GyselaX++: Exascale Challenges for tokamak plasma turbulence simulations

Virginie Grandgirard

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

**Mesh**
(equidistant in $(r, \theta, \varphi)$)

**Magnetic configuration**
(simplified circular concentric)

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

$$B^* \frac{\partial \bar{F}_s}{\partial t} + \nabla \cdot \left( \frac{d\mathbf{x}_G}{dt} B^*_s \bar{F}_s \right) + \frac{\partial}{\partial v_{G||}} \left( \frac{d\mathbf{v}_{G||}}{dt} B^*_{||s} \bar{F}_s \right) = C(\bar{F}_s) + S + K_{\text{buff}}(\bar{F}_s) + D_{\text{buff}}(\bar{F}_s)$$

with the equations of motion:

$$B^*_{||s} \psi_{G||} = v_{G||} B^* + \frac{1}{\epsilon} b \times \nabla \Lambda ; \quad B^*_{||s} m_s d_s v_{G||} = -\mathbf{B} \cdot \nabla \Lambda$$

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

**3D Poisson solver**: 2D $(r, \theta)$ spline finite elements for each $\varphi$

$$\frac{e}{T_{\epsilon, eq}} \left( \phi - \langle \phi \rangle \right) - \frac{1}{n_{\theta_0}} \sum_s Z_s V_{\perp} \left( \frac{n_{s, eq}}{B \Omega_s} V_{\perp} \phi \right) = \frac{1}{n_{\theta_0}} \sum_s Z_s \int J_0 \cdot (\bar{F}_s - \bar{F}_{s, eq}) d^3\mathbf{v}$$

**Ampere**
3D equation for the potential vector $\mathbf{A}$ ([Finite Differences in $r$ + Fourier in $\theta$] for each $\varphi$)
Time-splitting for Boltzmann equation

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

\[
B^*_s \frac{\partial \tilde{F}_s}{\partial t} + \nabla \cdot \left( \frac{d x_G}{d t} B^*_s \tilde{F}_s \right) + \frac{\partial}{\partial v_G} \left( \frac{d v_G}{d t} B^*_s \tilde{F}_s \right) = C(\tilde{F}_s) + S
\]

- Let us define three advection operators (with \(X_G = (r, \theta)\))

\[
B^*_s \frac{\partial \tilde{F}_s}{\partial t} + \nabla \cdot \left( B^*_s \frac{d X_G}{d t} \tilde{F}_s \right) = 0 : (\tilde{X}_G)
\]

\[
B^*_s \frac{\partial \tilde{F}_s}{\partial t} + \frac{\partial}{\partial \phi} \left( B^*_s \frac{d \phi}{d t} \tilde{F}_s \right) = 0 : (\tilde{\phi})
\]

\[
B^*_s \frac{\partial \tilde{F}_s}{\partial t} + \frac{\partial}{\partial v_G} \left( B^*_s \frac{d v_G}{d t} \tilde{F}_s \right) = 0 : (\tilde{v}_G)
\]

- And the collision operator (\(\tilde{C}\)) on a \(\Delta t\) : \(\partial_t \tilde{F}_s = C(\tilde{F}_s)\)

- And the source operator (\(\tilde{S}\)) on a \(\Delta t\) : \(\partial_t \tilde{F}_s = S\)

- Then, a Boltzmann solving sequence (\(\tilde{B}\)) is performed:

\[
(\tilde{B}) = \left( \frac{\tilde{S}}{2}, \frac{\tilde{C}}{2} \right) \left( \frac{\tilde{v}_G}{2}, \frac{\tilde{\phi}}{2}, \frac{\tilde{X}_G}{2}, \frac{\tilde{\phi}}{2}, \frac{\tilde{v}_G}{2} \right) \left( \frac{\tilde{C}}{2}, \frac{\tilde{S}}{2} \right)
\]
Example of Backward Semi-Lagrangian (BSL) approach for 2D advection operators

We consider the advection equation:

\[ \mathbf{B}_\parallel \frac{\partial \mathbf{F}_s}{\partial t} + \nabla \cdot \left( \mathbf{B}_\parallel \frac{d\mathbf{X}_G}{dt} \mathbf{F}_s \right) = 0 \quad \text{(with } \mathbf{X}_G = (r, \theta)) \]

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

- Fixed grid on phase-space \((\text{Eulerian character})\)
- Method of characteristics : ODE \(\rightarrow\) origin of characteristics \((\text{PIC character})\)

\[ \mathbf{X}(\mathbf{x}_i; t_n, t_{n+1}) \]

\[ \text{not a mesh point} \]

- \(f\) is conserved along the characteristics, i.e. \(f^{n+1}(\mathbf{x}_i) = f^n(\mathbf{X}(\mathbf{x}_i; t_n, t_{n+1}))\)
- 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 $\mu$ + domain decomposition in 2D

Figure 3.1: MPI_COMM_WORLD communicator decomposition for two species, 8 values of $\mu$, $p_r = 4$ 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

\[ F_s(r = \text{block}, \theta = \text{block}, \varphi = *, v_{G||} = *, \mu = \mu_{id}) \]

- Multi-species collision operator

\[ F_s(r = \text{block}, \theta = \text{block}, \varphi = *, v_{G||} = *, \mu = \mu_{id}) \]

\[ F_{s1}(r = \text{block}, \theta = \text{block}, \varphi = *, v_{G||} = *, \mu = \mu_{id}) \]

\[ F_{s2}(r = \text{block}, \theta = \text{block}, \varphi = *, v_{G||} = *, \mu = \mu_{id}) \]

\[ T_{\text{vlasov}} \rightarrow F_s(r = *, \theta = *, \varphi = \text{block}, v_{G||} = \text{block}, \mu = \mu_{id}) \]

\[ T_{\text{vlasov}}^{-1} \]

\[ T_{\text{collisions}} \rightarrow F(r = \text{block}, \theta = \text{block}, \varphi = \text{block}, v_{G||} = *, \mu = *, s = *) \]

\[ T_{\text{collisions}}^{-1} \]

Boltzmann solving sequence \( (\vec{B}) \equiv \left( \frac{\tilde{S}}{2}, \frac{\tilde{C}}{2} \right) \left( \frac{v_{G||}}{2}, \frac{\tilde{\varphi}}{2}, \tilde{\chi}_G, \frac{\tilde{\varphi}}{2}, \frac{v_{G||}}{2} \right) \left( \frac{\tilde{C}}{2}, \frac{\tilde{S}}{2} \right) \)
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 \((r,\theta)\) spline finite elements for each \(\phi\)

\[
\frac{\epsilon}{T_{e,eq}} \left( \phi - \langle \phi \rangle \right) - \frac{1}{n_{e_0}} \sum_{s} Z_s V_{\perp} \cdot \left( \frac{n_{s,eq}}{B\Omega_s} V_{\perp} \phi \right) = \frac{1}{n_{e_0}} \sum_{s} Z_s \int J_0 \cdot (\vec{F}_s - \vec{F}_{s,eq}) d^3v
\]

- Most of the computation time is spent for computing the rhs:
  \[
  \frac{1}{n_{e_0}} \sum_{s} Z_s \int J_0 \cdot (\vec{F}_s - \vec{F}_{s,eq}) d^3v
  \]

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

Big MPI collective communication (reduction) for integral computation
Two main bottlenecks for exascale simulations
→ 2. Diagnostics + restart file writing

GYSELA computing kernels + diagnostics

\[ f_{5D} \]

\[ 0 \text{ to } 3D \text{ diag} \]

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 writing the 45568 files
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/](https://pdi.julien-bigot.fr/master/)
- DEISA (dask-enabled in situ analytics) library (developed at MDLS+INRIA)  [A. Gueroudji et al., HiPC 2021](https://docs.dask.org/)
- DASK a flexible library for parallel computing in Python  [https://docs.dask.org/](https://docs.dask.org/)

Diagram:
- GYSELA computing kernels
  - $f^{5D}$
  - $\phi^{3D}$
- PDI
- Optimal I/O libraries (parallel HDF5, FTI, SIONLIB…)
- 5D restart file writing
- 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 $\rightarrow$ Automatic stop of simulation $\rightarrow$ CPU or GPU consumption optimization
- Automatic rare event detection $\rightarrow$ Optimisation of diagnostic saving $\rightarrow$ Memory storage reduction

---

**Diagram:**

- GYSELA computing kernels
  - $f^{5D}$
  - $\phi^{3D}$
- Stop simulation
- Anomaly detection
- PINN, Event detection, …
- 0 to 3D data optimized saving
- Reduction - Compression
- DASK for parallel Python diagnostics
- DEISA
- PDI
- Optimized I/O libraries (parallel HDF5, FTI, SIONLIB…)
- 5D restart file writing
- Deep learning

---

Virginie Grandgirard – 2023/11/08
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-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**

- **Vlasov**
  - 5D semi-lagrangian scheme with equidistant splines
  - Collisions, Sources
- **Poisson**
  - 3D solver based on 2D spline FEM + 1D
- **I/O**
  - IMAS via PDI
- **High level of parallelism**
  - MPI
- **GITLAB + Jenkins non-regression tests**

---

**Future GyselaX++ Code structure**

- **Input**
  - YAML + Paraconf (developed at MDLS)
- **Poisson**
  - 3D solver based on 2D spline FEM + 1D
- **I/O**
  - IMAS via PDI
- **High level of parallelism**
  - MPI + KOKKOS
- **GITLAB + google tests + GITLAB-CI + SPACK installation**

---

(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++ : Rewriting 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