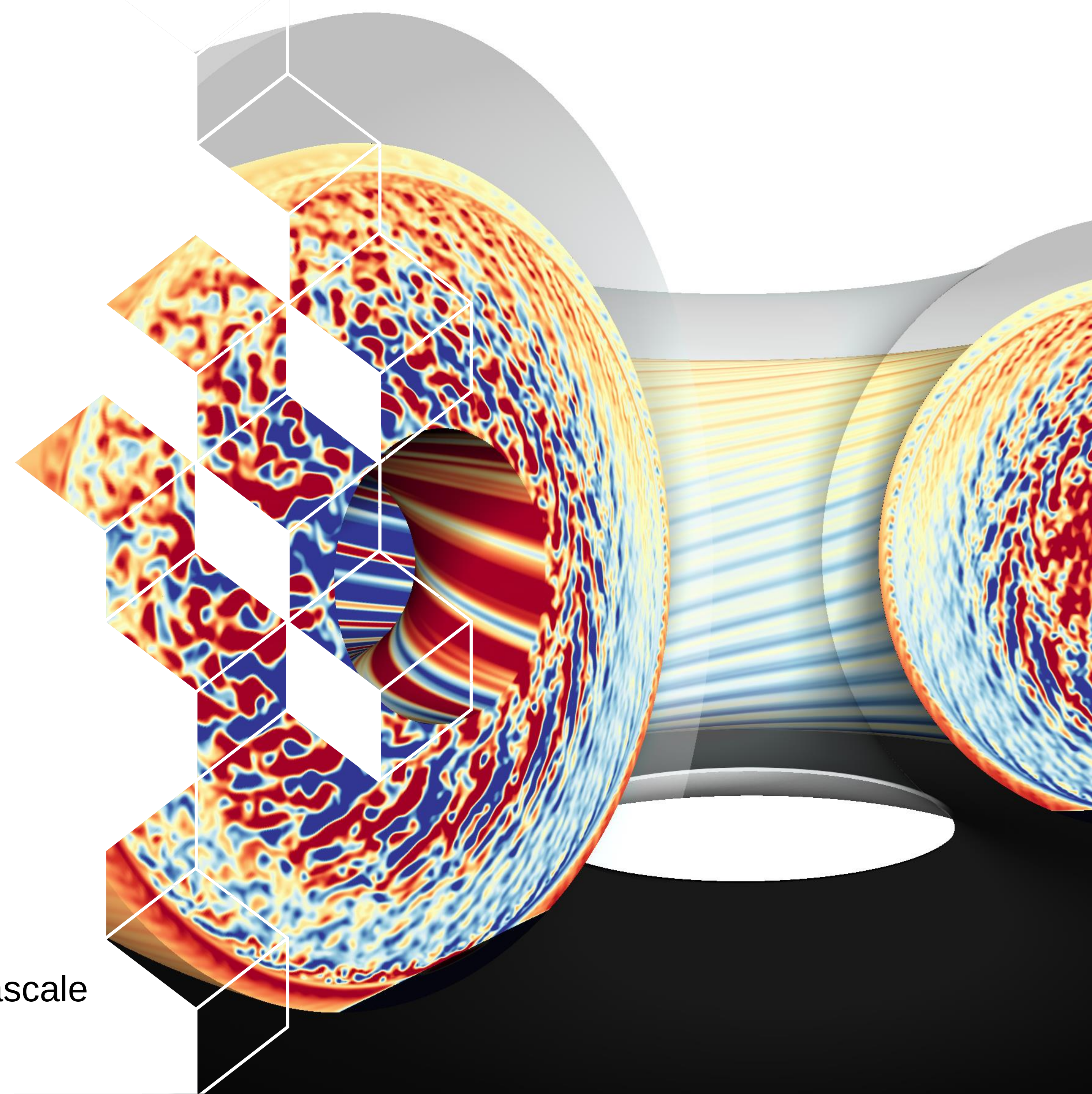
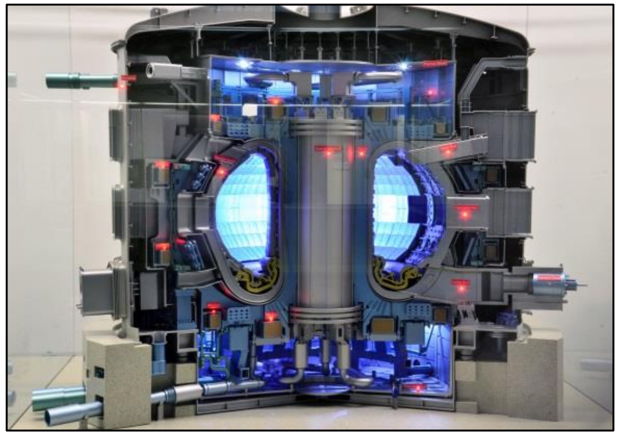
GyselaX++: Exascale Challenges for tokamak plasma turbulence simulations

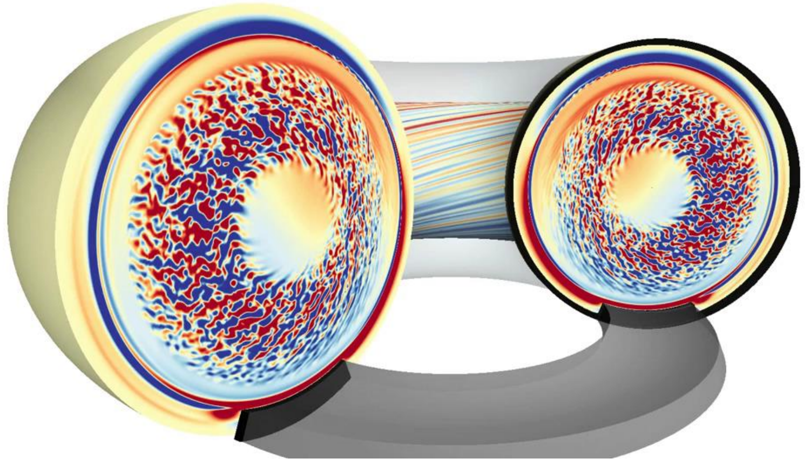
Virginie Grandgirard



Workshop Efficient discretisation for PDE@Exascale
(2023/11/08)

First principle simulations required for ITER
→ Gyrokinetic plasma turbulence simulations


ITER project


GYSELA simulation

- To optimize performance and minimize risks, each ITER scenario will have to be numerically validated.
- A complete chain of numerical tools will be required, ranging from scale models, which can be used in real time, to first-principles simulations, which are more costly but more reliable.
- Turbulent transport mainly governs confinement in Tokamaks
- Tokamak plasmas weakly collisional → Kinetic approach mandatory
  - Fusion plasma turbulence is low frequency → fast gyro-motion is averaged out
  - Gyrokinetic approach: phase space reduction from 6D to 5D

GYSELA: a highly parallelised code running at petascale

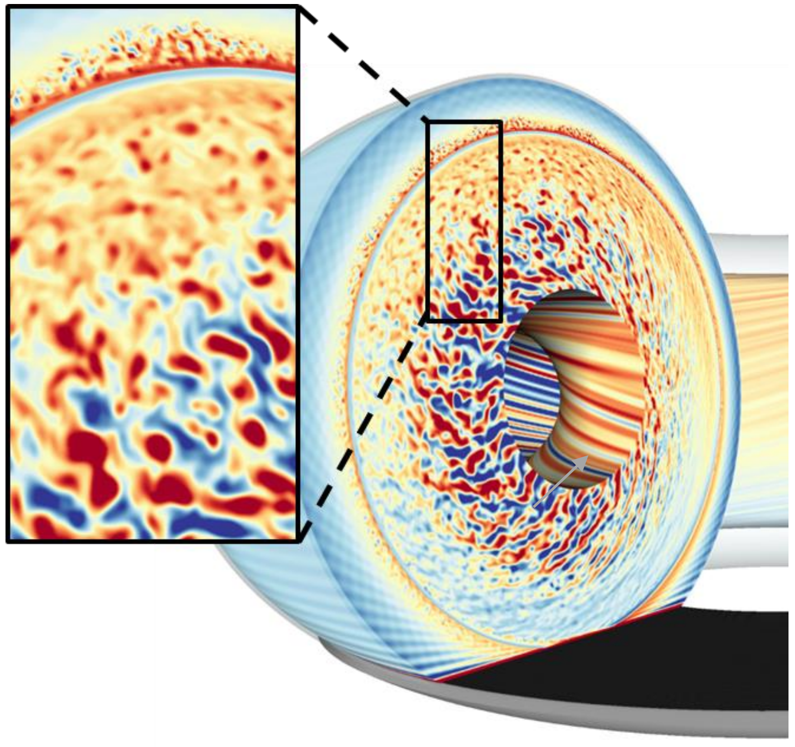
- Gyrokinetic codes require state-of-the-art HPC techniques and must run efficiently on several thousand processors
  - Non-linear 5D simulations (3D in space + 2D in velocity)

    + multi-scale problem in space and time

- GYSELA (GYrokinetic SEmi-LAgrangian) developed at IRFM/CEA for 20 years
  - Unique gyrokinetic code based on a semi-Lagrangian scheme modelling both core & edge plasmas
  - Fortran 90 + few C modules with hybrid MPI/OpenMP parallelisation

- **Intensive use of petascale resources**: ~ 150 Mhours / year
  - (GENCI + PRACE + HPC Fusion resources)

- **Exascale needs for ITER plasma turbulence** simulation with electromagnetic effects

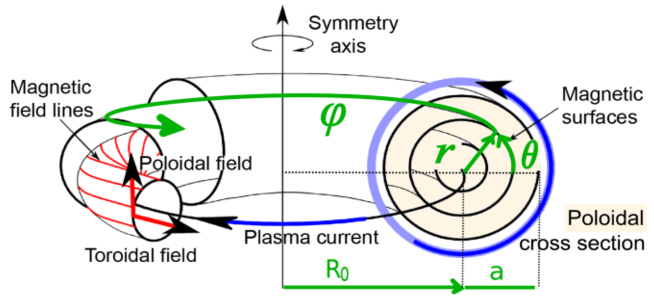


GYSELA – 5D Boltzmann equations

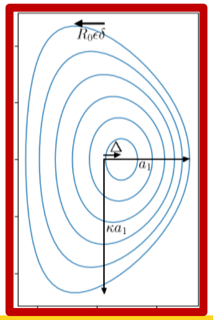
Geometry

mesh
(equidistant in (r,θ,φ))

magnetic configuration
(simplified circular concentric)



+ Culham equilibrium



Vlasov

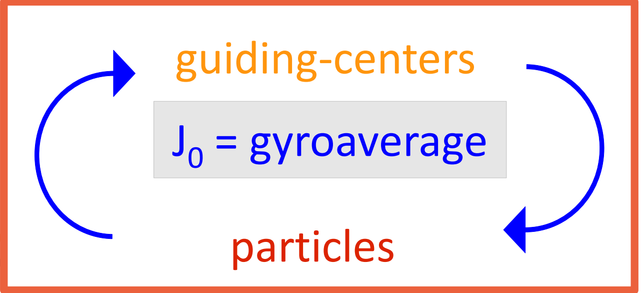
5D Vlasov solver for multi-ions + electrons (trapped or **full kinetic**)
(semi-lagrangian scheme)

+ collisions + sources

$$B^*_{\parallel s}\frac{\partial \bar{F}_s}{\partial t} + \nabla\cdot\left(\frac{d\mathbf{x_G}}{dt}B^*_{\parallel s}\bar{F}_s\right) + \frac{\partial}{\partial v_{G\parallel}}\left(\frac{dv_{G\parallel}}{dt}B^*_{\parallel s}\bar{F}_s\right) = \mathcal{C}(\bar{F}_s) + S + \mathcal{K}_{\rm buff}(\bar{F}_s) + \mathcal{D}_{\rm buff}(\bar{F}_s)$$

with the equations of motion: $B^*_{\parallel s}d_t\mathbf{x}_G = v_{G\parallel}\mathbf{B}^* + \frac{1}{e}\mathbf{b}\times\nabla\Lambda$; $B^*_{\parallel s}m_s d_t v_{G\parallel} = -\mathbf{B}^*\cdot\nabla\Lambda$

where $\mathbf{B}^* = \mathbf{B} + (m_s v_{G\parallel}/e)\nabla\times\mathbf{b}$ and $\Lambda = eJ_0\phi + \mu B$; ϕ = 3D electrostatic potential



Poisson

3D Poisson solver: 2D (r,θ) spline finite elements for each φ

$$\frac{e}{T_{e,eq}}\left(\phi - \langle\phi\rangle\right) - \frac{1}{n_{e_0}}\sum_s Z_s\nabla_\perp\cdot\left(\frac{n_{s,eq}}{B\Omega_s}\nabla_\perp\phi\right) = \frac{1}{n_{e_0}}\sum_s Z_s\int J_0\cdot\left(\bar{F}_s - \bar{F}_{s,eq}\right)d^3v$$

+ Ampere

3D equation for the potential vector A ([Finite Differences in r + Fourier in θ] for each φ)

Time-splitting for Boltzmann equation

- A time-splitting of Strang is applied to the 5D non-linear Boltzmann equation:

$$B^*_{\|s}\frac{\partial \bar{F}_s}{\partial t} + \nabla\cdot\left(\frac{d\mathbf{x_G}}{dt}B^*_{\|s}\bar{F}_s\right) + \frac{\partial}{\partial v_{G\|}}\left(\frac{dv_{G\|}}{dt}B^*_{\|s}\bar{F}_s\right) = C(\bar{F}_s) + S$$

- Let us define three advection operators (with $\mathcal{X}_G = (r,\theta)$)

$$B^*_{\|s}\frac{\partial \bar{F}_s}{\partial t} + \nabla\cdot\left(B^*_{\|s}\frac{d\mathcal{X}_G}{dt}\bar{F}_s\right) = 0 \quad : (\tilde{\mathcal{X}}_G)$$

$$B^*_{\|s}\frac{\partial \bar{F}_s}{\partial t} + \frac{\partial}{\partial \varphi}\left(B^*_{\|s}\frac{d\varphi}{dt}\bar{F}_s\right) = 0 \quad : (\tilde{\varphi})$$

$$B^*_{\|s}\frac{\partial \bar{F}_s}{\partial t} + \frac{\partial}{\partial v_{G\|}}\left(B^*_{\|s}\frac{dv_{G\|}}{dt}\bar{F}_s\right) = 0 \quad : (\tilde{v_{G\|}})$$

➠ Semi-Lagrangian scheme

- And the collision operator $(\tilde{\mathcal{C}})$ on a Δt : $\partial_t \bar{F}_s = \mathcal{C}(\bar{F}_s)$ ➠ Crank-Nicolson
- And the source operator $(\tilde{\mathcal{S}})$ on a Δt : $\partial_t \bar{F}_s = \mathcal{S}$ ➠ Crank-Nicolson
- Then, a Boltzmann solving sequence $(\tilde{\mathcal{B}})$ is performed:

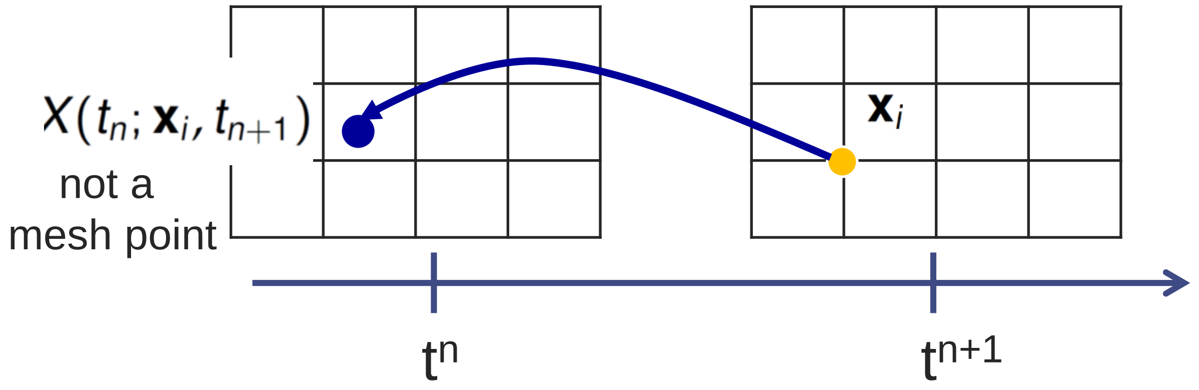
$$(\tilde{\mathcal{B}}) \equiv \left(\frac{\tilde{\mathcal{S}}}{2},\frac{\tilde{\mathcal{C}}}{2}\right)\left(\frac{\tilde{v_{G\|}}}{2},\frac{\tilde{\varphi}}{2},\tilde{\mathcal{X}}_G,\frac{\tilde{\varphi}}{2},\frac{\tilde{v_{G\|}}}{2}\right)\left(\frac{\tilde{\mathcal{C}}}{2},\frac{\tilde{\mathcal{S}}}{2}\right)$$

Example of Backward Semi-Lagrangian (BSL) approach for 2D advection operators

We consider the advection equation: $B^*_{\|s}\frac{\partial \bar{F}_s}{\partial t} + \nabla\cdot\left(B^*_{\|s}\frac{\mathrm{d}\mathcal{X}_G}{\mathrm{d}t}\bar{F}_s\right) = 0$ (with $\mathcal{X}_G = (r,\theta)$)

The Backward Semi-Lagrangian scheme: *(mix between PIC and Eulerian approach)*

- Fixed grid on phase-space *(Eulerian character)*
- Method of characteristics : ODE $\longrightarrow$ origin of characteristics *(PIC character)*



- f is conserved along the characteristics, i.e $f^{n+1}(\mathbf{x}_i) = f^n(X(t_n;\mathbf{x}_i,t_{n+1}))$
- Interpolate on the origin using known values of previous step at mesh points (initial distribution f^0 known).
  - Cubic spline interpolation: good compromise between accuracy and complexity.

GYSELA – MPI parallelization

→ MPI communicator in species + MPI communicator in μ + domain decomposition in 2D

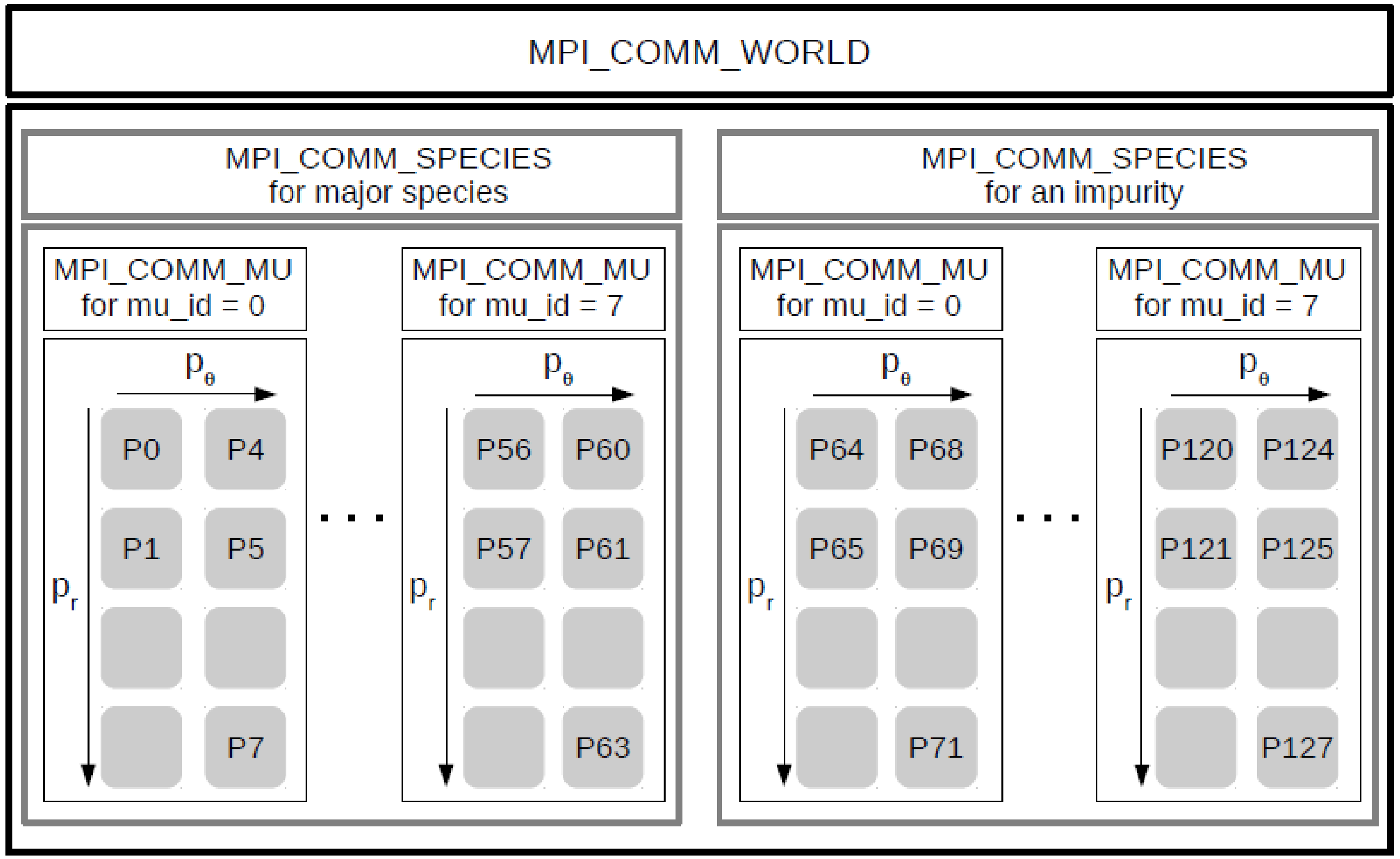


Figure 3.1: MPI_COMM_WORLD communicator decomposition for two species, 8 values of μ, $p_r = 4$ radial sub-domains and $p_\theta = 2$ sub-domains in the poloidal direction. In this case, the number of MPI processes is equal to 128.

GYSELA – MPI parallelization

→ global operators that need huge data transposition = Huge MPI communications

- Several huge **data transposition** to keep all information needed to apply global operator
  - BSL = advection eq.+ Splines interpolation

$$F_s(r = block, \theta = block, \varphi =*, v_{G||} =*, \mu = \mu_{id}) \underset{T_{vlasov}^{-1}}{\overset{T_{vlasov}}{\rightleftarrows}} F_s(r =*, \theta =*, \varphi = block, v_{G||} = block, \mu = \mu_{id})$$

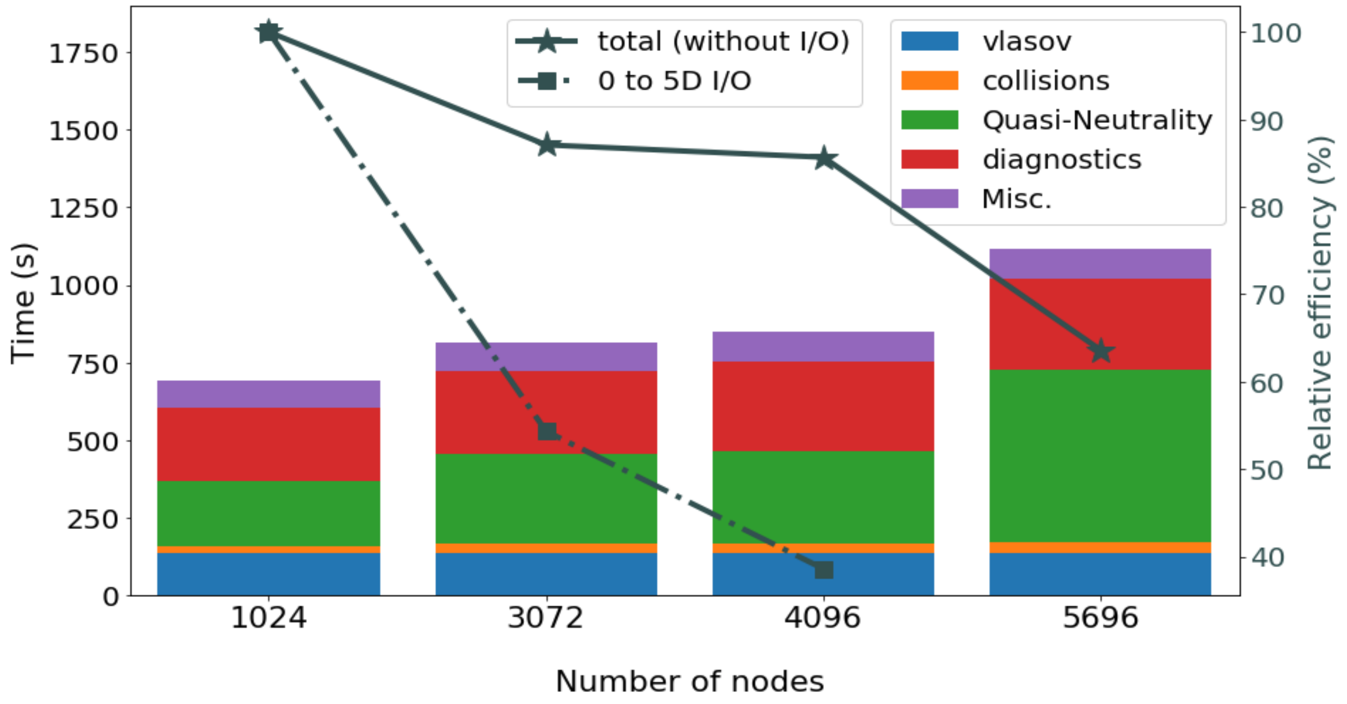
  - Multi-species collision operator

$$\begin{matrix} F_s(r = block, \theta = block, \varphi =*, v_{G||} =*, \mu = \mu_{id}) \\ F_{s1}(r = block, \theta = block, \varphi =*, v_{G||} =*, \mu = \mu_{id}) \\ F_{s2}(r = block, \theta = block, \varphi =*, v_{G||} =*, \mu = \mu_{id}) \end{matrix} \underset{T_{collisions}^{-1}}{\overset{T_{collisions}}{\rightleftarrows}} F(r = block, \theta = block, \varphi = block, v_{G||} =*, \mu =*, s =*)$$

$$\text{Boltzmann solving sequence } (\tilde{\mathcal{B}}) \equiv \left(\frac{\tilde{S}}{2}, \frac{\tilde{C}}{2}\right)\underset{\uparrow \atop T_{collisions}^{-1}}{}\left(\frac{\tilde{v_{G||}}}{2}, \frac{\tilde{\varphi}}{2}\underset{\uparrow \atop T_{vlasov}}{,} \tilde{X}_G, \underset{\uparrow \atop T_{vlasov}^{-1}}{}\frac{\tilde{\varphi}}{2}, \frac{\tilde{v_{G||}}}{2}\right)\underset{\uparrow \atop T_{collisions}}{}\left(\frac{\tilde{C}}{2}, \frac{\tilde{S}}{2}\right)$$

GYSELA optimized up to 700k CPU cores

- Weak scaling up to 5696 nodes (729 088 cores) on CEA-HF : BullSequana XH2000, AMD EPYC 7763 64C 2.45GHz, Atos BXI V2, 810 240 cores (= 6330 nodes)



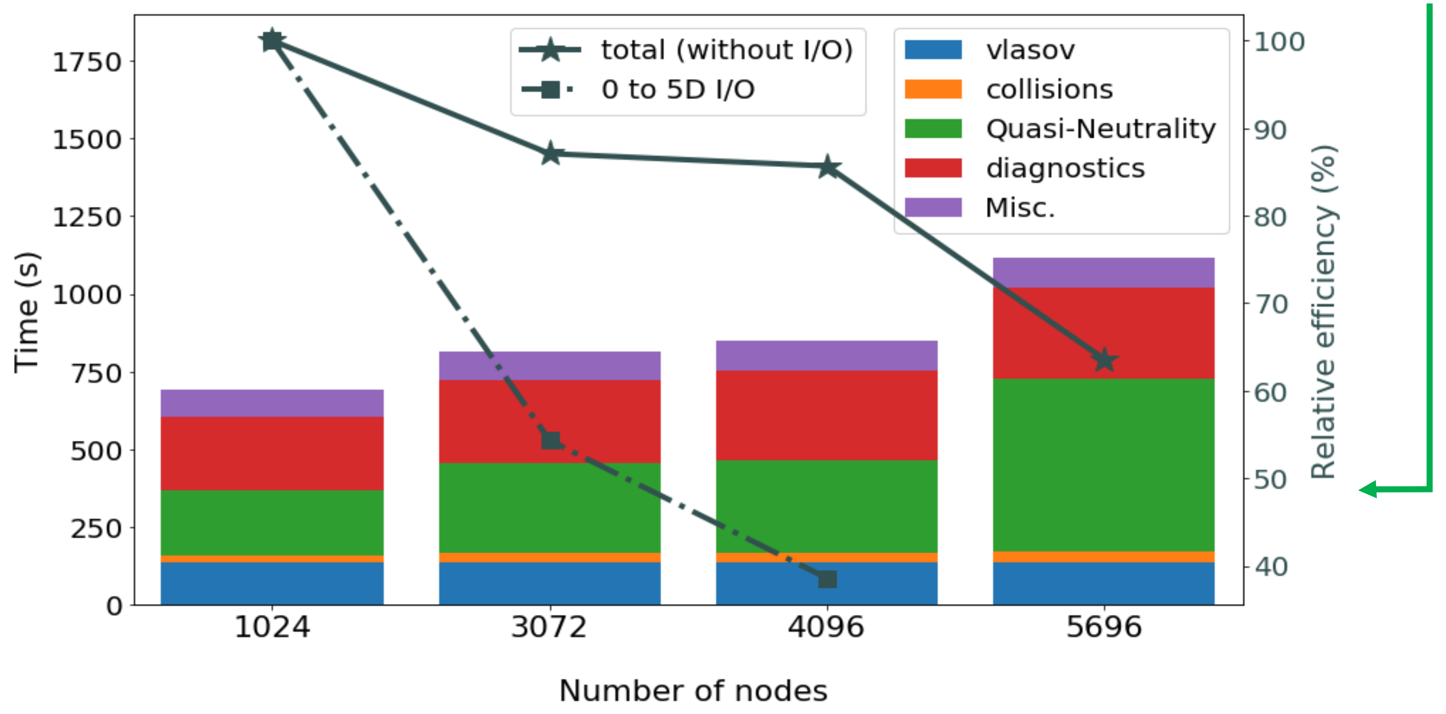
Relative efficiency of 85% on more than 500k cores and 63% on 729 088 cores

Two main bottlenecks for exascale simulations
→ 1. QN solver due to huge MPI reduction

3D Poisson solver (2D+1D) **:** 2D (r,θ) spline finite elements for each φ

$$\frac{e}{T_{e,eq}}\left(\phi - \langle\phi\rangle\right) - \frac{1}{n_{e_0}}\sum_s Z_s \nabla_\perp \cdot \left(\frac{n_{s,eq}}{B\Omega_s}\nabla_\perp \phi\right) = \frac{1}{n_{e_0}}\sum_s Z_s \int J_0 \cdot \left(\bar{F}_s - \bar{F}_{s,eq}\right) d^3v$$

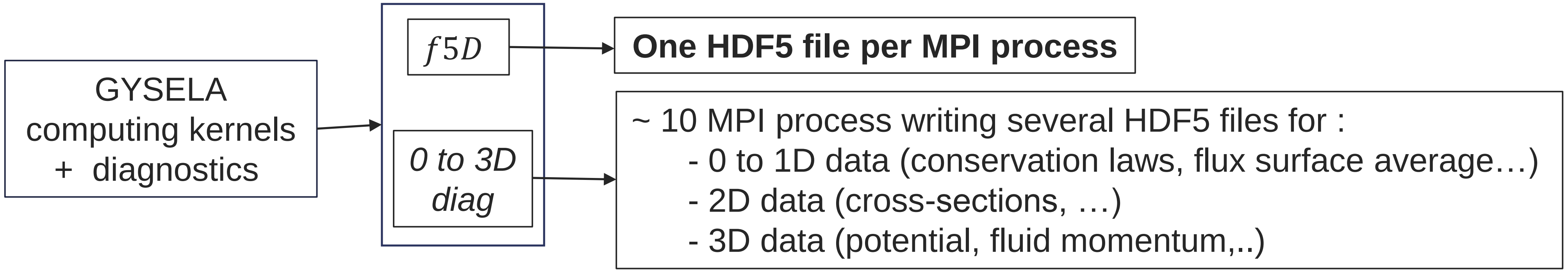
- Most of the computation time is spent for computing the rhs: $\frac{1}{n_{e_0}}\sum_s Z_s \int J_0 \cdot \left(\bar{F}_s - \bar{F}_{s,eq}\right) d^3v$



Big MPI collective communication (reduction) for integral computation

- **Future work** (within EoCoE-III project) : **Full 3D Poisson solver** to tackle stelarator geometries

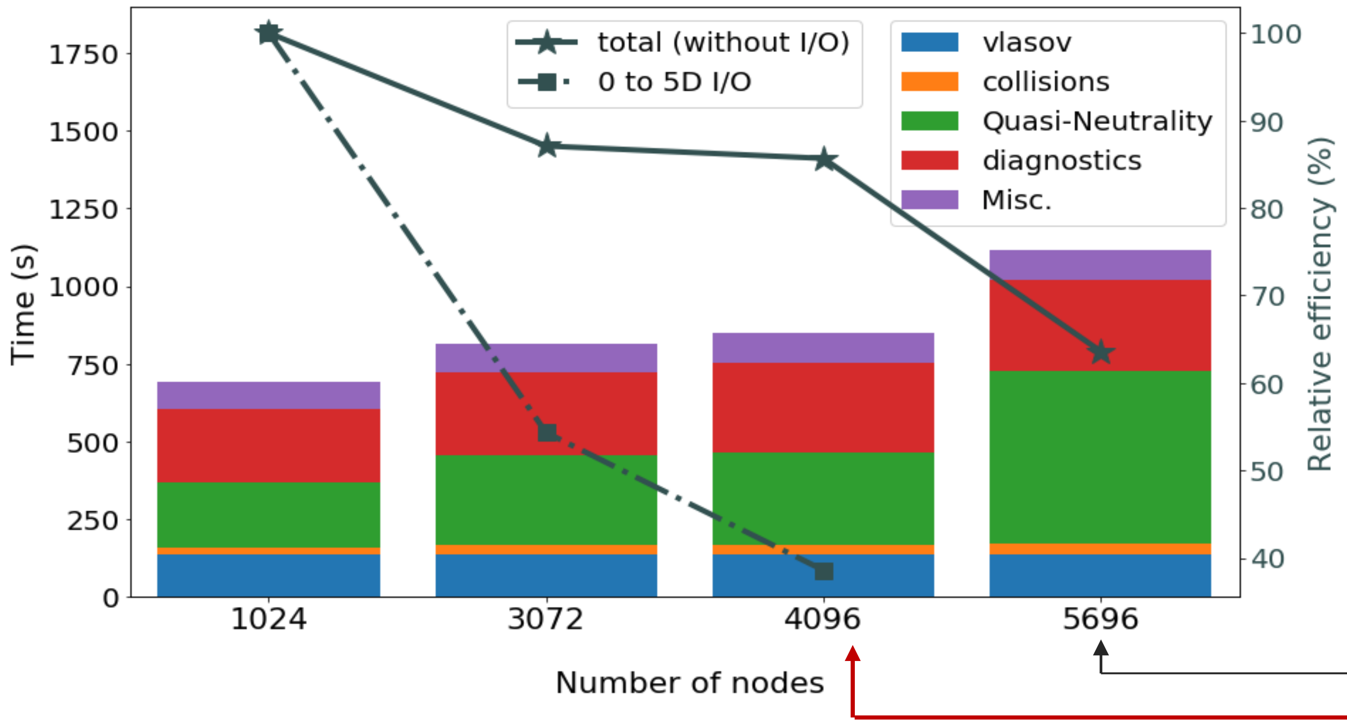
Two main bottlenecks for exascale simulations
→ 2. Diagnostics + restart file writing



One HDF5 file per MPI process

~ 10 MPI process writing several HDF5 files for :
- 0 to 1D data (conservation laws, flux surface average…)
- 2D data (cross-sections, …)
- 3D data (potential, fluid momentum,..)

- I/O scalability: ~50% for 3072 nodes and ~38% for 4096 nodes. Crash on 5696 nodes



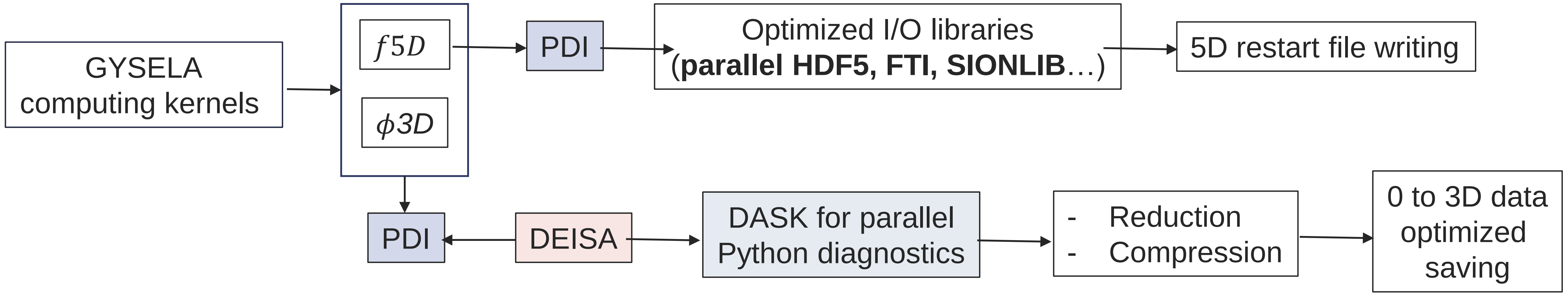
Huge amount of data successfully written with 22784 MPI process
→ 16.2 Tbytes
→ 22784 HDF5 files written at the same time

45568 MPI process
→ Not succeed in writting the 45568 files

Work in progress : Optimized Data workflow

- Main idea : **Decouple I/O from computing kernels**

- Development of in-situ diagnostics framework based on PDI + DEISA + DASK
  - PDI Data Interface for handling I/O (developed at MDLS) https://pdi.julien-bigot.fr/master/
  - DEISA (dask-enabled in situ analytics) library (developed at MDLS+INRIA) *[A. Gueroudji et al., HiPC 2021]*
  - DASK a flexible library for parallel computing in Python https://docs.dask.org/

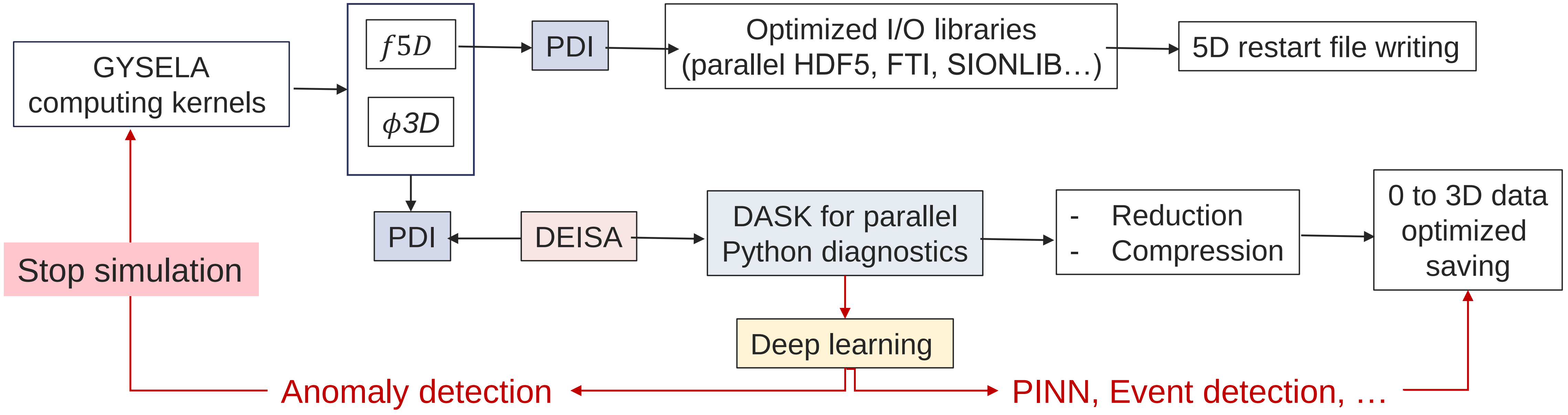

GYSELA computing kernels
$f5D$
$\phi3D$
PDI
Optimized I/O libraries (**parallel HDF5, FTI, SIONLIB**…)
5D restart file writing
PDI
DEISA
DASK for parallel Python diagnostics
- Reduction
- Compression
0 to 3D data optimized saving

Within ExaDost scope

Work in progress : In-situ AI diagnostics

- Main idea : **Decouple I/O from computing kernels**

- Development of **in-situ AI diagnostics to optimize exascale simulations**:
  - Automatic anomaly detection → Automatic stop of simulation → CPU or GPU consumption optimization
  - Automatic rare event detection → Optimisation of diagnostic saving → Memory storage reduction



Within ExaDost scope

Roadmap for GyselaX++ towards exascale
→ Why do we choose to rewrite GYSELA ?

- Unique code for both CPU (AMD milan or ARM-A64FX) and GPU with OpenMP directives is NOT optimal → extremely difficult to optimize on all architectures.

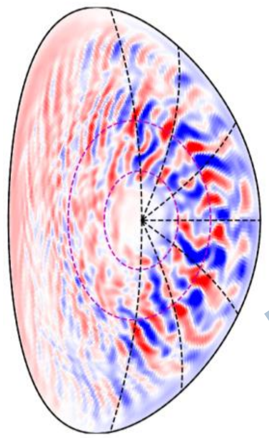
- Non-equidistant mesh mandatory for core-edge-SOL turbulence simulations

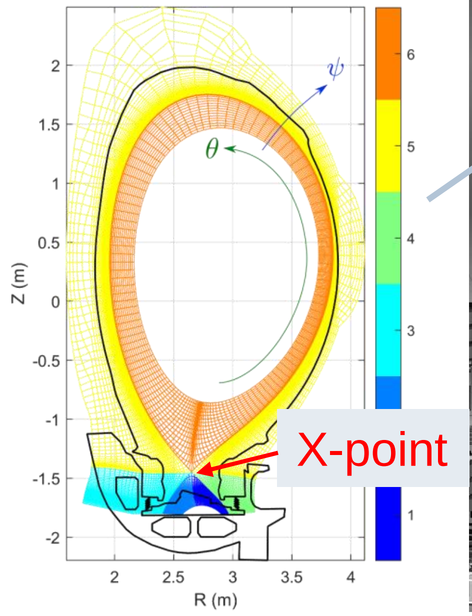
  → Modifying splines in GYSELA = rewrite most of the kernels

- X-point geometry

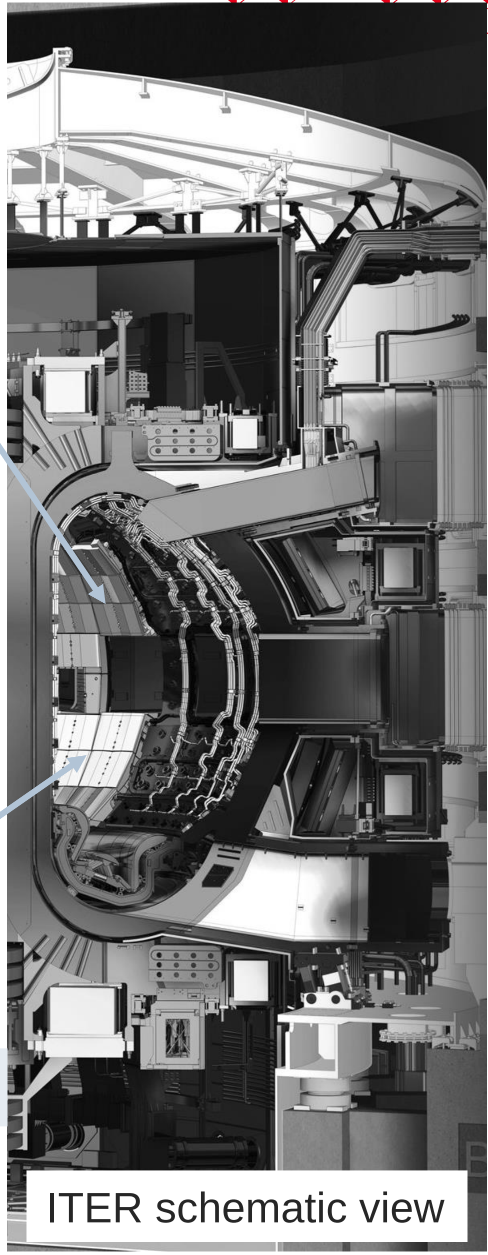
  → Development of new semi-Lagrangian scheme required to treat multipatches

Simpler to rewrite main kernels in modern C++ from scratch
→ GyselaX++ code


GYSELA
D-Shape geometry



SOLEDGE-3X
X-point geometry


ITER schematic view

Gysela-X towards exascale
→ Complete rewriting of the code in modern C++

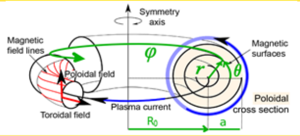
- Main idea: Mutualize all modules independent on the 3D space geometry between Fortran code and C++ code

Extract F90 modules → rewrite them in C++/ GPU → plug them to F90 old code + C++ new code

GYSELA Fortran Code

High level of parallelism MPI

Geometry | n, T, srce profiles, …



Vlasov | **5D semi-lagrangian** scheme with equidistant splines

+ Collisions | Sources

Poisson | **3D solver** based on 2D spline FEM + 1D

I/O | **IMAS** via PDI

+ **In-Situ diagnostics**

GITLAB + Jenkins non-regression tests

Future GyselaX++ Code structure

Input | YAML + Paraconf (developed at MDLS)

Geometry | DDC a domain decomposition library | C++ template meta-programming domain decomposition library (developed at MDLS)

Vlasov | **5D semi-lagrangian** scheme with non-equidistant splines

+ Collisions | Sources

Poisson | **3D solver** based on 2D spline FEM + 1D

I/O | **IMAS** via PDI

+ **In-Situ diagnostics**

GITLAB + google tests + GITLAB-CI + SPACK installation

+ High level of parallelism MPI + KOKKOS

(rewriting within EoCoE-III project (2024-2027) + moonshot CExA (2023-2025))

Conclusions

- The GYSELA code at the era of pre-exascale for ion-scale turbulence simulations for current tokamaks
  - Optimized up to more than 500k cores on standard CPU architecture (ex: AMD milan)
  - Resource needs: more than 150 millions of CPU hours / year
  - Petabytes of data manipulated per simulation with huge reduction to limit the storage to few Terabytes

  → Lot of physics still to be explored with this version of the code for the next five years.

- **GyselaX++ : Rewritting in modern C++**, more modular and scalable on different accelerated architectures
  - More realistic temperature gradients at the edge: Non-equidistant mesh
  - More realistic geometry: X-point
  - **Based on DDC library + Kokkos**