
        ddc::reducer::sum<double>(),
        KOKKKOS_LAMBDA(IdxVx idx) {
            return field(idx) * quad_coeffs(idx);
        });
}
```

- This ensures that dangerous typos cause compilation errors

  Example:

```
double integral_value = integrate_over_vx(field_on_vx, quad_coeffs_vy);
```

Applications Using Gyselalib++

Voicexx

- Implements the simulation described in
  E. Bourne et al. (2023) *Non-uniform splines for semi-Lagrangian kinetic simulations of the plasma sheath.* Journal of Computational Physics (JCP).
- Currently being used to study plasma-neutral interactions (TSVV4)
  M. Protais (CEA) PhD 2025-2028

Diocotron

- Used to test mathematical methods
  P. Vidal et al. (2025) *Local cubic spline interpolation for Vlasov-type equations on a multi-patch geometry.* arXiv preprint arXiv:2505.22078

Hybrid 4D model

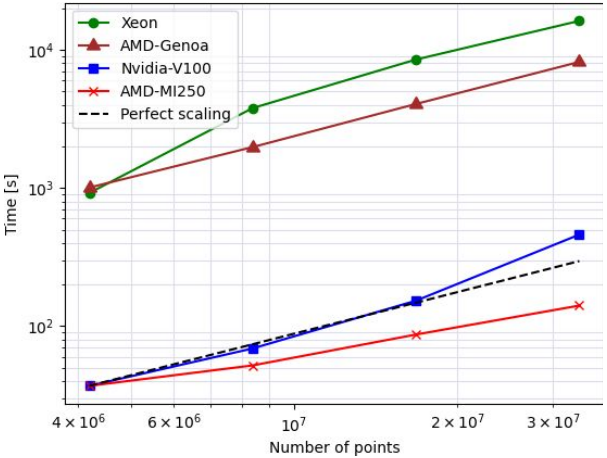
- Li, Y., et al. (2025). *Semi-Lagrangian methods of a plasma hybrid model with multi-species kinetic ions and massless electrons*. arXiv preprint arXiv:2504.04282

Gysela-Axisymmetric (2X-2V)

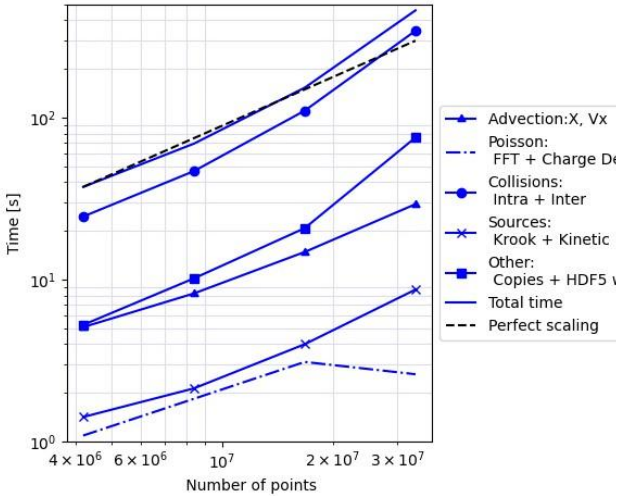
- In progress
- Highlights ease with which we will be able to go to 5D

Performance : 1X-1V Sheath simulation

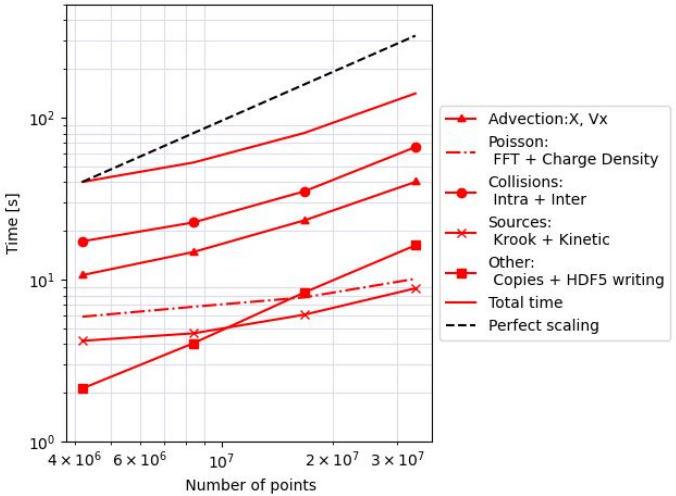
- See [E.Bourne et al. 2023] for numerics and [Y.Munschy et al. 2023] for physics
- Tests performed Oct. 2024
- Single GPU simulations (No MPI)



CPU versus GPU



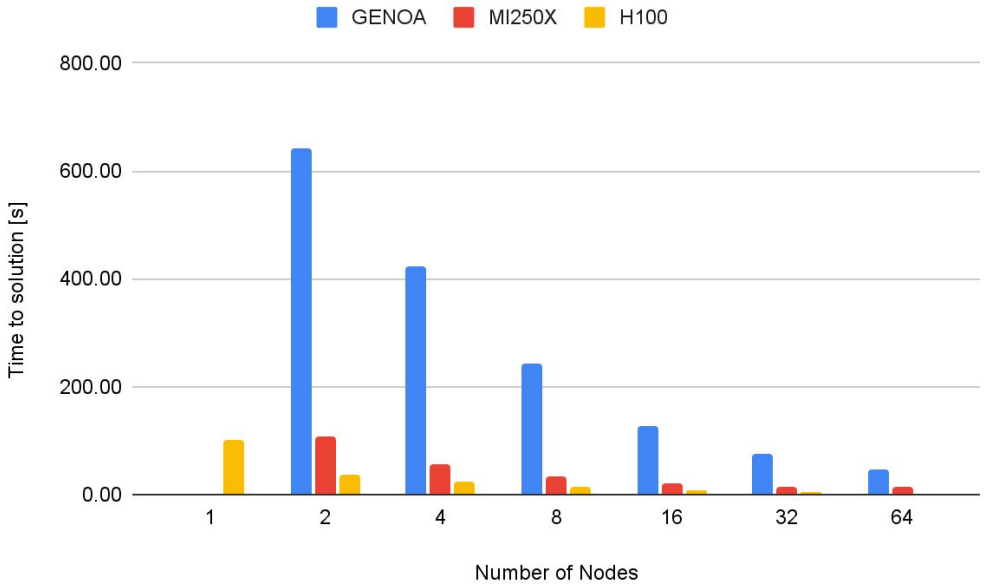
GPU: NVIDIA V100



GPU: AMD-MI250X

Performance : 2X-2V Landau damping

- Code is portable to various different hardware
- Tests performed Mar. 2025 on Adastra AMD MI250 (CINES)
- Tests performed Jun. 2025 on Pitagora H100 (Cineca)
- **Preliminary** results - to be improved (256 x 256 x 256 x 256)



Gyselalib++ : A collaborative open source project

github.com/gyselax/gyselalibxx/
gyselax.github.io/gyselalibxx/

Funded and supported by multiple collaborations:

- Moonshot CExA → Development of new functionalities in Kokkos (e.g. KokkosFFT and KokkosKernels)
- EoCoE-III → Mathematics expertise (IPP), HPC Optimisation (FAU), Linear solvers (CERFACS)
- PEPR NumPEx (Exa-DoST) → I/O Optimisation, in situ diagnostics and dataworkflows for IA

Gyselalib++ : Software Design

- Gyselalib++ contains the building blocks for plasma simulations
- It is modular so mathematical choices can be changed without rewriting the methods that use them
- All mathematical choices should appear in the main simulation file

```
SplineXBuilder const builder_x(idx_range_x_vx);
SplineVxBuilder const builder_vx(idx_range_x_vx);

// Initialisation of the equilibrium function
MaxwellianEquilibrium const init_fequilibrium
        = MaxwellianEquilibrium::init_from_input(idx_range_kinsp, conf_gyselalibxx);
init_fequilibrium(get_field(allfequilibrium));

double time_start(0);
SingleModePerturbInitialisation const init
        = SingleModePerturbInitialisation::init_from_input(
                get_const_field(allfequilibrium),
                idx_range_kinsp,
                conf_gyselalibxx);
init(get_field(allfdistribu));
auto allfequilibrium_host = ddc::create_mirror_view_and_copy(get_field(allfequilibrium));

ddc::ConstantExtrapolationRule<X> bv_x_min(ddc::coordinate(idx_range_x.front()));
ddc::ConstantExtrapolationRule<X> bv_x_max(ddc::coordinate(idx_range_x.back()));

// Creating operators
SplineXEvaluator const spline_x_evaluator(bv_x_min, bv_x_max);
PreallocatableSplineInterpolator const spline_x_interpolator(builder_x, spline_x_evaluator);

ddc::ConstantExtrapolationRule<Vx> bv_v_min(ddc::coordinate(idx_range_vx.front()));
ddc::ConstantExtrapolationRule<Vx> bv_v_max(ddc::coordinate(idx_range_vx.back()));

SplineVxEvaluator const spline_vx_evaluator(bv_v_min, bv_v_max);
PreallocatableSplineInterpolator const spline_vx_interpolator(builder_vx, spline_vx_evaluator);

BslAdvectionSpatial<GeometryXVx, GridX> const advection_x(spline_x_interpolator);
BslAdvectionVelocity<GeometryXVx, GridVx> const advection_vx(spline_vx_interpolator);

SplitVlasovSolver const vlasov(advection_x, advection_vx);

DFieldMemVx const quadrature_coeffs(neumann_spline_quadrature_coefficients<
                        Kokkos::DefaultExecutionSpace>(idx_range_vx, builder_vx));

ChargeDensityCalculator rhs(get_field(quadrature_coeffs));
FFTPoissonSolver<IdxRangeX> fft_poisson_solver(idx_range_x);
QNSolver const poisson(fft_poisson_solver, rhs);

PredCorr const predcorr(vlasov, poisson);

predcorr(get_field(allfdistribu), time_start, deltat, nbiter);
```

Gyselalib++ : Software Design

Example:

- We use the Backward SemiLagrangian method

```
SplineXBuilder const builder_x(idx_range_x_vx);
SplineVxBuilder const builder_vx(idx_range_x_vx);

// Initialisation of the equilibrium function
MaxwellianEquilibrium const init_fequilibrium
        = MaxwellianEquilibrium::init_from_input(idx_range_kinsp, conf_gyselalibxx);
init_fequilibrium(get_field(allfequilibrium));

double time_start(0);
SingleModePerturbInitialisation const init
        = SingleModePerturbInitialisation::init_from_input(
                get_const_field(allfequilibrium),
                idx_range_kinsp,
                conf_gyselalibxx);
init(get_field(allfdistribu));
auto allfequilibrium_host = ddc::create_mirror_view_and_copy(get_field(allfequilibrium));

ddc::ConstantExtrapolationRule<X> bv_x_min(ddc::coordinate(idx_range_x.front()));
ddc::ConstantExtrapolationRule<X> bv_x_max(ddc::coordinate(idx_range_x.back()));

// Creating operators
SplineXEvaluator const spline_x_evaluator(bv_x_min, bv_x_max);
PreallocatableSplineInterpolator const spline_x_interpolator(builder_x, spline_x_evaluator);

ddc::ConstantExtrapolationRule<Vx> bv_v_min(ddc::coordinate(idx_range_vx.front()));
ddc::ConstantExtrapolationRule<Vx> bv_v_max(ddc::coordinate(idx_range_vx.back()));

SplineVxEvaluator const spline_vx_evaluator(bv_v_min, bv_v_max);
PreallocatableSplineInterpolator const spline_vx_interpolator(builder_vx, spline_vx_evaluator);

BslAdvectionSpatial<GeometryXVx, GridX> const advection_x(spline_x_interpolator);
BslAdvectionVelocity<GeometryXVx, GridVx> const advection_vx(spline_vx_interpolator);

SplitVlasovSolver const vlasov(advection_x, advection_vx);

DFieldMemVx const quadrature_coeffs(neumann_spline_quadrature_coefficients<
                        Kokkos::DefaultExecutionSpace>(idx_range_vx, builder_vx));

ChargeDensityCalculator rhs(get_field(quadrature_coeffs));
FFTPoissonSolver<IdxRangeX> fft_poisson_solver(idx_range_x);
QNSolver const poisson(fft_poisson_solver, rhs);

PredCorr const predcorr(vlasov, poisson);

predcorr(get_field(allfdistribu), time_start, deltat, nbiter);
```

Gyselalib++ : Software Design

Example:

- We use the Backward SemiLagrangian method
- With a spline interpolation

```
SplineXBuilder const builder_x(idx_range_x_vx);
SplineVxBuilder const builder_vx(idx_range_x_vx);

// Initialisation of the equilibrium function
MaxwellianEquilibrium const init_fequilibrium
        = MaxwellianEquilibrium::init_from_input(idx_range_kinsp, conf_gyselalibxx);
init_fequilibrium(get_field(allfequilibrium));

double time_start(0);
SingleModePerturbInitialisation const init
        = SingleModePerturbInitialisation::init_from_input(
                get_const_field(allfequilibrium),
                idx_range_kinsp,
                conf_gyselalibxx);
init(get_field(allfdistribu));
auto allfequilibrium_host = ddc::create_mirror_view_and_copy(get_field(allfequilibrium));

ddc::ConstantExtrapolationRule<X> bv_x_min(ddc::coordinate(idx_range_x.front()));
ddc::ConstantExtrapolationRule<X> bv_x_max(ddc::coordinate(idx_range_x.back()));

// Creating operators
SplineXEvaluator const spline_x_evaluator(bv_x_min, bv_x_max);
PreallocatableSplineInterpolator const spline_x_interpolator(builder_x, spline_x_evaluator);

ddc::ConstantExtrapolationRule<Vx> bv_v_min(ddc::coordinate(idx_range_vx.front()));
ddc::ConstantExtrapolationRule<Vx> bv_v_max(ddc::coordinate(idx_range_vx.back()));

SplineVxEvaluator const spline_vx_evaluator(bv_v_min, bv_v_max);
PreallocatableSplineInterpolator const spline_vx_interpolator(builder_vx, spline_vx_evaluator);

BslAdvectionSpatial<GeometryXVx, GridX> const advection_x(spline_x_interpolator);
BslAdvectionVelocity<GeometryXVx, GridVx> const advection_vx(spline_vx_interpolator);

SplitVlasovSolver const vlasov(advection_x, advection_vx);

DFieldMemVx const quadrature_coeffs(neumann_spline_quadrature_coefficients<
                        Kokkos::DefaultExecutionSpace>(idx_range_vx, builder_vx));

ChargeDensityCalculator rhs(get_field(quadrature_coeffs));
FFTPoissonSolver<IdxRangeX> fft_poisson_solver(idx_range_x);
QNSolver const poisson(fft_poisson_solver, rhs);

PredCorr const predcorr(vlasov, poisson);

predcorr(get_field(allfdistribu), time_start, deltat, nbiter);
```

Gyselalib++ : Software Design

Example:

- We use the Backward SemiLagrangian method
- With a spline interpolation
- And a constant extrapolation when the feet of the characteristics are outside the domain

```
SplineXBuilder const builder_x(idx_range_x_vx);
SplineVxBuilder const builder_vx(idx_range_x_vx);

// Initialisation of the equilibrium function
MaxwellianEquilibrium const init_fequilibrium
        = MaxwellianEquilibrium::init_from_input(idx_range_kinsp, conf_gyselalibxx);
init_fequilibrium(get_field(allfequilibrium));

double time_start(0);
SingleModePerturbInitialisation const init
        = SingleModePerturbInitialisation::init_from_input(
                get_const_field(allfequilibrium),
                idx_range_kinsp,
                conf_gyselalibxx);
init(get_field(allfdistribu));
auto allfequilibrium_host = ddc::create_mirror_view_and_copy(get_field(allfequilibrium));

ddc::ConstantExtrapolationRule<X> bv_x_min(ddc::coordinate(idx_range_x.front()));
ddc::ConstantExtrapolationRule<X> bv_x_max(ddc::coordinate(idx_range_x.back()));

// Creating operators
SplineXEvaluator const spline_x_evaluator(bv_x_min, bv_x_max);
PreallocatableSplineInterpolator const spline_x_interpolator(builder_x, spline_x_evaluator);

ddc::ConstantExtrapolationRule<Vx> bv_v_min(ddc::coordinate(idx_range_vx.front()));
ddc::ConstantExtrapolationRule<Vx> bv_v_max(ddc::coordinate(idx_range_vx.back()));

SplineVxEvaluator const spline_vx_evaluator(bv_v_min, bv_v_max);
PreallocatableSplineInterpolator const spline_vx_interpolator(builder_vx, spline_vx_evaluator);

BslAdvectionSpatial<GeometryXVx, GridX> const advection_x(spline_x_interpolator);
BslAdvectionVelocity<GeometryXVx, GridVx> const advection_vx(spline_vx_interpolator);

SplitVlasovSolver const vlasov(advection_x, advection_vx);

DFieldMemVx const quadrature_coeffs(neumann_spline_quadrature_coefficients<
                        Kokkos::DefaultExecutionSpace>(idx_range_vx, builder_vx));

ChargeDensityCalculator rhs(get_field(quadrature_coeffs));
FFTPoissonSolver<IdxRangeX> fft_poisson_solver(idx_range_x);
QNSolver const poisson(fft_poisson_solver, rhs);

PredCorr const predcorr(vlasov, poisson);

predcorr(get_field(allfdistribu), time_start, deltat, nbiter);
```

Mathematically Typed Data

- Gyselalib++ also statically types objects that are not found in DDC.
- Gyrokinetic expressions on curvilinear geometries lead to complex error-prone tensor equations
- Example:
  - The electric field is defined as:

$$E = -\nabla\phi$$

  - If we want to calculate E on the Cartesian basis (e_x, e_y) from a gradient calculated on a curvilinear basis (b_r, b_θ) then in tensor notation we have:

$$\hat{E}^i = (J^{-1})^i_j E^j = (J^{-1})^i_j g^{jk} E_k = (J^{-1})^i_j g^{jk} \partial_k \phi$$

  - Gyselalib++ is constructed such that this expression will only compile if the bases are correctly managed

```
DVector<R_cov, Theta_cov> deriv_phi = get_derivative(phi);
DTensor<VectorIndexSet<R, Theta>, VectorIndexSet<R, Theta>> G = metric_tensor(coord);
DTensor<VectorIndexSet<X, Y>, VectorIndexSet<R, Theta>> inv_J = inv_jacobian(coord);
DVector<X, Y> E = tensor_mul(index<'i', 'j'>(inv_J), index<'j', 'k'>(G), index<'k'>(deriv_phi));
```