Seminar

Compile-time Symbolic Solver for GMEC and FP3D

P.Y. Jiang[1], Z.Y. Liu[1], S.Y. Liu[1], J. Bao[2], G.Y. Fu[1]

[1]Institute for Fusion Theory and Simulation and School of Physics,

Zhejiang University, Hangzhou 310027, China

[2]Institute of Physics, Chinese Academy of Sciences, Beijing 100190, China

浙江大学聚变理论与模拟中心

Institute for Fusion Theory and Simulation, Zhejiang University

Princeton Plasma Physics Laboratory. Dec 5. 2023

Outline

1. Background
2. Compile-time Symbolic Solver CSS
3. Optimization for fluid and particle simulations
4. Shifted metric method
5. Gyrokinetic MHD hybrid code GMEC
6. Field and particle code FP3D
7. Conclusion

Outline

1 Background

2 Compile-time Symbolic Solver CSS

3 Optimization for fluid and particle simulations

4 Shifted metric method

5 Gyrokinetic MHD hybrid code GMEC

6 Field and particle code FP3D

7 Conclusion

Background

Hybrid codes and gyrokinetic codes

Program name	Physics model	Type	Numerical method
MEGA	Kinetic-MHD hybrid	Initial value	Explicit, finite difference, PIC
M3D-K	Kinetic-MHD hybrid	Initial value	Semi-Implicit, finite element, PIC
M3D-C1-K	Kinetic-MHD hybrid	Initial value	Implicit, finite element, PIC
GEM	Gyrokinetic	Initial value	PIC
GTC	Gyrokinetic	Initial value	PIC
GYRO	Gyrokinetic	Initial value	Vlasov
GENE	Gyrokinetic	Initial value	Vlasov
BOUT++	Extended MHD	Initial value	Finite difference, fluid
GMEC	Gyrokinetic-MHD hybrid	Initial value	Explicit, finite difference, PIC

More physics, high orders and high efficiency
Field-align coordinates

Gyrokinetic-MHD hybrid model (GMEC)

Extended MHD equations of GMEC

Vorticity equation

$$\frac{d}{dt}\left(\frac{n_i e^2}{T_i}(1-\Gamma_0)\Phi\right) + \delta\vec{B}\cdot\nabla\left(\frac{\mu_0 J_\parallel}{B}\right) + \left(\vec{B}+\delta\vec{B}\right)\cdot\nabla\left(\frac{\mu_0\delta J_\parallel}{B}\right) + \frac{\mu_0\vec{B}\times\nabla B}{B^3}\cdot\nabla_\perp(\delta P + \delta P_h)$$

Parallel Ohm's law

$$\frac{\partial}{\partial t}\delta A_\parallel = -\partial_\parallel\delta\phi - \eta_\parallel\delta J_\parallel + \frac{1}{en_e}\partial_\parallel P_e$$

Electron pressure equation

$$\frac{d}{dt}P_e = -\gamma\nabla\cdot\vec{v}_e P_e$$

where

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \left(\frac{\vec{b}\times\nabla\Phi}{B} + \vec{v}_{*i}\right)\cdot\nabla \qquad \Gamma_0 = e^{-k_\perp^2\rho_i^2} I_0\left(k_\perp^2\rho_i^2\right) \qquad \partial_\parallel = \vec{b}\cdot\nabla \qquad \delta\vec{B} = \nabla\times\left(\delta A_\parallel\vec{b}\right)$$

$$\bar{\omega} = \nabla\cdot\frac{1}{v_A^2}\nabla_\perp\Phi \qquad \delta J_\parallel = -\nabla_\perp^2\delta A_\parallel \qquad \vec{v}_e = \frac{\vec{E}\times\vec{B}}{B^2} - \frac{\delta J_\parallel}{en_e}\vec{b} \qquad \vec{v}_{*i} = \frac{1}{enB}\vec{b}\times\nabla P_i$$

Full FLR effect, diamagnetic effect, parallel gradient of electron pressure

Gyrokinetic-MHD hybrid model (GMEC)

PIC method of GMEC

$$f = f_0 + \delta f$$
$$w \equiv \frac{\delta f}{g}$$
$$\frac{dw_i}{dt} = -\left[\frac{f_i(t=0)}{g_i(t=0)} - w_i\right]\frac{1}{f_0}\left(\frac{dP_\phi}{dt}\frac{\partial f_0}{\partial P_\phi} + \frac{dE}{dt}\frac{\partial f_0}{\partial E}\right)$$

Where

$$\frac{dE}{dt} = \frac{d\boldsymbol{X}}{dt}\cdot\mu\nabla B + \frac{dv_\parallel}{dt}mv_\parallel$$

$$\frac{dP_\phi}{dt} = \frac{d\boldsymbol{X}}{dt}\cdot\nabla P_\phi + \frac{dv_\parallel}{dt}\frac{\partial P_\phi}{\partial v_\parallel}$$

$$E = \frac{1}{2}mv_\parallel^2, \qquad P_\phi = qg\rho_\parallel - q\psi_p$$

Gyro-center $\boldsymbol{X}, v_\parallel$

Gyro-kinetic equations

$$\frac{d\boldsymbol{X}}{dt} = \frac{1}{B^{**}}\left\{v_\parallel\boldsymbol{B}^* - \boldsymbol{b}\times\left[\langle\boldsymbol{E}\rangle - \frac{\mu}{e}\nabla(B + \langle\delta B\rangle)\right]\right\}$$
$$\frac{dv_\parallel}{dt} = \frac{e}{mB^{**}}\boldsymbol{B}^*\cdot\left[\langle\boldsymbol{E}\rangle - \frac{\mu}{e}\nabla(B + \langle\delta B\rangle)\right]$$

where

$$\boldsymbol{B}^* = \boldsymbol{B} + \langle\delta\boldsymbol{B}\rangle + \frac{mv_\parallel}{e}\nabla\times\boldsymbol{b}, \qquad B^{**} = \boldsymbol{B}^*\cdot\boldsymbol{b}$$

$$\delta\boldsymbol{B} = \nabla\times(\delta A_\parallel\boldsymbol{b}), \qquad \boldsymbol{E} = -\nabla\delta\phi - \frac{\partial\delta A_\parallel}{\partial t}\boldsymbol{b}$$

Pressure coupling

$$\delta P_\parallel = \iiint m\,v_\parallel^2\delta f d^3v = \frac{1}{N_p}\sum_i^{N_p} mv_{\parallel,i}^2 w_i\frac{1}{J}\delta(x-x_i)\delta(y-y_i)\delta(z-z_i)$$

$$\delta P_\perp = \iiint \frac{1}{2}mv_\perp^2\delta f d^3v = \frac{1}{N_p}\sum_i^{N_p}\frac{1}{2}mv_{\perp,i}^2 w_i\frac{1}{J}\delta(x-x_i)\delta(y-y_i)\delta(z-z_i)$$

Field-aligned coordinates

Field-aligned coordinate

- Many instabilities: strong flute mode character

$$\nabla_{\parallel}^2 \ll \nabla_{\perp}^2$$

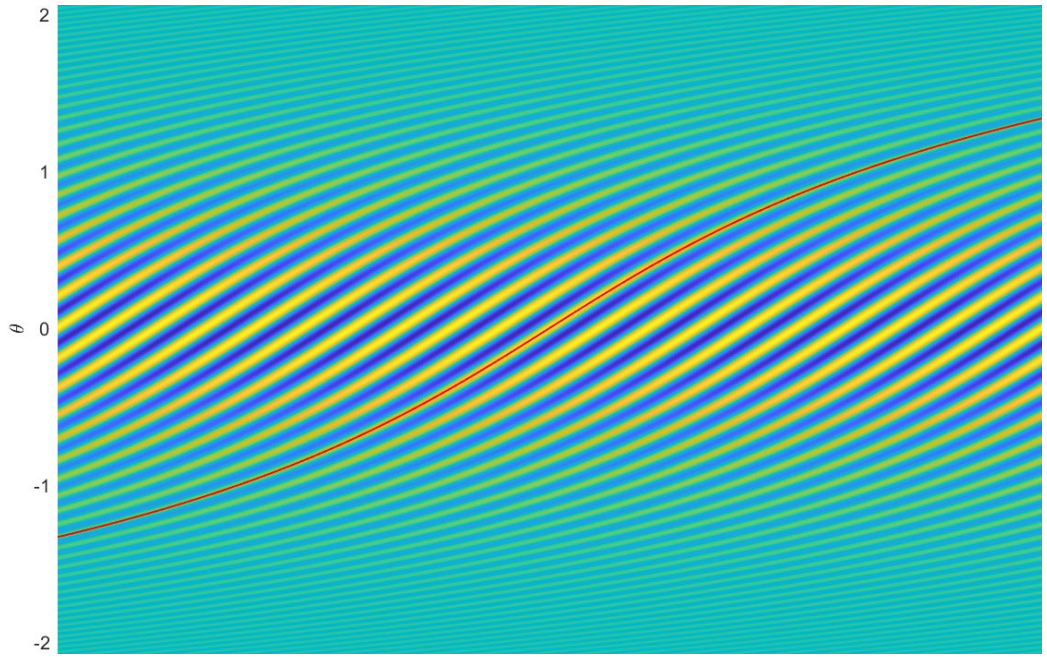
- Field-aligned coordinates:
  relatively fewer grids in parallel coordinates

Shifted metric method

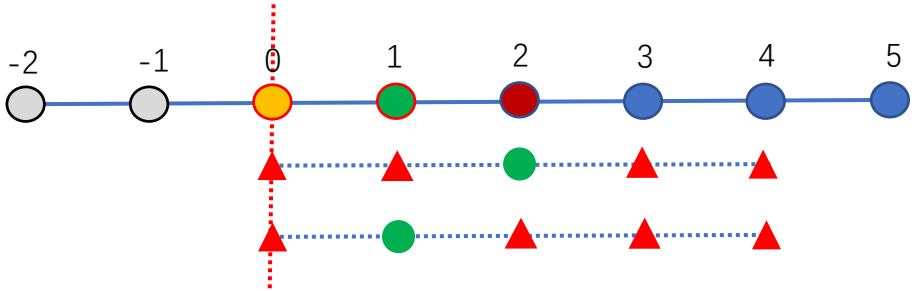
- The discontinuity of radial derivative in field-aligned boundary causes numerical instability.
- Shifted metric method: avoid boundary numerical instability.
- No numerical diffusion term, smoothing, and filtering needed in GMEC.

High order numerical difference

- More precise calculation in numerical difference, low truncation error.
- Boundary needs biased difference scheme.



Mode structure of IBM in θ, ϕ coordinate



Biased difference scheme in boundary

High efficiency

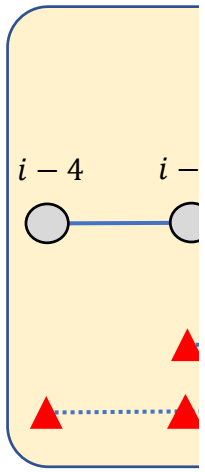
Hybrid MPI and TBB parallel scheme

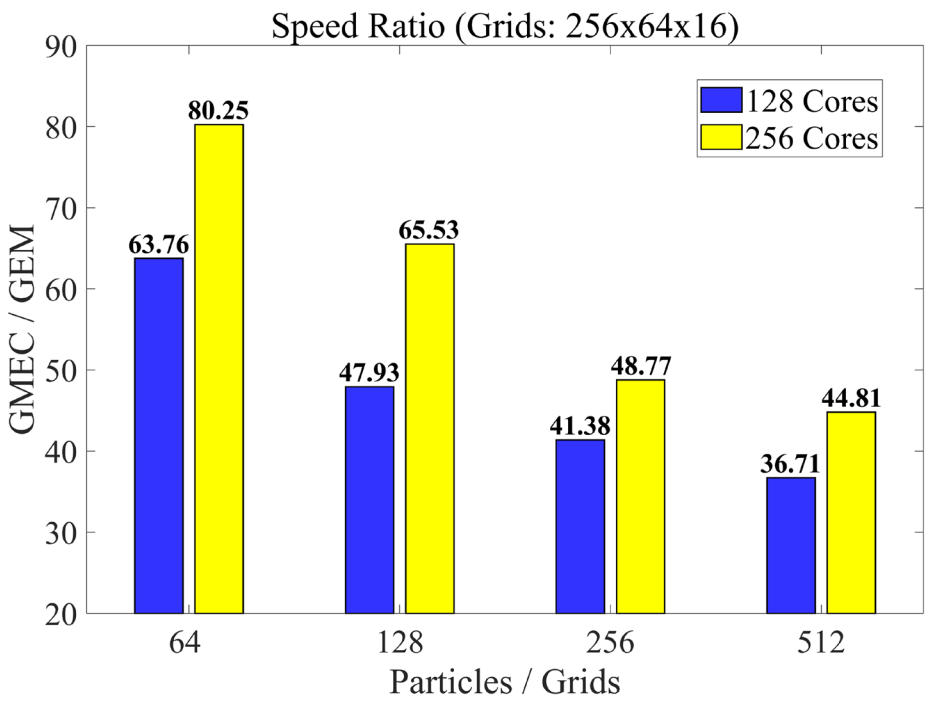
- MPI: distributed memory, needed by clusters.
- TBB: sheared memory, more efficient in one node.

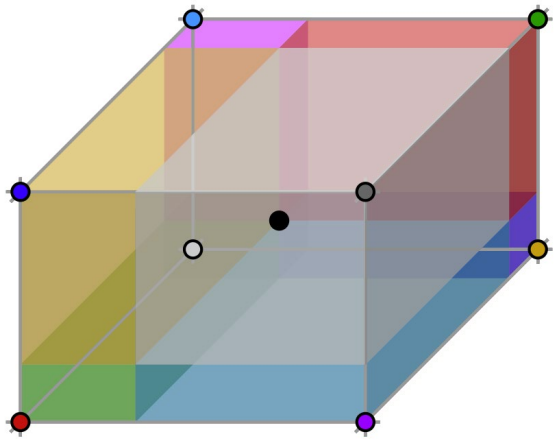
Instruction optimization

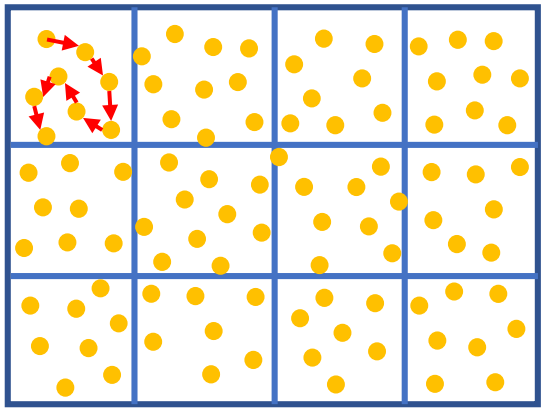
- For MHD:
- Computational complexity optimization:
  merging coefficients, minimize the number of multiplications.
- Memory access optimization:
  transform random access into sequential access. Increasing cache hitting ratio.

- For particle:
- Redundant data structure:
  Ensure grid continuity for one particle. Increase cache hitting ratio.
- Parallel counting sort:
  Travel particle by cell id. Decrease memory access.









Easier to implement

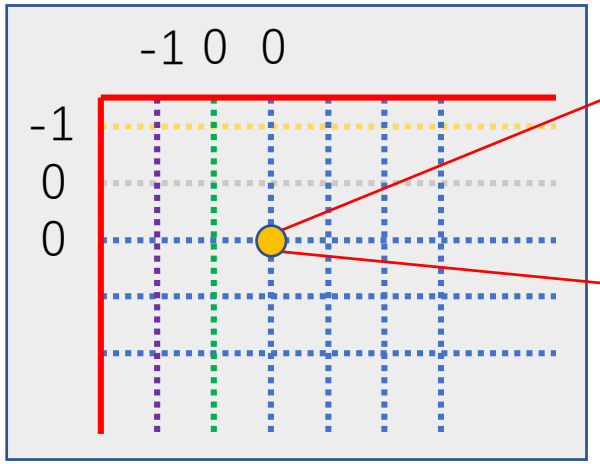
- Curvilinear coordinates: both contravariant and covariant vector/tensor forms, metric tensor.

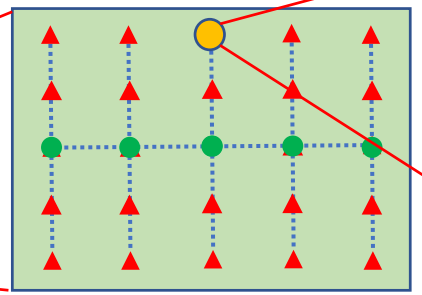
$$\nabla_\perp^2 \Phi = \frac{1}{J}\partial_i\left(J g^{ij}\partial_j \Phi\right), \text{ 18 terms}$$

$$\delta J_\parallel = -\frac{1}{B}\nabla\cdot\left(B^2\nabla_\perp\left(\frac{\delta A_\parallel}{B}\right)\right) = -\frac{1}{JB}\partial_i\left(JB^2 g^{ij}\partial_j\left(\frac{\delta A_\parallel}{B}\right)\right), \text{ 48 terms}$$

- Implicit equation: matrix loading
- Boundary needs biased difference scheme.

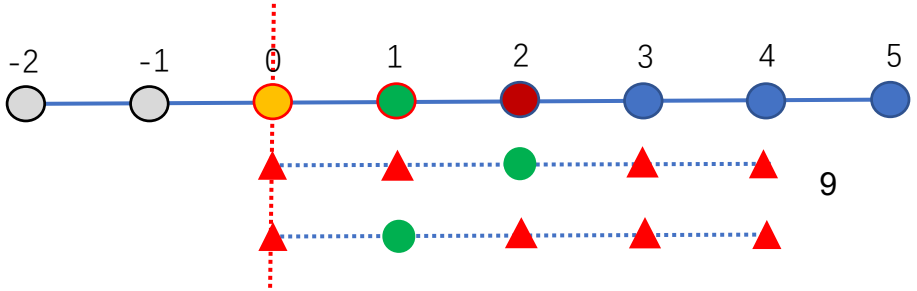
  totally different coefficients





5 points

$$C_{2,0,0} = -\frac{1}{12}\left[\frac{1}{J}\left(\frac{\partial J}{\partial x}g_{xx} + \frac{\partial J}{\partial y}g_{yx} + \frac{\partial J}{\partial z}g_{zx}\right) + \left(\frac{\partial g_{xx}}{\partial x} + \frac{\partial g_{yx}}{\partial y} + \frac{\partial g_{zx}}{\partial z}\right) + (g_{xx})\right]$$



Compile-time Symbolic Solver

General-purpose framework. Solving PDE and ODE in finite difference method.

C++20 metaprogramming code. All the operations completed at compile time.

Functions:

- Automatically expand arbitrary vector/tensor equations into scalar forms.
- Arbitrary coordinates system with arbitrary boundary conditions.
- Arbitrary order of finite difference.
- Load matrix automatically.
- Automatic instruction optimization.

$$\delta\bar{\omega} = \nabla \cdot \frac{1}{v_A^2} \nabla_\perp \delta\Phi$$

```
auto dw = Div*(_va2*Nabla*dPhi)
```

Feature:

- Implement new models quickly.
- Greatly reduce the risks of implementation errors.
- Compile-time abstraction, no cost at run time.
- Instruction optimization, much faster than direct codes.

CSS: comparison with other methods

Method	Arbitrary equations	Arbitrary coordinates system	Implement work	Implement mistake probability	Runtime efficiency
Manual	No	No	huge	high	usual
Code with Mathematica help	No	No	usual	usual	usual
BOUT++	Partial	No	little	low	low
Runtime symbolic	Yes	Yes	little	low	very low
CSS	Yes	Yes	little	low	high

Mathematica:

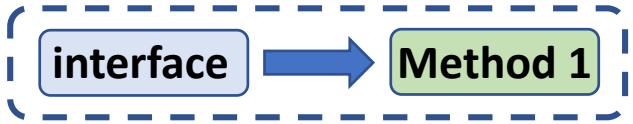
- expand equations, transform into C form.
- only symbolic expression, no operation of memory.

BOUT++:

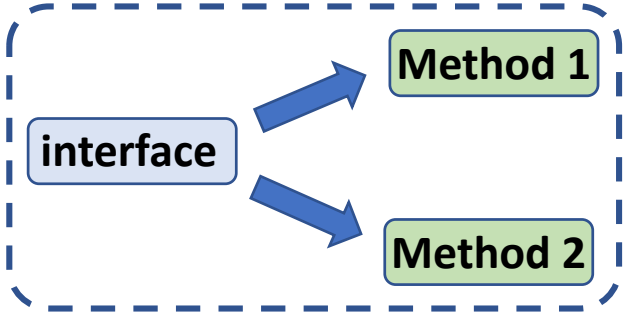
- Only given vector operations, given coordinate system.
- Run-time abstraction, slower than usual codes.

Compile-time abstraction

Abstraction cost



Direct call



Abstract call

Direct call:

- No cast at runtime
- More information for compiler optimization. (inline)

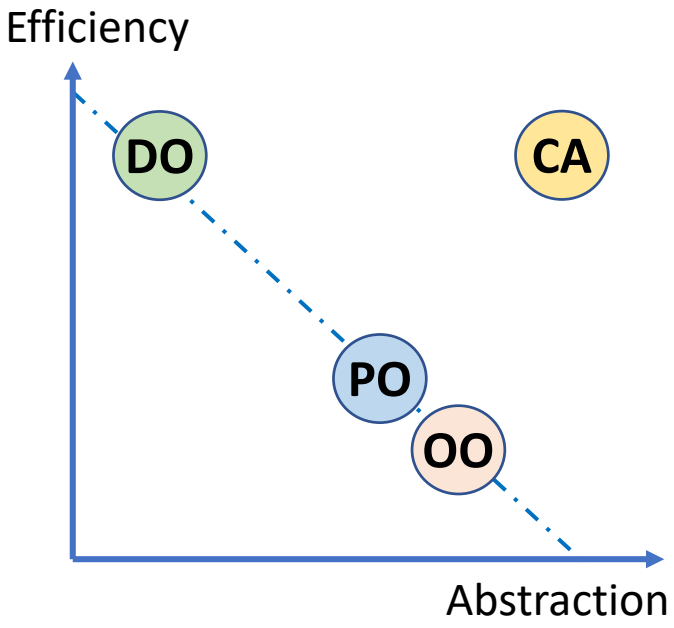
Runtime abstract call:

- Need time to judge which function to call.
- Hinder compiler optimization.

Compile-time abstract:

- Only generate the needed instruction.
- Generate data-oriented code automatically.

Programing paradigm

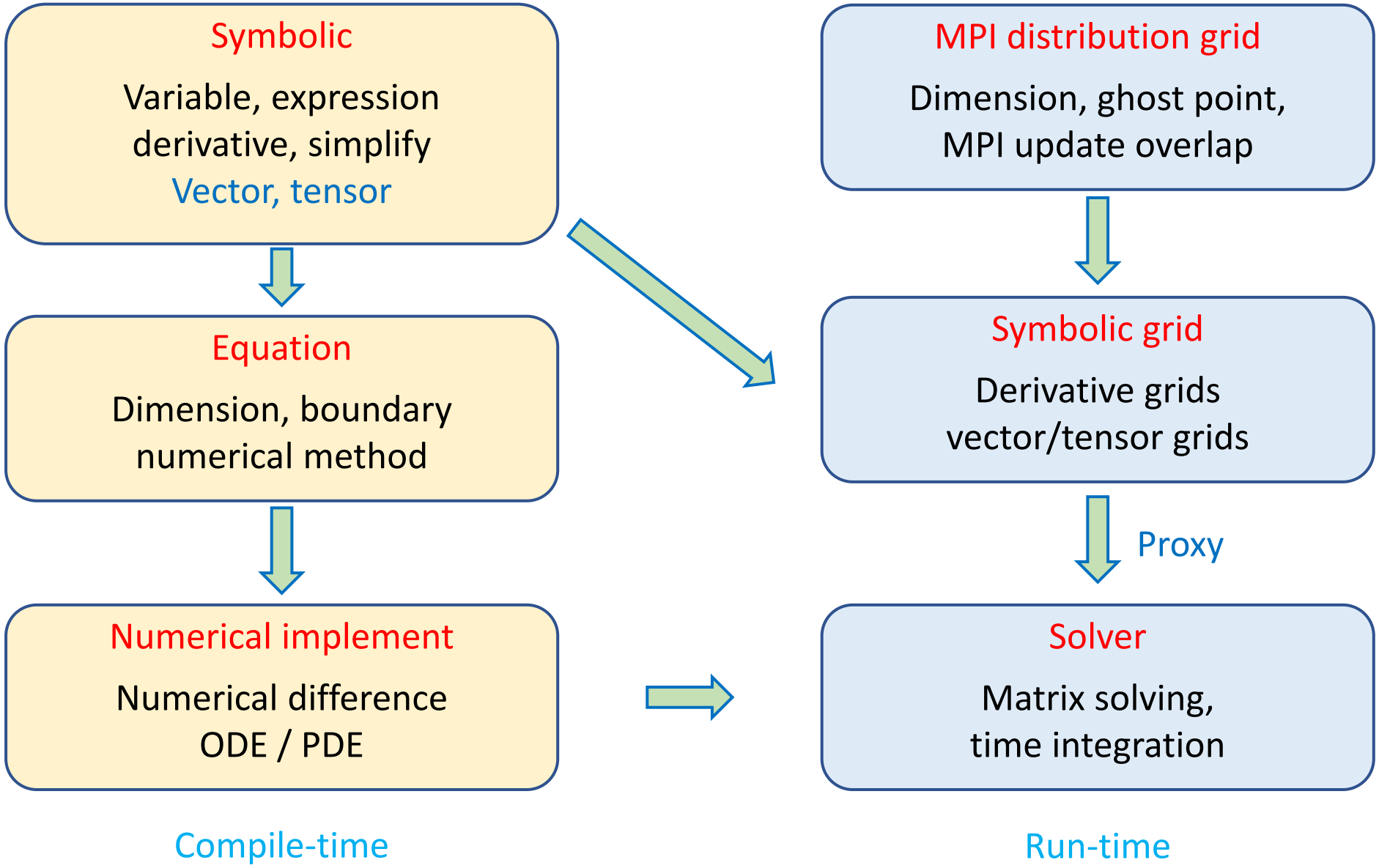


- PO — Process-oriented
- OO — Object-oriented
- DO — Data-oriented
- CA — Compile-time abstract

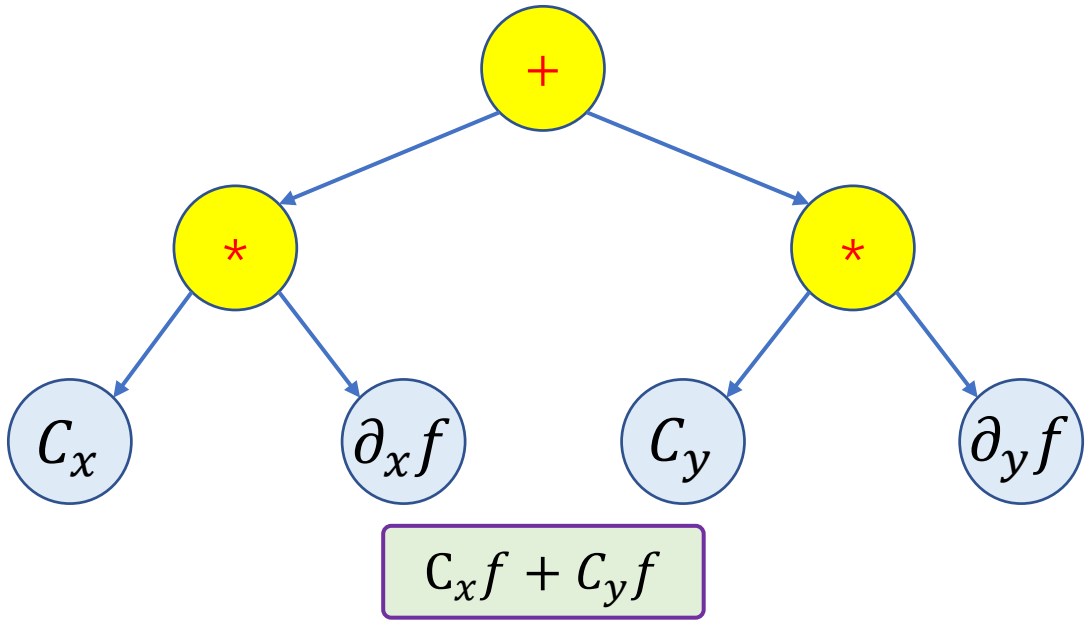
Outline

1. Background
2. Compile-time Symbolic Solver CSS
3. Optimization for fluid and particle simulations
4. Shifted metric method
5. Gyrokinetic MHD hybrid code GMEC
6. Field and particle code FP3D
7. Conclusion

CSS: structure



CSS: Compile-time Expression Tree



Op Node: Operator

V Leaf: Variable or Function

A data structure to store scalar expressions.

Leaf:

- Symbolic variables
- Symbolic constant
- Function

Node:

- Binary operations: $+, -, \times, /$
- Unary operations: $\cos, \sin, \exp, \mathrm{pow}$, etc

Evaluation

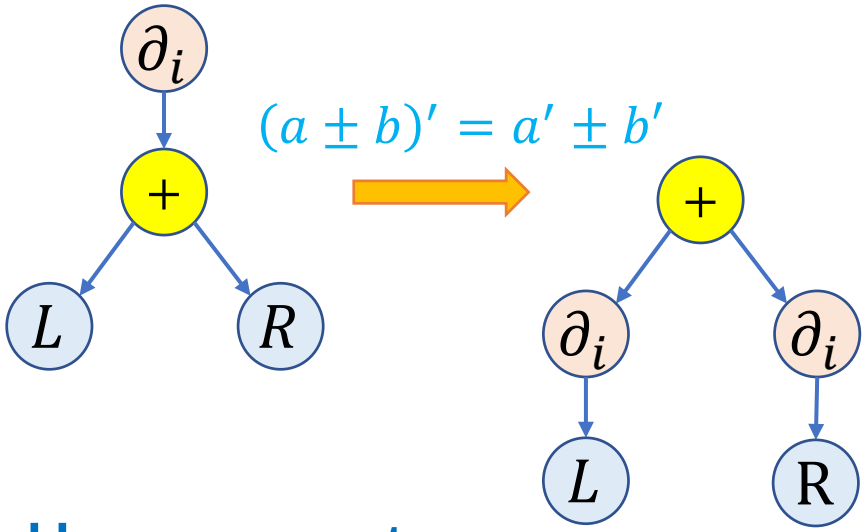
```
double operator()(coordinate r){
   return left(r) + right(r);
}
```

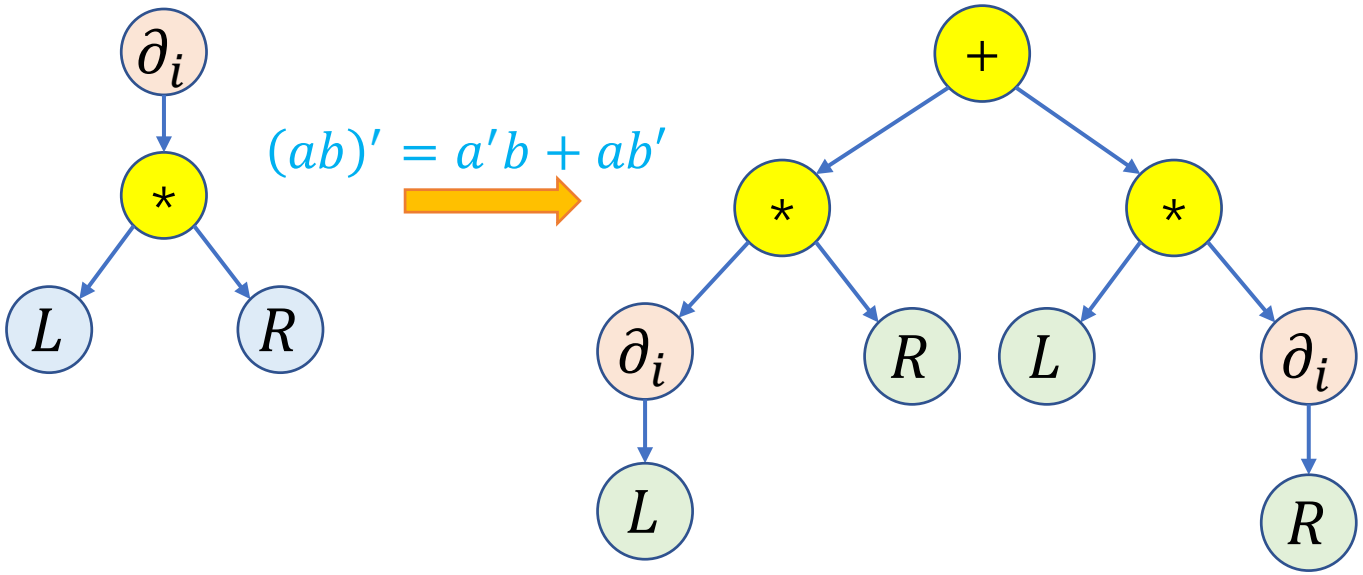
```
_mm256_add_ps(a, b);
```

- CET can be evaluated recursively.
- Evaluation function can be overloaded.
- Changing the structure of CET will change the instructions.
- Instruction optimizations

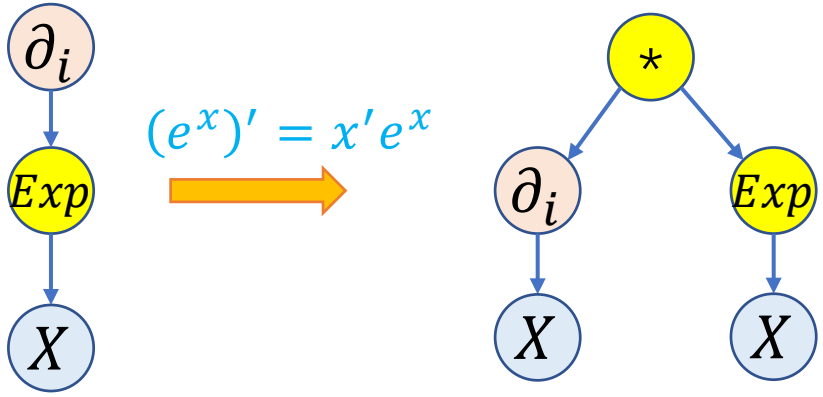
CSS: symbolic derivative

$Derivative: Exp \rightarrow var \rightarrow order \rightarrow Exp$





Unary operator

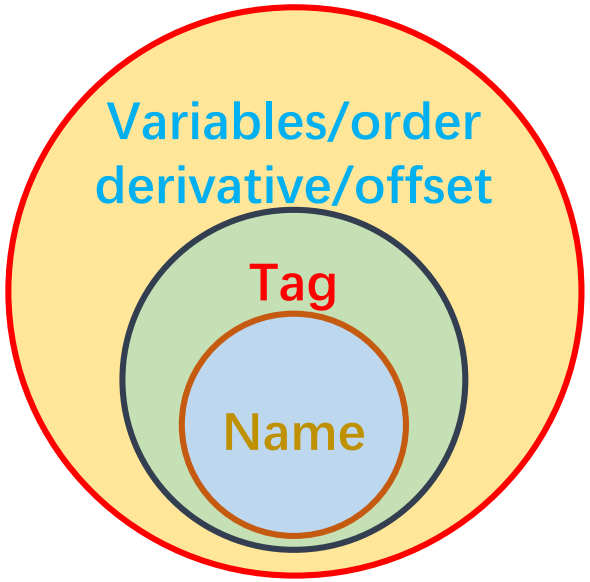


Exponential function

CSS supports custom binary and unary operators, the rules for symbolic derivative and evaluation function are needed.

CSS: leaf in expression tree

Symbolic variables, constants and functions



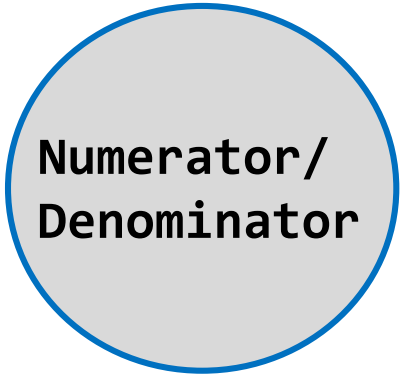
Variable

$$\left(f^{(d_i,\dots)}_{(p_i,\dots)}[\xi_i,\dots]\right)^n$$

$f(x, y, z, v_{\parallel}, \mu)$

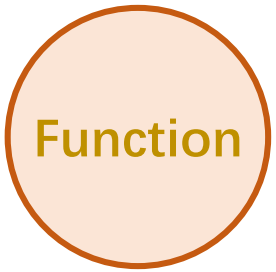
f: S, V_i, T_{ij}

$$\partial_{\xi_i} f^{(d_{\xi_j},\dots)}[\xi_j,\dots] = \begin{cases} f^{(d_{\xi_j}+1,\dots)}[\xi_j,\dots], & (\xi_i \in \{\xi_j\}) \\ 0, & (\xi_i \notin \{\xi_j\}) \end{cases}$$



Constant: C
ratio number

$\partial_{\xi_i} C = 0$



$Func: Exp \to Exp$

∂_x, ∂_x^2

The multiple sign is interpreted as function call if its left parameter is a function.

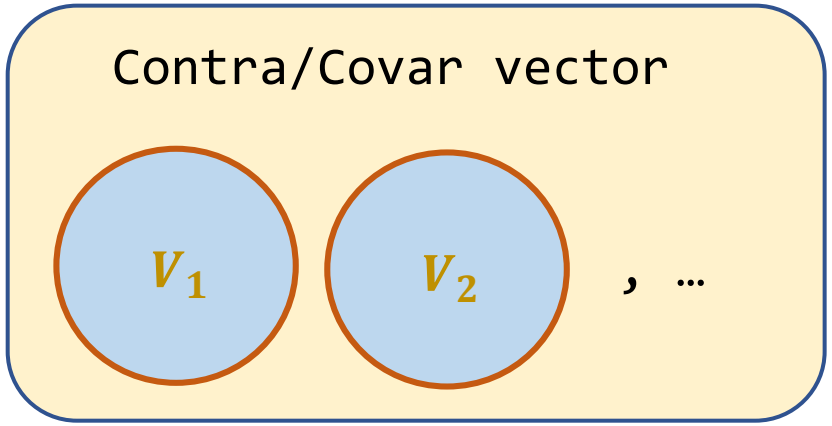
$$f * a = f(a)$$

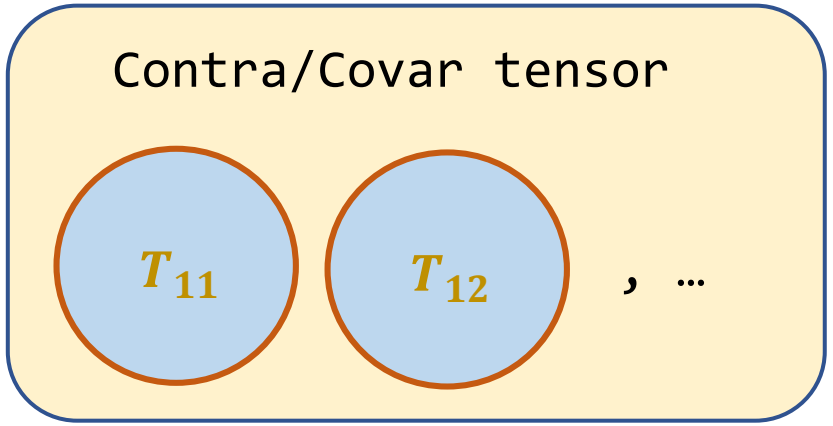
If both parameters are function, the result is a composed function.

$$f * g: (b \to c) \to (a \to b) \to (a \to c)$$

Allow custom operators.

CSS: symbolic vector and tensor





Covar

$$\begin{bmatrix} v_x & v_y & v_z \end{bmatrix}$$

$$\begin{bmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{bmatrix}$$

3d vector/tensor

- A vector consists of scalar components.
  The number of components depends on the coordinate dimension.

- Vectors and tensors have both contravariant and covariant forms.
  The transformation is done automatically.

- The components of vector/tensor can be different with the independent variables of component variable.

$g^{xz}[x, y]$

CSS: Vector/tensor operations

Dot product: $\vec{A} \cdot \vec{B} = A^i B_j = A_i B^j = A_i B_j g^{ij} = A^i B^i g_{ij}$

Cross product: $\vec{A} \times \vec{B} = \frac{1}{J} \epsilon_{ijk} A_j B_k = J \epsilon_{ijk} A^j B^k$

Operator ∇: covariant vector $\nabla = \partial_i$

Gradient: $\nabla f = \partial_i f$

Divergence: $\nabla \cdot \vec{v} = \frac{1}{J} \partial_i (J v^i)$

Curl: $\nabla \times \vec{v} = \frac{1}{J} \epsilon_{ijk} \partial_j v_k$

$A * B$

$Cross(A, B)$

$\nabla_{3d} =$ Covar $[\partial_x \quad \partial_y \quad \partial_z]$

$Nabla * f$

$Nabla * (J * v)/J = Div(v)$

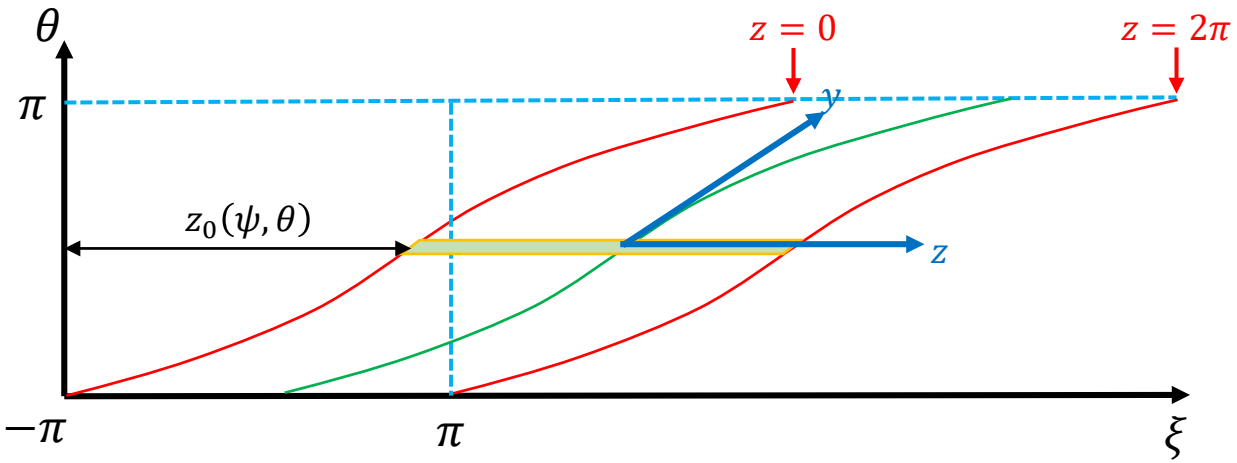
$Cross(Nabla, v)$

CSS: coordinates system

Coordinates system and boundary

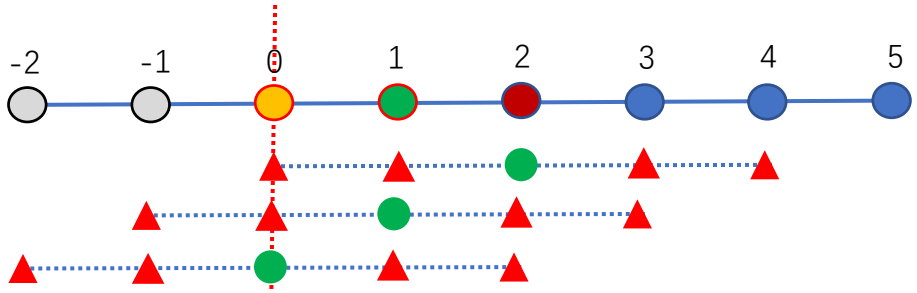
- Support arbitrary curvilinear coordinates.
- Support field-align coordinate with shifted metric.
- Support multiple boundary conditions:

Dirichlet, Neumann, periodic, twist-shift, etc.



$f(y + 2\pi, z) = f(y, z + 2\pi q)$ twist-shift bounary

CSS: numerical difference near boundary



In periodic boundary, the ghost points is used to simplify numerical difference.

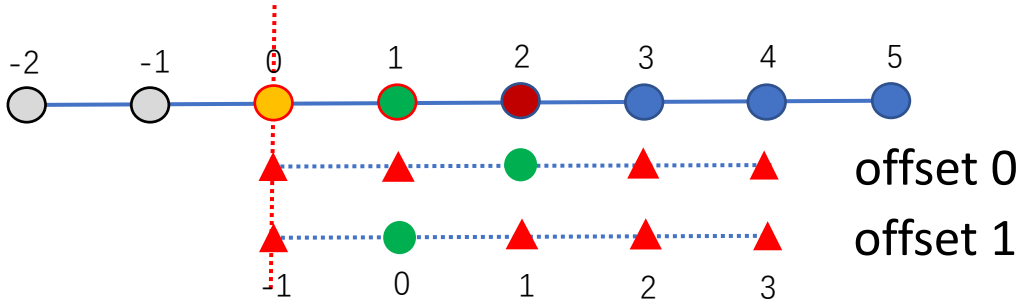
The number of ghost points

$$N_g = \lfloor N/2 \rfloor$$

where N is the number of stencil needed by numerical difference.

Boundary: Periodic, Twist-shift

$$f(x, y + 2\pi, z) = f(x, y, z + 2\pi q)$$



In non-periodic boundary, the biased difference scheme is needed which has totally different coefficients than central scheme. These points must be treated carefully.

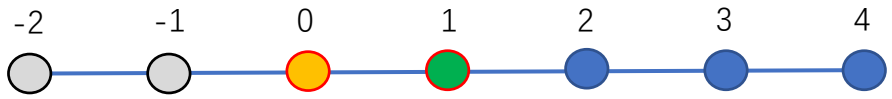
The boundary point 0 need to be treated specially by boundary condition.

Boundary: Dirichlet, Neumann

CSS: numerical difference

Solving coefficients of numerical difference

- Support classical N-point with p-offset numerical difference
- Coefficients can be set manually for custom method
- Others methods such as WENO will be included in future

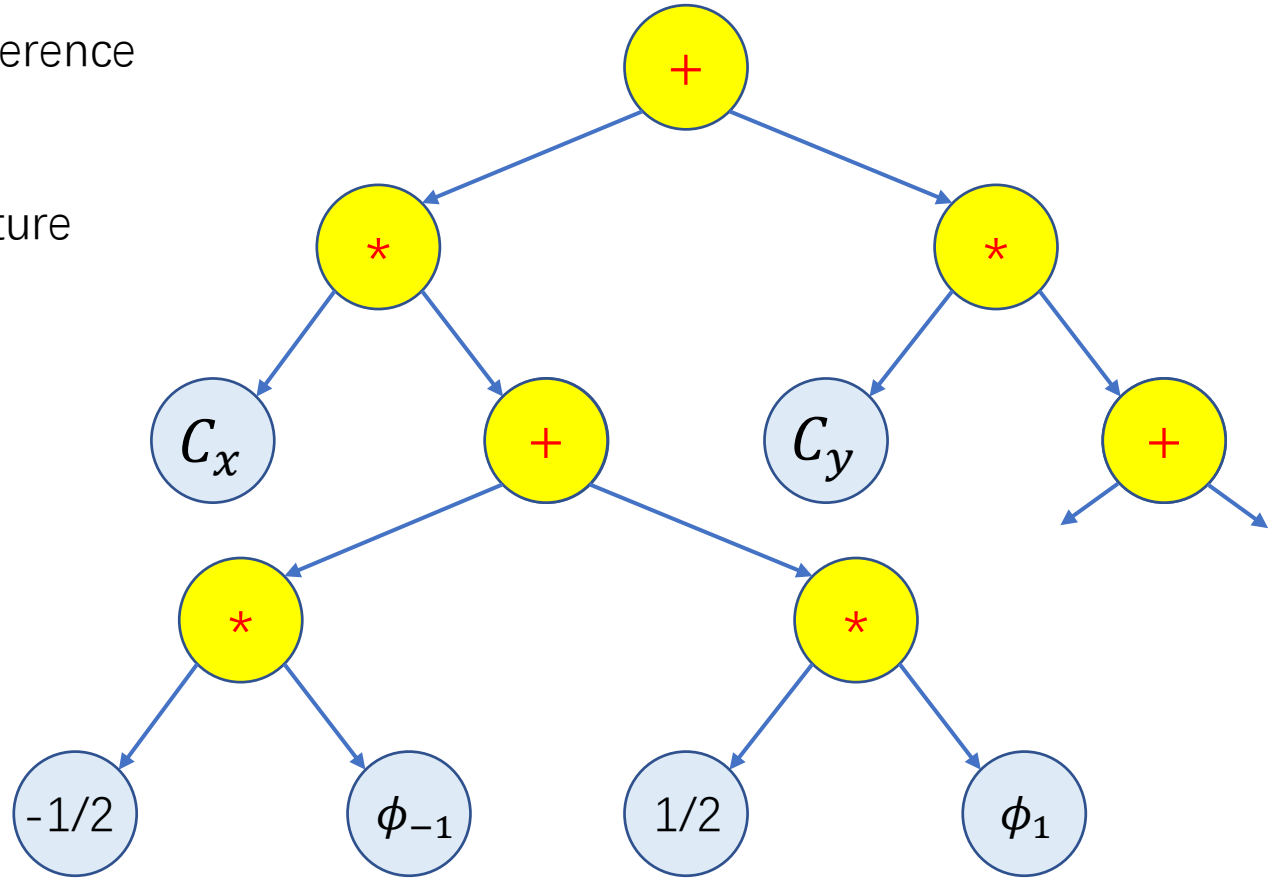


$$\begin{bmatrix} s_1^0 & \cdots & s_N^0 \\ \vdots & \ddots & \vdots \\ s_1^{N-1} & \cdots & s_N^{N-1} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix} = d! \begin{bmatrix} \delta_{0,d} \\ \vdots \\ \delta_{i,d} \\ \vdots \\ \delta_{N-1,d} \end{bmatrix}$$

System of linear equations for coefficients

Compile-time Linear Algebra Solver

$C_x\phi_x + C_y\phi_y$

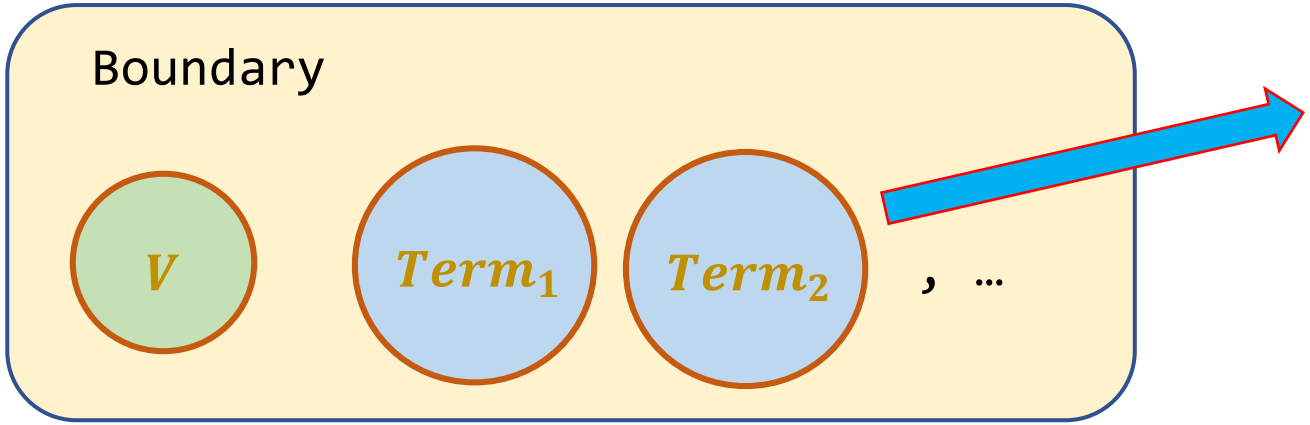


Numerical difference implement in CET

$Tree\ a \gg= (a \rightarrow Tree\ b) \rightarrow Tree\ b$

CSS: symbolic equations

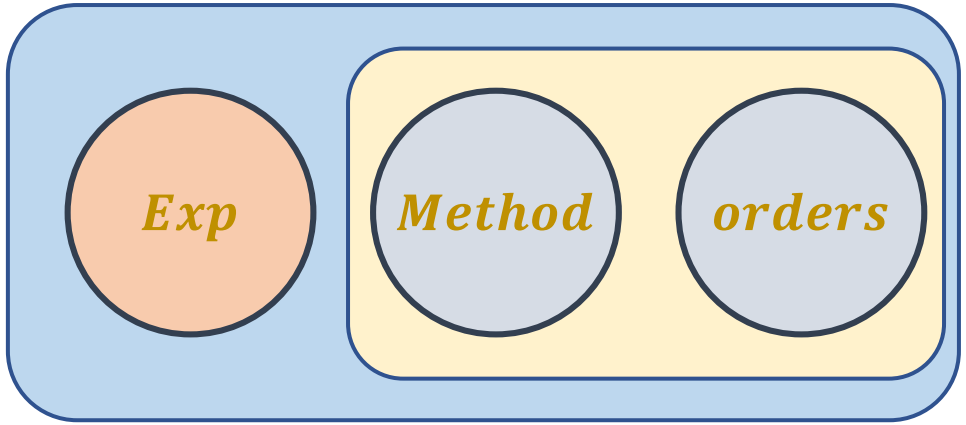
Symbolic equation



$$V = Term_1 + Term_2 + \cdots$$

In an equation, each term has independent numerical differential method and order.

Symbolic Term



A symbolic terms including symbolic expression, numerical differential method with orders in each direction.

CSS: Hybrid parallelization

Hybrid MPI and TBB parallelization

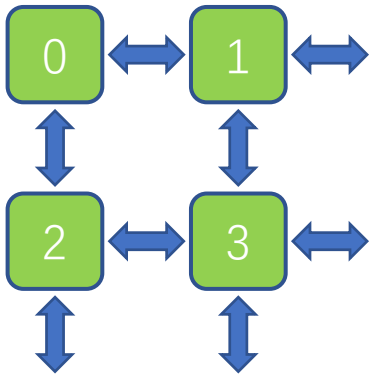
Memory parallel: N-dimensional MPI distribution grid

- Support N-dimensions MPI distribution grids and local grids;

  The sequence of dimensions is arbitrary;
- Ghost grid updates automatically;
- Parallelism can be set in arbitrary dimension.

$d\phi[z,x,y]$

$g_{ij}[x,y]$, $\partial_x g_{ij}[x,y]$




```
MPI_Cart_create
MPI_Dims_create
MPI_Cart_shift
```

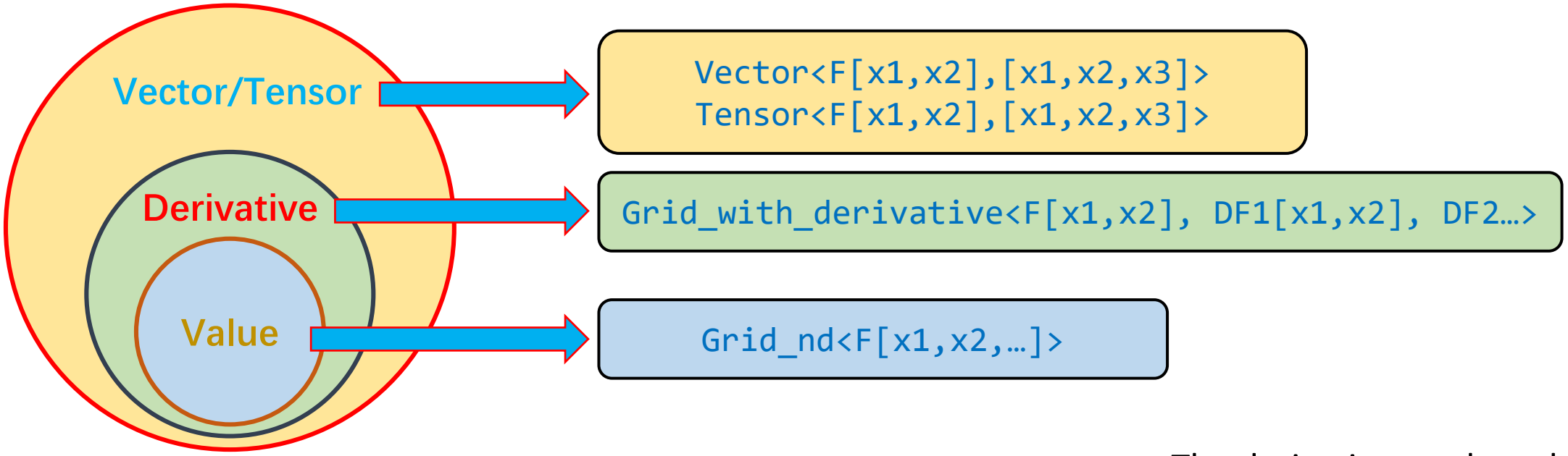
Task parallel: TBB shared memory parallel

- For each task including coordinates range and mission (as a function of coordinates).
- TBB invokes this function in parallel on each node.

TBB: Threading building block

Similar to OPENMP

CSS: Grid container



The derivative can be calculated accurately outside the program.

Coordinate variables can be different from vector/tensor components.

$$g_{xz}[x, y]$$

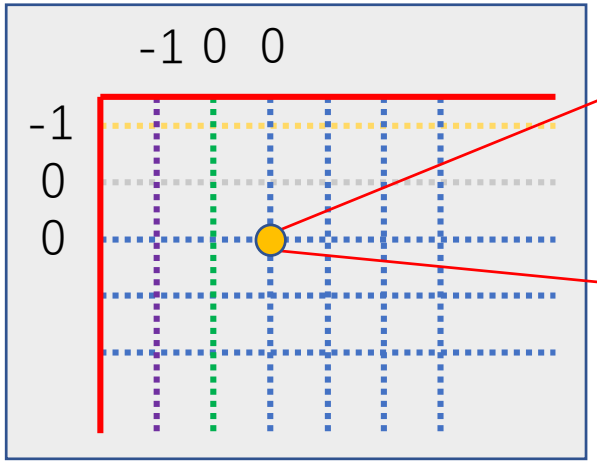
Access function: operator()

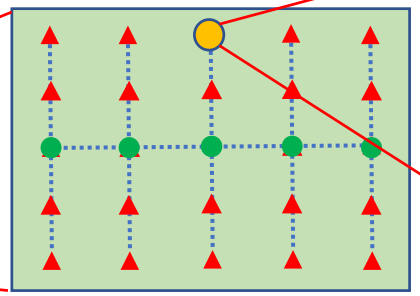
```
Auto operator()(auto var, auto coordinate)
```

Support N-dimensions B-spline with n-orders.

CSS: solving equations

Explicit equation and implicit equation





5 points scheme

$$C_{2,0,0} = -\frac{1}{12}\left[\frac{1}{J}\left(\frac{\partial J}{\partial x}g_{xx} + \frac{\partial J}{\partial y}g_{yx} + \frac{\partial J}{\partial z}g_{zx}\right) + \left(\frac{\partial g_{xx}}{\partial x} + \frac{\partial g_{yx}}{\partial y} + \frac{\partial g_{zx}}{\partial z}\right) + (g_{xx})\right]$$

$$\nabla_\perp^2\phi = \mathrm{J}^{-1}\partial_i\left(Jg^{ij}\partial_j\phi\right)$$

$$= \mathrm{J}^{-1}\partial_i\left(Jg^{ij}\right)\partial_j\phi + g^{ij}\partial_i\partial_j\phi$$

$$= C_x\partial_x\phi + C_y\partial_y\phi + C_{xx}\partial_x^2\phi + C_{yy}\partial_y^2\phi + C_{xy}\partial_x\partial_y\phi$$

$$= C_{-2,0,-2}\phi_{-2,0,-2} + \cdots + C_{2,0,2}\phi_{2,0,2}$$

$$\frac{\partial A}{\partial t} = \nabla_\perp^2\phi \qquad A\phi = b$$

CSS transforms vector equations into scalar forms, then it implements numerical difference with appropriate offset. Finally, CSS execute symbolic simplification such as expanding and merging coefficients.

For explicit equations, CSS evaluate the final expression directly.

For implicit equation, CSS uses these coefficients to load matrix automatically.

CSS: solvers

ODEs and PDEs solver

- Support arbitrary ODEs.
- Time integration method: RK4, CVODE(future), etc.
- Support arbitrary PDEs. Matrix loading is done automatically.
- PDE solver: MKL-PARDISO, CUDA-cuBLAS, PETSC(future).

Symbolic conclusion

Generate code easily and correctly

- Expanding equations and simplification automatically.
- Multiple dimensional grids with multiple parallel scheme.

Increasing runtime speed

- Automatic instruction optimization.

Outline

1 Background

2 Compile-time Symbolic Solver CSS

3 Optimization for fluid and particle simulations

4 Shifted metric method

5 Gyrokinetic MHD hybrid code GMEC

6 Field and particle code FP3D

7 Conclusion

Overview of acceleration methods

Acceleration of high-performance computing (HPC) includes parallelization, single instruction multiple data (SIMD) and memory access optimization.

- parallelization: Important, the main acceleration method, easy to implement.
- single instruction multiple data (SIMD): Important, but the compiler will implement it automatically in CPU if the calculation sequence is simple.
- memory access optimization: Important, hard to implement.

Method	type	effect	implement
parallelization	calculation	high	easy
SIMD	calculation	middle	by compiler
memory access	memory	high	hard

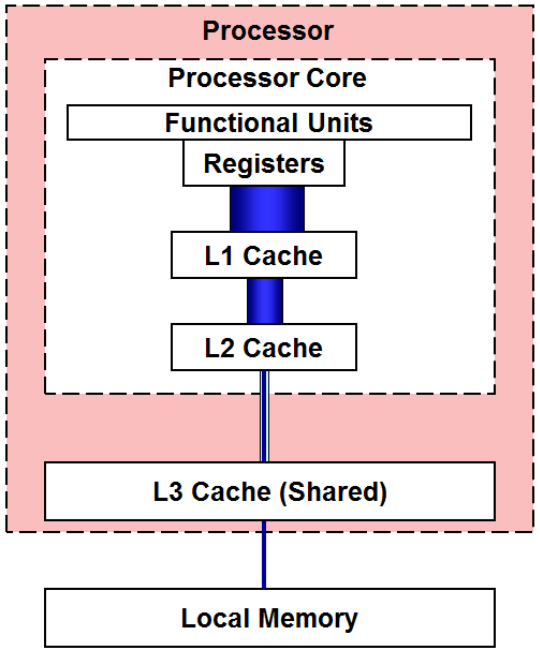
The computational efficiency ratio for typical code 5%

All of these acceleration method can coexist. Data Oriented Programming, DOP.

Optimization methods

CPU operation cost:

- Addition ≪ Multiplication ≈ L1 read ≪ L2 read ≪ L3 read ≪ memory read



- An cache line (64 bytes) is the minimal unit in each memory access;

- The calculation and access can be done at the same time.
- Memory prefetch: sequential access.
- Compiler automatic SIMD: sequential access.

Optimization target:

- decrease the number of multiplications
- Local access is good.
- sequential access is best.

Optimization strategies

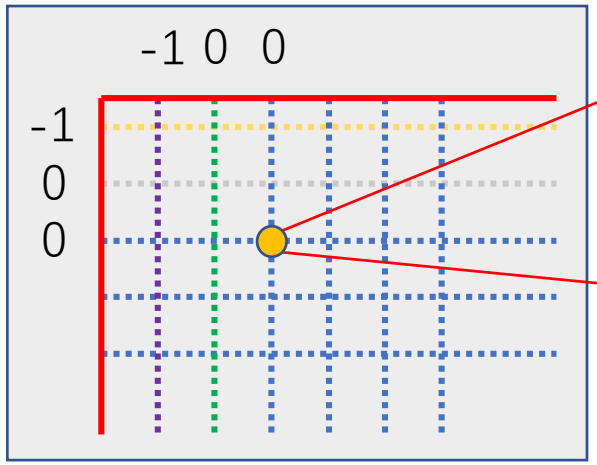
Fluid

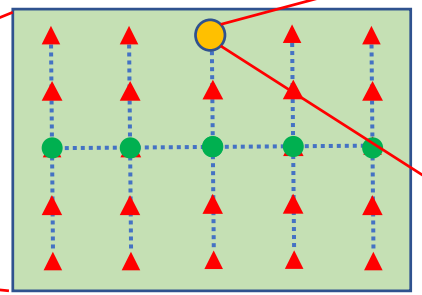
- Level 1: merging of coefficients;
- Level 2: optimization for duplicate access;
- Level 3: optimization for 1d grid iteration;
- Level 4: optimization for n-dimension grid iteration;
- Level 5: optimization for catch hitting ratio;

PIC

- Redundant data structure; boozer coordinates;
- Parallel counting sort;
- Space filling curves; (gyro average)

Level 1: merging of coefficients





5 points

$$C_{2,0,0} = -\frac{1}{12}\left[\frac{1}{J}\left(\frac{\partial J}{\partial x}g_{xx} + \frac{\partial J}{\partial y}g_{yx} + \frac{\partial J}{\partial z}g_{zx}\right) + \left(\frac{\partial g_{xx}}{\partial x} + \frac{\partial g_{yx}}{\partial y} + \frac{\partial g_{zx}}{\partial z}\right) + (g_{xx})\right]$$

For ODE, CSS creates coefficient arrays and calculates these coefficients at the beginning of program. It ensures the number of multiplications is minimal and leads to faster codes.

$$\frac{\partial A}{\partial t} = \nabla_\perp^2 \phi = \mathrm{J}^{-1}\partial_i\left(J g^{ij}\partial_j \phi\right)$$

$$= \mathrm{J}^{-1}\partial_i\left(J g^{ij}\right)\partial_j\phi + g^{ij}\partial_i\partial_j\phi$$ → 50 times multiplications, 45 times access; (merged)

$$= C_x\partial_x\phi + C_y\partial_y\phi + C_{xx}\partial_x^2\phi + C_{yy}\partial_y^2\phi + C_{xy}\partial_x\partial_y\phi$$ → 50 times multiplications, 45 times access;

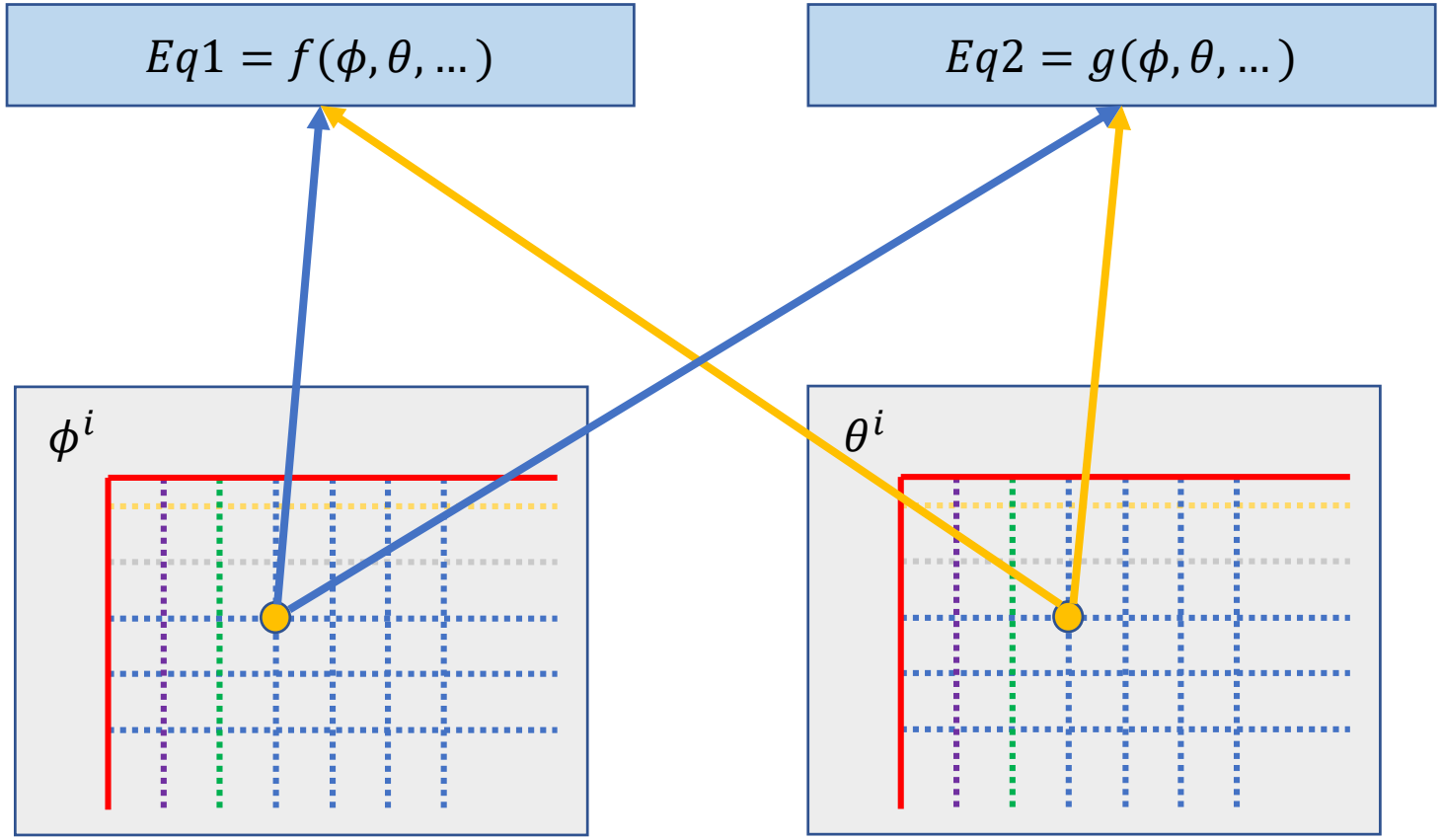
$$= C_{-2,0,-2}\phi_{-2,0,-2} + \cdots + C_{2,0,2}\phi_{2,0,2}$$ → 25 times multiplications, 25 times sequential access;

$$A\phi = b$$

Level 2: optimization for duplicate access

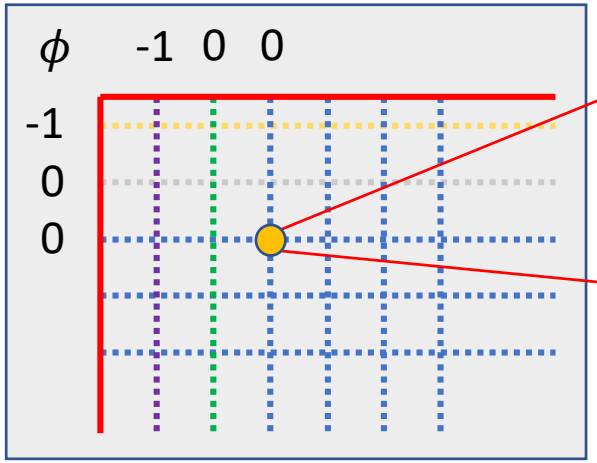
Multi equations with multi variables

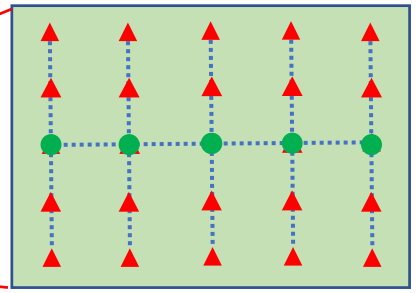
$$\partial_t y_1 = f(\phi, \theta, \dots)$$
$$\partial_t y_2 = g(\phi, \theta, \dots)$$



Grid access repeatedly if equations evaluate in sequence.

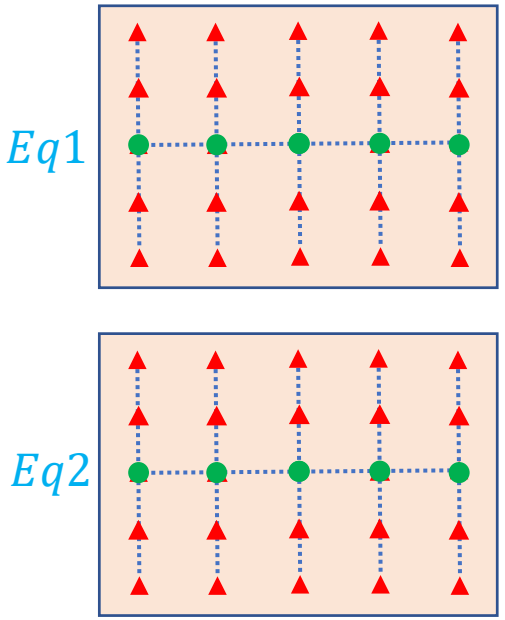
Level 2: optimization for duplicate access



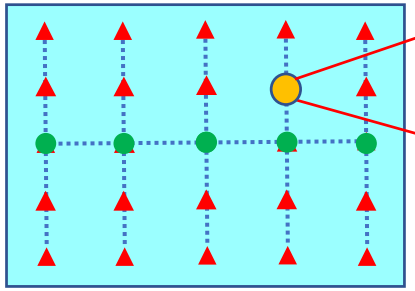


ϕ_{ijk}

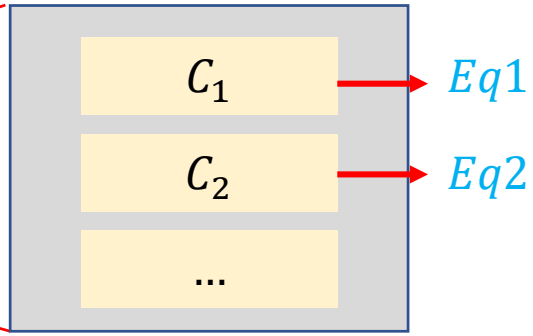
$\Leftrightarrow$



$C_{\phi_{ijk}}$



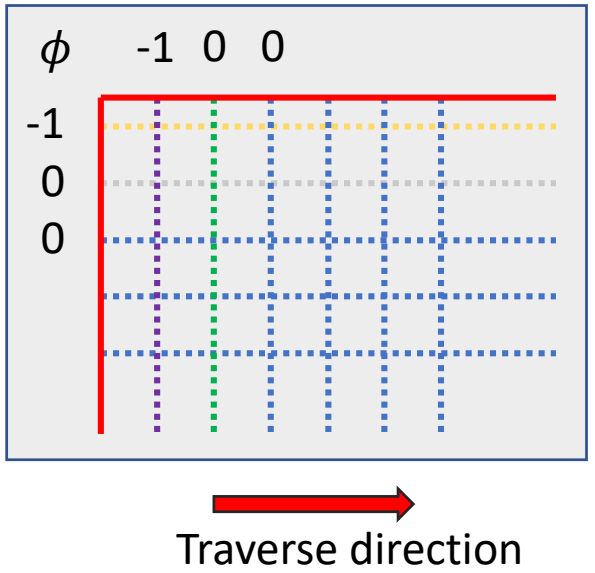
$C_{\phi_{ijk}}$

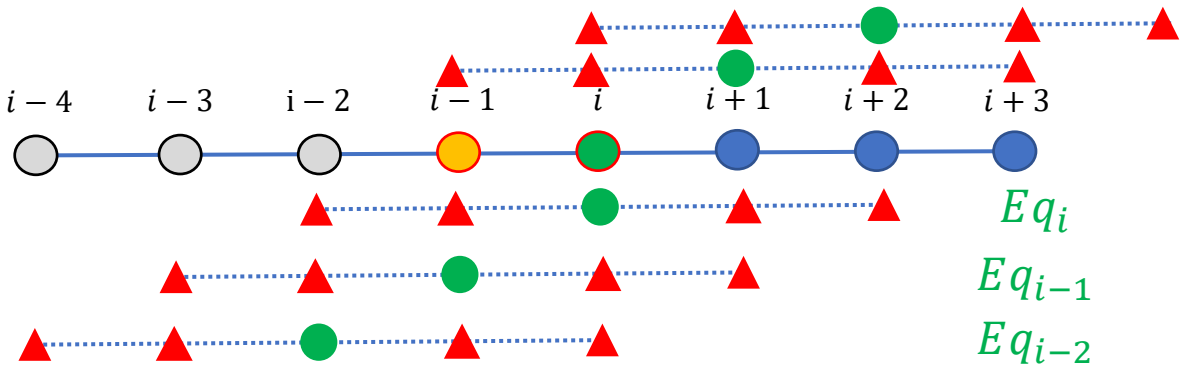


$C_{\phi_{110}}$

Create a coefficients array for each equations in each offsets point. Calculate ODE systems directly.

Level 3: optimization for 1d grid iteration





Assuming 1d stencil, the number of repeat access times is N where N is the number of numerical difference points.

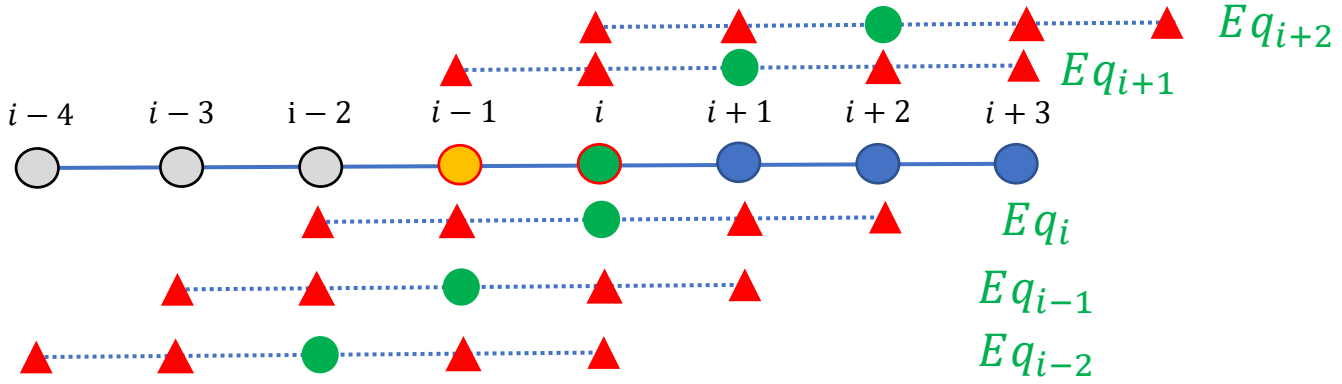
Dividing equations by the value of offsets.

$$Eqs = Eqs_{-2} + Eqs_{-1} + Eqs_0 + Eqs_1 + Eqs_2$$

Where

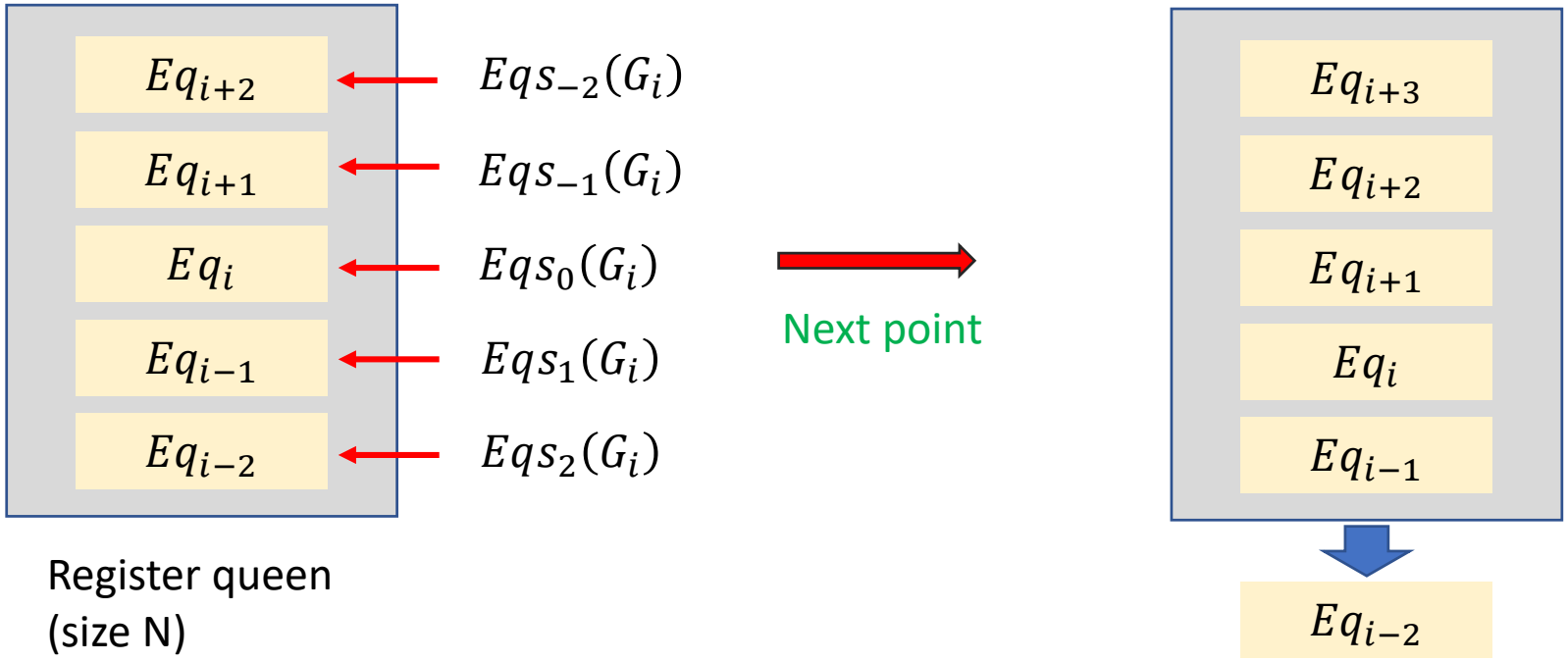
Eqs_i is the part with i offset.

Level 3: optimization for 1d grid iteration

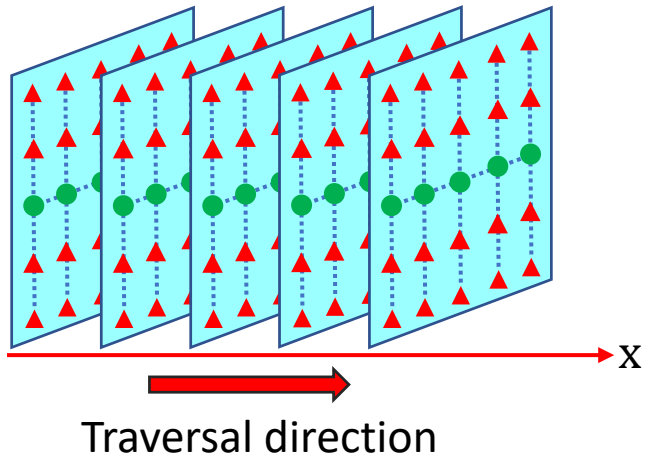


1. Reduce N-1 access times.
2. Transform stencil access into sequential access.

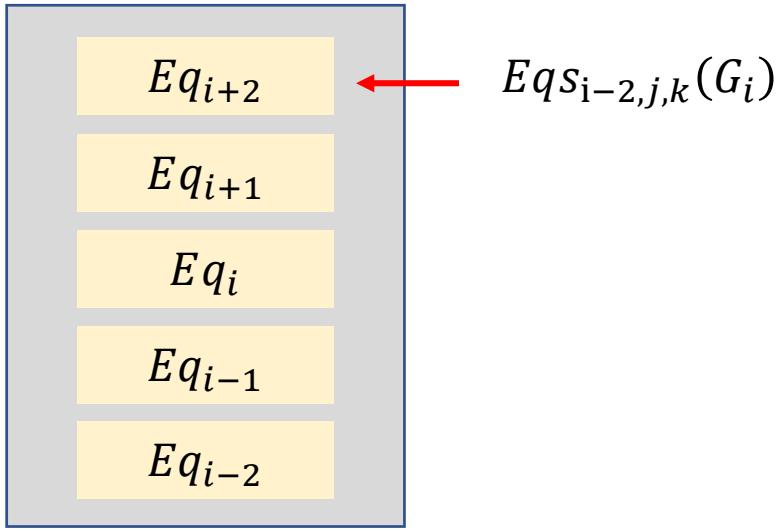
$$Eqs(G_i) = \sum_s Eqs_s(G_{i-s}) = Eqs_2(G_{i-2}) + Eqs_1(G_{i-1}) + Eqs_0(G_i) + Eqs_{-1}(G_{i+1}) + Eqs_{-2}(G_{i+2})$$



Level 4: optimization for n-dimensions grid iteration



$$Eqs = Eqs_{i-2,j,k} + Eqs_{i-1,j,k} + Eqs_{i,j,k} + Eqs_{i+1,j,k} + Eqs_{i+2,j,k}$$

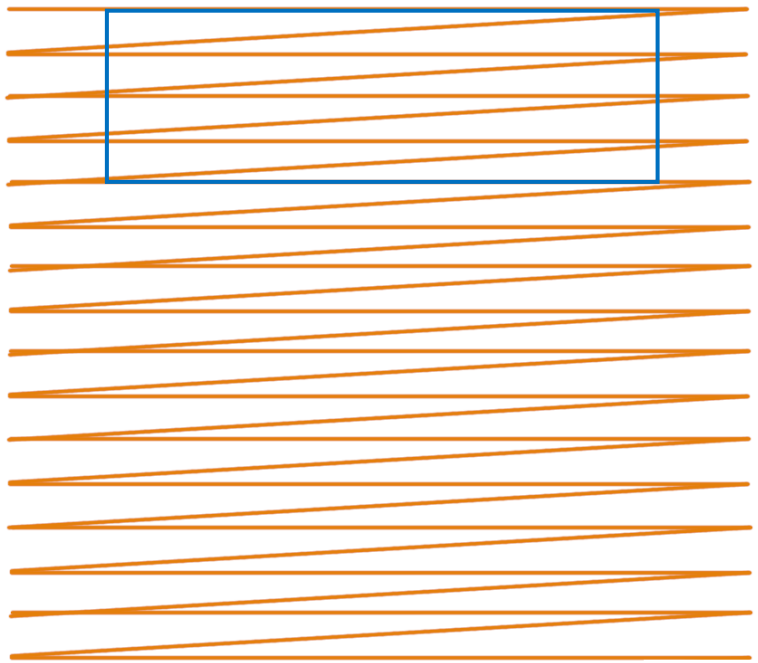


X dimension in continuous in memory. Traverse along x direction, the cache hitting ratio is good in x, but bad in y and z.

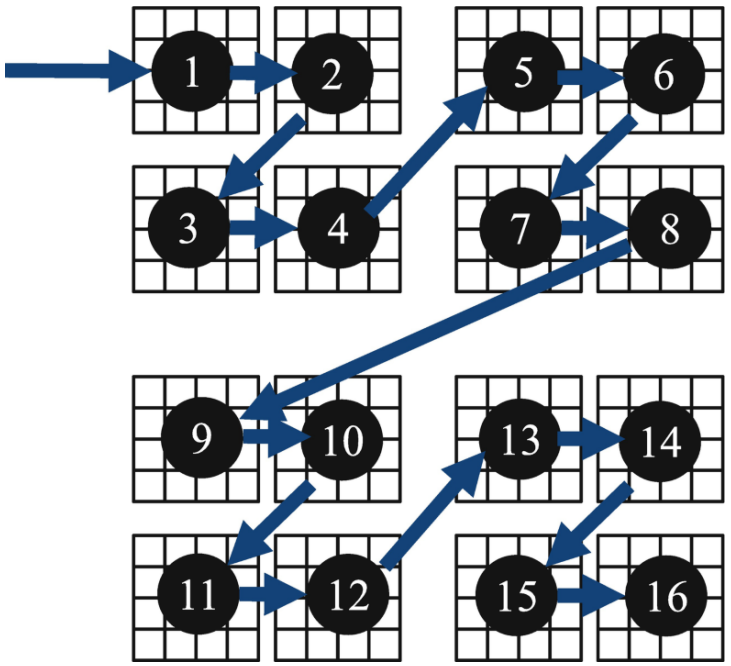
The accessing data should close enough in 3D space during the traversal.

Level 4: optimization for n-dimensions grid iteration

Morton space-filling code



Traverse 2D array in normal sequence



Traverse 2D array in Morton code

$$x = x_1 x_2 x_3, \qquad y = y_1 y_2 y_3$$

$$m(x, y) = y_1 x_1 y_2 x_2 y_3 x_3$$

Morton code map multidimensional data to one dimension while preserving locality of the data points.

Increasing catch hitting ratio obviously.

The larger the cache, the higher the cache hit rate

Level 5: optimization for catch hitting ratio

The larger the cache, the higher the cache hit rate.

The cache size is not easy to increase, but we can reduce cache requirements to increase cache utilization

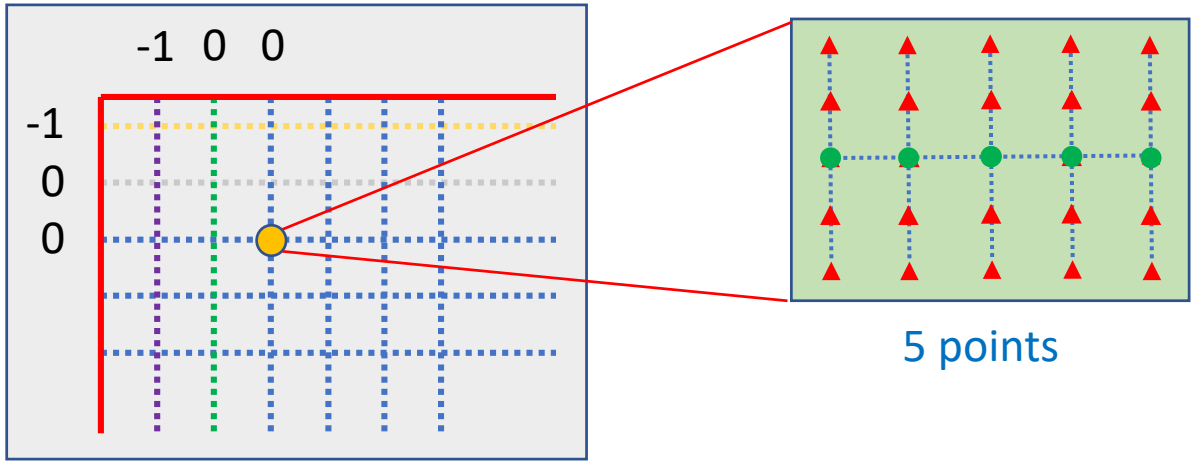
$$Eqs = f(\phi, \theta, \dots) = f_1(\phi) + f_2(\theta) + \cdots$$

Separate linear equation into many terms which only contains one variables. Evaluate them term by term.

One variable share the whole cache. The cache hit rate is N times that of directly evaluating the equation where N is the number of terms.

Optimization strategies for fluid

Conclusion



In finite difference method, each equation is a function which map the adjacent points to one point

$$\phi(\xi_{i+p}, \xi_{j+q}, \dots) \rightarrow \vec{y}(\xi_i, \xi_j, \dots)$$

The number of independent variables is n^d,

where n is the number of numerical difference points, d is the number of dimensions.

Need n^d times of multiplication at least.

$$\partial_t \vec{y} = \vec{f}(\phi, \theta, \dots)$$

$$\partial_t \vec{y}(\xi_i, \xi_j, \dots) = \vec{f}(\phi(\xi_i, \xi_j, \dots), \theta(\xi_i, \xi_j, \dots), \dots)$$

In finite difference method, each ODE evaluation is a function which map variable grids to the righthand variable grids

$$\phi(\xi_i, \xi_j, \dots), \theta(\xi_i, \xi_j, \dots), \dots \rightarrow \vec{f}(\xi_i, \xi_j, \dots)$$

The number of independent variables is $m \times N$,

where m is the number of equation variables $(\phi, \theta, \dots)$, N is the number of grid point for each variable.

Need $m \times N$ times of access at least.

Approach the theoretical maximal speed

Redundant data structure

Normal 3D grid: only one dimension is continuous in memory
low cache hitting ratio in the others two dimensions.

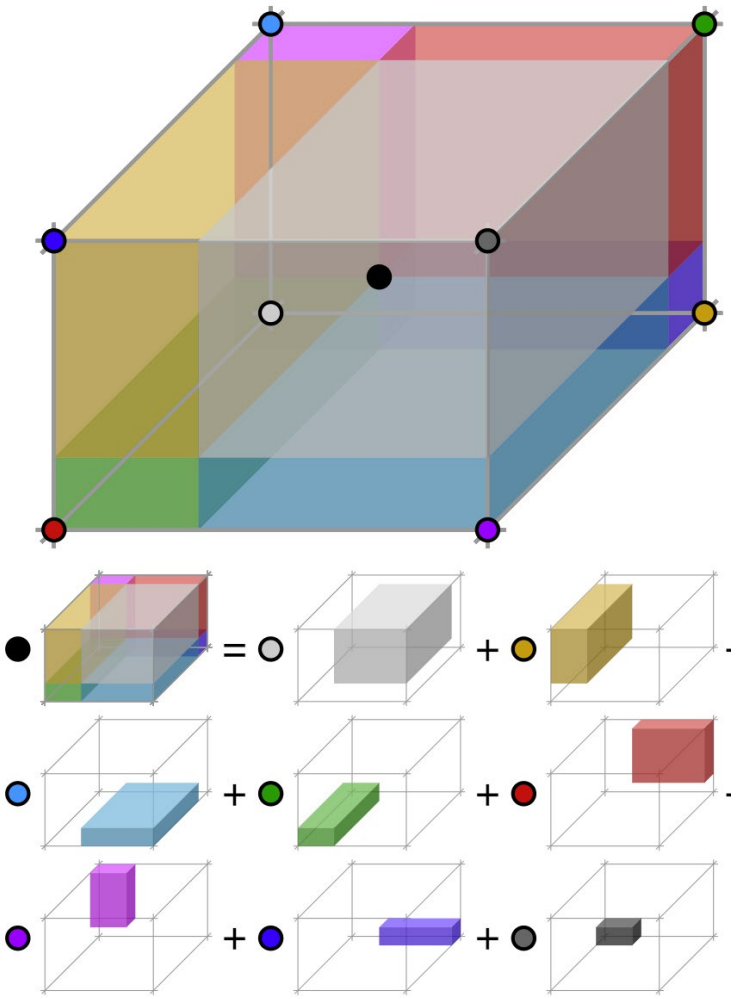
Redundant grid: store every vertex in local continuous array for each cell. The linear interpolation for particle in this cell has high cache hitting ratio.

The memory needed by redundant grid is eight times than normal grid (3D).

For $(N_\psi, N_\theta, N_\xi) = (256,64,256)$, memory need 0.5GB
For $N_p = 64$ particle per cell, memory need 15GB

Boozer coordinate: $q, J, B, \delta, I, G, \frac{\partial B}{\partial x}, \frac{\partial B}{\partial y}, \frac{\partial \delta}{\partial x}, \frac{dI}{dx}, \frac{dG}{dx}, \delta\phi, \delta A_{\parallel}$

Greatly lower the memory access requirements



Black point: particle position

Parallel counting sort

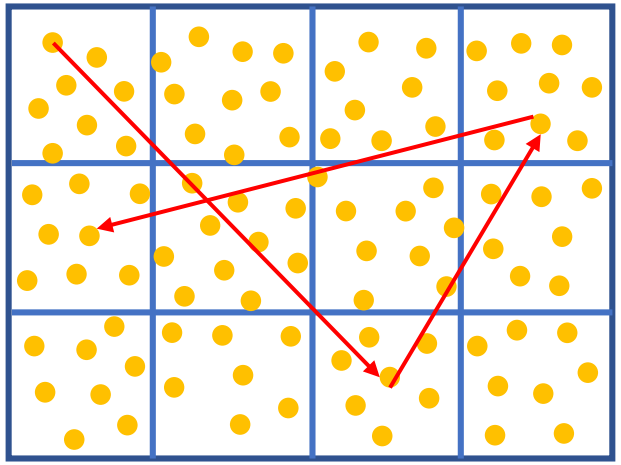
Particle loop by particle id:
Random access in grid memory, bad cache hitting.

Particle loop by cell id:
Local access in grid memory, good cache hitting.

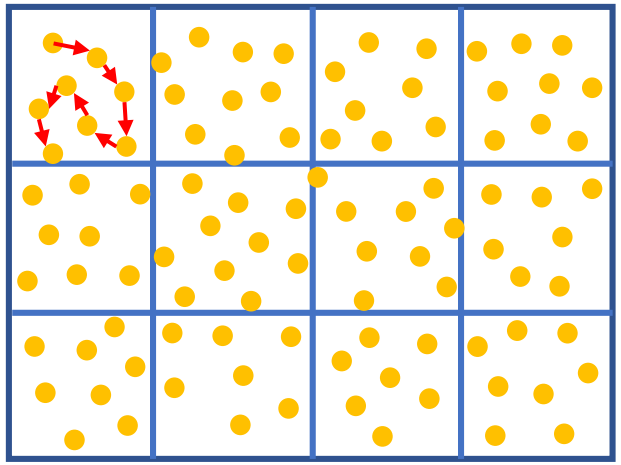
Particle need parallel counting sort base on cell id

$$id_{cell} = f\left(\left\lfloor\frac{\psi}{\Delta\psi}\right\rfloor, \left\lfloor\frac{\theta}{\Delta\theta}\right\rfloor, \left\lfloor\frac{\xi}{\Delta\xi}\right\rfloor\right)$$

Where f is linearization function for cell.



Iterate by particle id

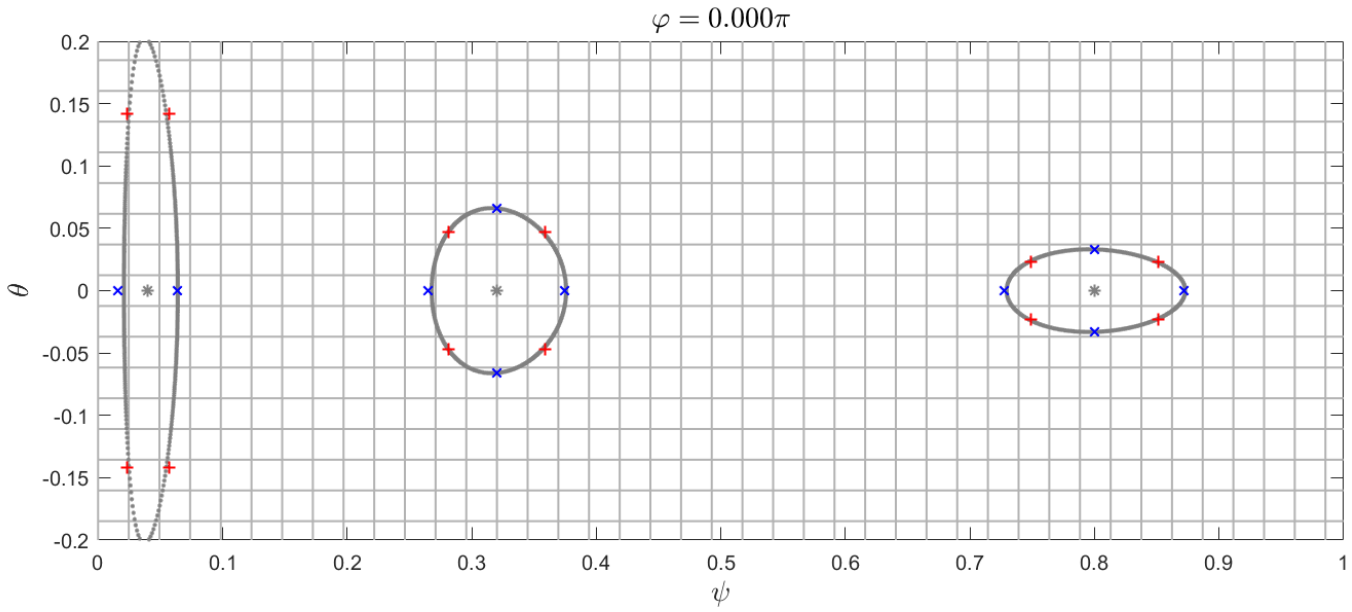


Iterate by cell id

Space filling curves

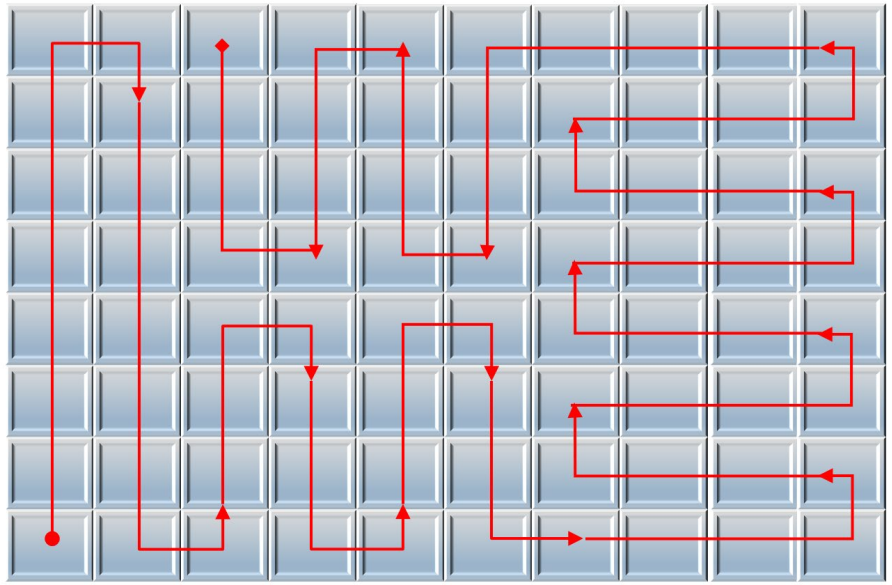
The gyro orbit crosses more y grids near the inner boundary and more x grids near the outer boundary.

Involving filling curves on the x-y plane to optimize gyro-average interpolation.



In areas where ψ is small,
memory is mainly continuous in θ direction.

In areas where ψ is large,
memory is mainly continuous in ψ direction.



θ −major order

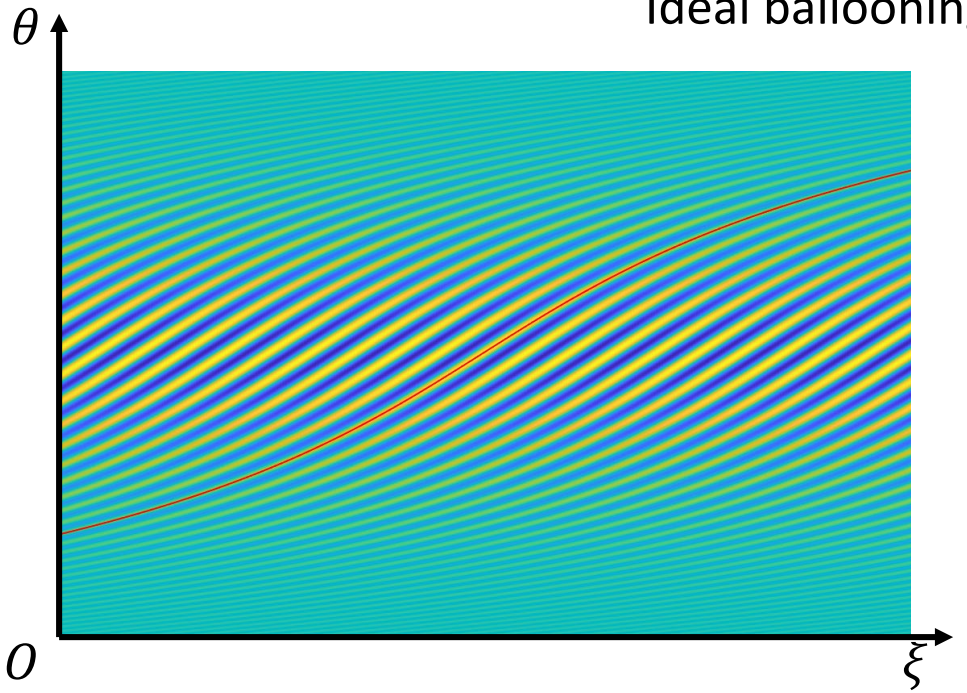
ψ −major order

Outline

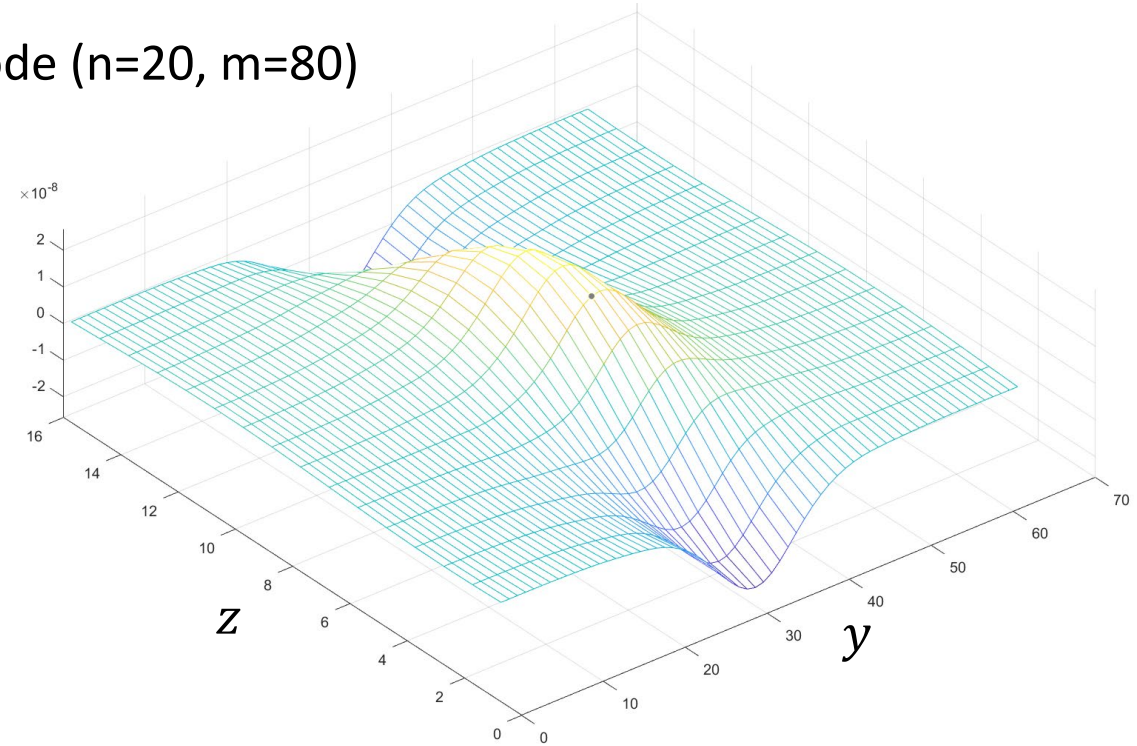
1. Background
2. Compile-time Symbolic Solver CSS
3. Optimization for fluid and particle simulations
4. Shifted metric method
5. Gyrokinetic MHD hybrid code GMEC
6. Field and particle code FP3D
7. Conclusion

Field-align coordinates

Ideal ballooning mode (n=20, m=80)



Mode structure in one magnetic surface
Non-straight field coordinate



Mode structure in field-align coordinate
(1 toroidal period)

Field-aligned coordinates brings the benefit that relatively few grids are needed in the parallel coordinate.

$$\nabla_{\parallel}^2 \ll \nabla_{\perp}^2$$

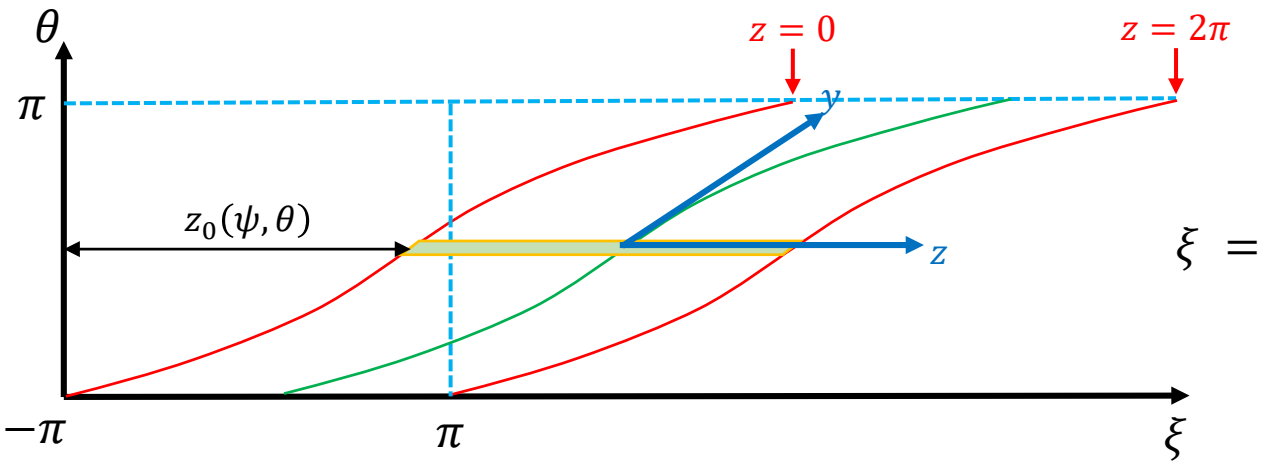
Field-aligned coordinate

Define field-aligned coordinates (x, y, z) from flux coordinate (ψ, θ, ξ)

$$\begin{cases} x = \psi - \psi_0 \\ y = \theta \\ z = \xi - \int_{\theta_0}^{\theta} \nu(\psi, \theta')d\theta' \end{cases}$$

Where

$$\nu(\psi, \theta) \equiv \frac{B \cdot \nabla\xi}{B \cdot \nabla\theta}$$



$$\xi = z + z_0$$

$$e_x = e_\psi + I_s(\psi, \theta)e_\xi$$
$$e_y = e_\theta + \nu(\psi, \theta)e_\xi$$
$$e_z = e_\xi$$

$$\vec{B} = \nu\nabla\psi \times \nabla\theta - \nabla\psi \times \nabla\phi$$
$$\vec{B} = \nabla z \times \nabla x = \frac{1}{J_{xyz}}\vec{e}_y$$

where

$$z_0(\psi, \theta) \equiv \int_{\theta_0}^{\theta} \nu(\psi, \theta')d\theta'$$

$$I_s(\psi, \theta) \equiv \int_{\theta_0}^{\theta} \frac{\partial\nu(\psi, \theta)}{\partial\psi}d\theta = \frac{\partial z_0(\psi, \theta)}{\partial\psi}$$

$$\partial_x = \partial_\psi + I_s(\psi, \theta)\partial_\xi$$
$$\partial_y = \partial_\theta + \nu(\psi, \theta)\partial_\xi = \partial_\parallel$$
$$\partial_z = \partial_\xi$$

$$f(x, y, z) = f(x, y, z + 2\pi)$$

$$f(x, y, z) = f(x, y + 2\pi, z + 2\pi q)$$ Twist-shifted

The field line in flux coordinate (ψ, θ, ξ) can be non-straight.

Field-align coordinates

The discontinuity of radial derivative

$$I_s(\psi,\theta) = \int_{\theta_0}^{\theta} \frac{\partial \nu(\psi,\theta)}{\partial \psi} d\theta = \frac{\partial z_0(\psi,\theta)}{\partial \psi}$$

$$I_s(\psi,\theta+2\pi) \neq I_s(\psi,\theta)$$

$$\theta_0 = \theta_i, \qquad I_s(\psi,\theta) = 0$$

$$\partial_x f\Big|_{\theta\to 0^+} = \partial_\psi f$$

$$\partial_x f\Big|_{\theta\to 0^-} = \partial_\psi f + I_s(\psi,2\pi)\partial_\xi f$$

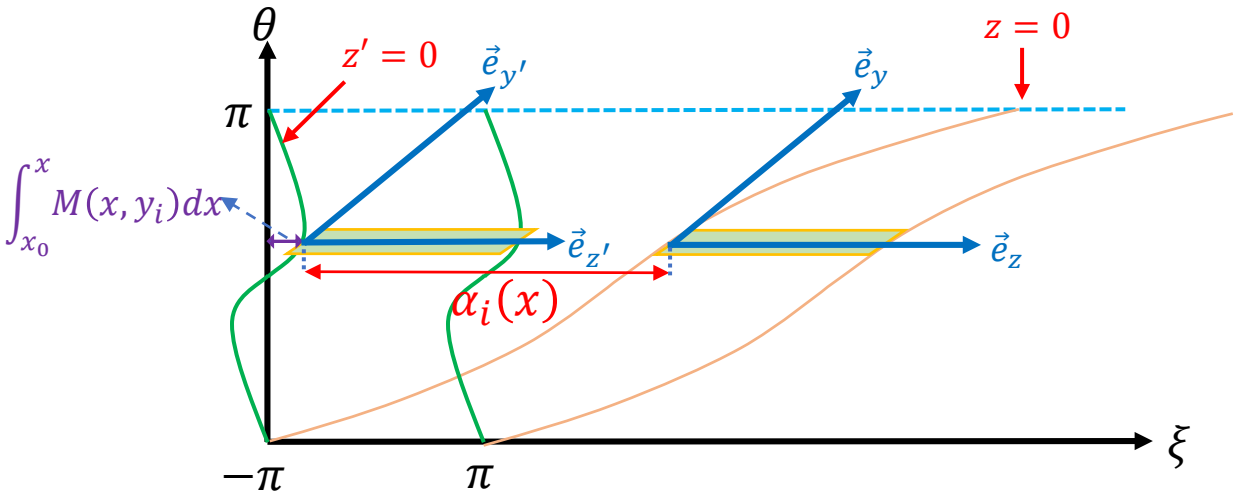
$$g^{xyz} = \begin{bmatrix} g^{\psi\psi} & g^{\psi\theta} & -I_s g^{\psi\psi} - \nu g^{\psi\theta} + g^{\psi\xi} \\ g^{\psi\theta} & g^{\theta\theta} & -I_s g^{\psi\theta} - \nu g^{\theta\theta} + g^{\theta\xi} \\ -I_s g^{\psi\psi} - \nu g^{\psi\theta} + g^{\psi\xi} & -I_s g^{\psi\theta} - \nu g^{\theta\theta} + g^{\theta\xi} & I_s^2 g^{\psi\psi} + 2I_s\left(\nu g^{\psi\theta} - g^{\psi\xi}\right) + \nu^2 g^{\theta\theta} - 2\nu g^{\theta\xi} + g^{\xi\xi} \end{bmatrix}$$

$$g^{xyz} = \begin{bmatrix} g^{\psi\psi} & 0 & -I_s g^{\psi\psi} \\ 0 & g^{\theta\theta} & -\nu g^{\theta\theta} \\ -I_s g^{\psi\psi} & -\nu g^{\theta\theta} & I_s^2 g^{\psi\psi} + \nu^2 g^{\theta\theta} + g^{\xi\xi} \end{bmatrix}$$

Orthogonal $\left(g^{\psi\theta} = g^{\psi\xi} = g^{\theta\xi} = 0\right)$

Shifted metric method

B. Scott shifted metric



For each y_i, define local coordinates

$$\begin{cases} x = \psi - \psi_0 \\ y = \theta \\ z = \xi - z_0(\psi, \theta) \end{cases} \qquad \begin{cases} x' = x \\ y' = y \\ z' = z - \alpha_i(x) \end{cases}$$

$$I_s(\psi, \theta) = \frac{\partial z_0(\psi, \theta)}{\partial \psi}$$

B. Scott 2001 POP

$$\boxed{\alpha_i'(x) = \left.\frac{g^{xz}}{g^{xx}}\right|_{y=y_i}} = \left.\left(-I - \nu\frac{g^{\psi\theta}}{g^{\psi\psi}} + \frac{g^{\psi\xi}}{g^{\psi\psi}}\right)\right|_{y=y_i}$$

$$\alpha_i(x) = \left.\int_{x_0}^{x} \frac{g^{xz}(x', y)}{g^{xx}(x', y)} dx'\right|_{y=y_i} \qquad g^{x'z'} = 0$$

$$z_i'(x) = \xi - \int_{x_0}^{x} M(x, y_i)dx$$

Where

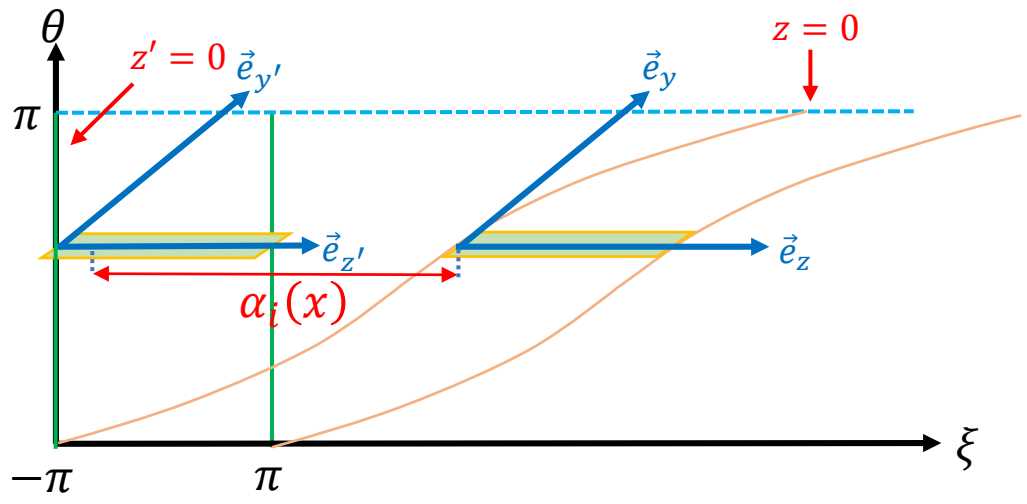
$$M(x, y) \equiv \frac{\partial \phi}{\partial x'} = -\nu\frac{g^{\psi\theta}}{g^{\psi\psi}} + \frac{g^{\psi\xi}}{g^{\psi\psi}}$$

$M(x, y)$ is continuous in boundary

$$\partial_{x'} = \partial_\psi + M\partial_\xi \approx \partial_\psi$$

Shifted metric method

Simple shifted metric



$$\begin{cases} x = \psi - \psi_0 \\ y = \theta \\ z = \xi - z_0(\psi, \theta) \end{cases} \qquad \begin{cases} x' = x \\ y' = y \\ z' = z - \alpha_i(x) \end{cases}$$

Local coordinate in $y = y_i$

$$z_0(\psi, \theta) \equiv \int_{\theta_0}^{\theta} \nu(\psi, \theta') d\theta' \qquad I_s(\psi, \theta) = \int_{\theta_0}^{\theta} \frac{\partial \nu(\psi, \theta)}{\partial \psi} d\theta$$

$$\boxed{\alpha_i'(x) = -I_s(x, y_i)}$$

$$\alpha_i(x) = \int_{x_0}^{x} \alpha_i'(x) dx' = -z_0(x, y_i)$$

$$z_i' = \xi - z_0(x, y) + z_0(x, y_i) = \xi - \int_{\theta_i}^{\theta} \nu(\psi, \theta') d\theta'$$

$$z_i' = \xi\Big|_{y'=y_i'}, M = 0\Big|_{y'=y_i'}$$

Simple shifted metric is same as field-align coordinates with $\theta_0 = \theta_i$.

$$\boxed{z_0' = 0, \qquad I_s' = 0, \qquad \partial_x I_s' = 0, \qquad \partial_y I_s' = \nu}$$

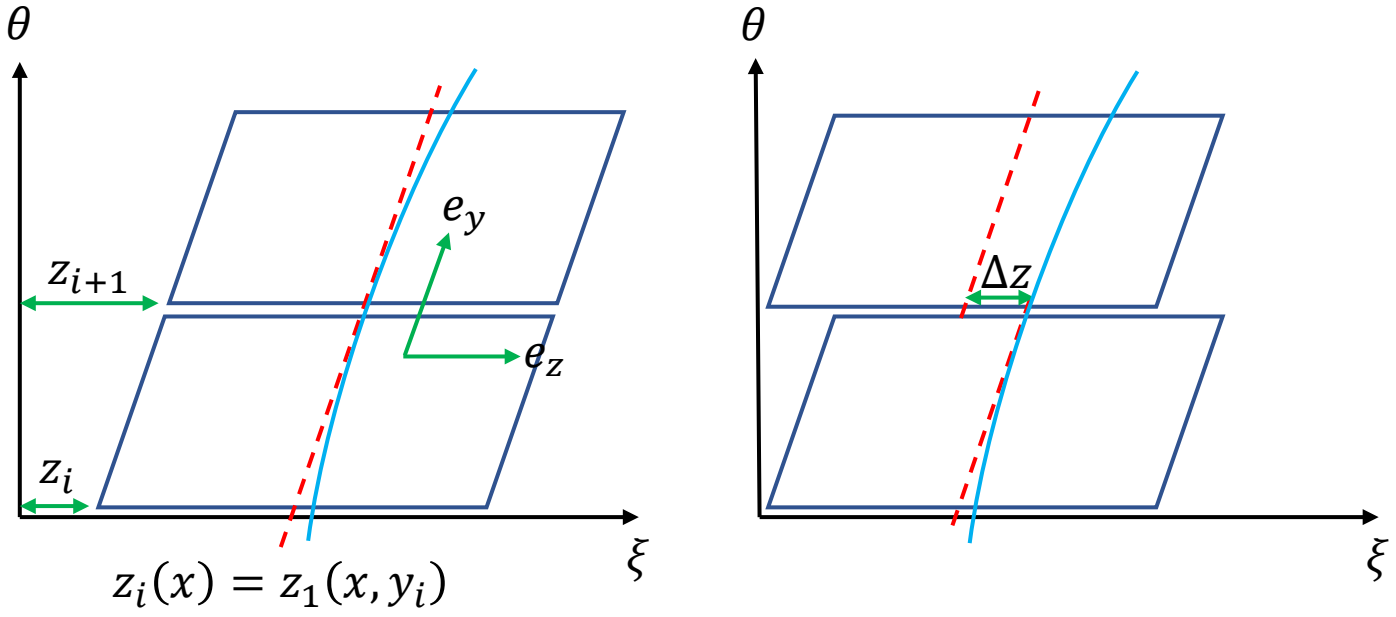
$$\partial_{x'} = \partial_\psi + I_s' \partial_\xi = \partial_\psi$$

The grid points in simple shifted metric coordinates coincide with normal flux coordinate (ψ, θ, ϕ).

Avoid interpolation in hybrid code.

Shifted metric method

Simple shifted metric implement



$z_i(x) = z_1(x, y_i)$

$$\alpha_i'(x) = \left.\frac{g^{xz}}{g^{xx}}\right|_{y=y_i}$$

$$\alpha_i'(x) = -I(x, y_i)$$

Find a function $z_1(x, y)$

$$z_1'(x, y_i) = \alpha_i'(x)$$

For B. Scott shifted metric

$$z_1(x, y) = \int_{x_0}^{x} \frac{g^{xz}(x', y)}{g^{xx}(x', y)} dx'$$

For simple shifted metric

$$z_1(x, y) = z_0(x, y)$$

Numerical difference in y direction need interpolation.

$$\partial_y f = \sum_p C_p f^i(x, y + \mathrm{p}\,\Delta y, z)$$

$$f^i(x, y + \mathrm{p}\,\Delta y, z) = f^{i+p}(x, y + \mathrm{p}\,\Delta y, z + \Delta z)$$

$$\Delta z = z_{i+p} - z_i \approx \nu(\psi, \theta_i) p \Delta y$$

Coordinates transform

Coordinates transform from equilibrium coordinates

$$(R, Z, \phi) \rightarrow (\psi, \theta_V, \xi_V)/(\rho, \theta_D, \xi_D) \rightarrow (\psi, \theta_B, \xi_B) \rightarrow (x, y, z) \rightarrow (x', y', z')$$

Cylindrical VMEC DESC Boozer Field-aligned Shifted metric

$$M = \frac{\partial(R, Z, \phi)}{\partial(\psi, \theta, \xi)} = \begin{bmatrix} \partial_\psi R & \partial_\theta R & 0 \\ \partial_\psi Z & \partial_\theta Z & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad g^{\psi\theta\phi} = M^{-1} g^{RZ\phi} (M^{-1})^T$$

$$M_2 = \frac{\partial(x, y, z)}{\partial(\psi, \theta, \xi)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -I & -\nu & 1 \end{bmatrix} \qquad g^{xyz} = M_2 g^{\psi\theta\phi} (M_2)^T$$

$$I_s'(\psi, \theta) = \int_{\theta_i}^{\theta} \nu(\psi, \theta') d\theta'$$

$$z_0' = 0, \qquad I_s' = 0, \qquad \partial_x I_s' = 0, \qquad \partial_y I_s' = \nu$$

$$\partial_\psi g^{ij}, \partial_\theta g^{ij}, \partial_\psi g_{ij}, \partial_\theta g_{ij}$$

Initial disturb function

$$f(x'', y'', z'') = f(\psi, \theta, \phi) = f_0 \cos(m\theta - n\phi)$$

$$m - nq(\psi_c) = 0$$

Use **Mathematica** to generate transform code for **MATLAB**.
Support field-aligned, B. Scott's shifted metric and simple shifted metric coordinates.

Equilibrium is calculated by VMEC or DESC [D. W. Dudt. 2022 POP]

Outline

1. Background
2. Compile-time Symbolic Solver CSS
3. Optimization for fluid and particle simulations
4. Shifted metric method
5. Gyrokinetic MHD hybrid code GMEC
6. Field and particle code FP3D
7. Conclusion

P. Y. Jiang, Z.Y. Liu. et. al. *GMEC: Gyrokinetic MHD energetic particle simulation code* . Prepare for submission

MHD part of gyrokinetic-MHD hybrid model (GMEC)

Vorticity equation $$\frac{\partial}{\partial t}\delta\bar{\omega} + \vec{v}_{*i}\cdot\nabla\delta\bar{\omega} = \delta\vec{B}\cdot\nabla\left(\frac{\mu_0 J_\parallel}{B}\right) + \vec{B}\cdot\nabla\left(\frac{\mu_0\delta J_\parallel}{B}\right) + \frac{2\mu_0}{B}\vec{b}\times\vec{\kappa}\cdot\nabla_\perp(\delta P + \delta P_h)$$

Ohm's law $$\frac{\partial}{\partial t}\delta A_\parallel = -\partial_\parallel\delta\phi - \eta_\parallel\delta J_\parallel + \frac{P_e}{en_e^2 B}\delta\vec{B}\cdot\nabla n_e + \frac{P_e}{en_e^2}\vec{b}\cdot\nabla\delta n_e$$

Pressure equation $$\frac{\partial}{\partial t}\delta P = -\vec{v}_{E\times B}\cdot\nabla P - \frac{2\Gamma \mathrm{P}}{B}\vec{b}\times\vec{\kappa}\cdot\nabla\delta\Phi$$

Electron density equation $$\frac{\partial}{\partial t}\delta n_e = -\vec{v}_{E\times B}\cdot\nabla n_e$$

where

$$\partial_\parallel = \vec{b}\cdot\nabla \qquad \delta\vec{B} = \nabla\times\left(\delta A_\parallel\vec{b}\right) \qquad P = P_i + P_e$$

$$\delta\bar{\omega} = \nabla\cdot\frac{1}{v_A^2}\nabla_\perp\delta\Phi \qquad \delta J_\parallel = -\nabla_\perp^2\delta A_\parallel \qquad \vec{v}_{E\times B} = \frac{1}{B}\vec{b}\times\nabla\delta\Phi \qquad \vec{v}_{*i} = \frac{1}{enB}\vec{b}\times\nabla \mathrm{P_i}$$

Using field-aligned coordinates with the shifted metric method

Equilibrium is calculated by VMEC or DESC [D. W. Dudt. 2022 POP]

Symbolic implement of GMEC

x, y, z	`enum class  cname = {x,y,z};`
$m, P, g_{ij}, g^{ij}, J, B, j_{\parallel}, \ldots$	`enum class  name = {m, pres, gcov, gcon, jacobi, Bs, jp, …};`
m	`using m = make_variable_t<name::m>;`
$P[x]$	`using pres = make_variable_t<name::pres, cname::x>;`
$B[x, y]$	`using Bs = make_variable_t<name::Bs, cname::x, cname::y>;`
$\delta\Phi[x, y, z]$	`using dPhi = make_variable_t<name::Bs, cname::x, cname::y, cname::z>;`

m	`auto m = global_const<m>;`
$P[x]$	`auto pres = make_local_grid_t<pres, type::deriv>;`
$B[x, y]$	`using Bs = make_local_grid_t<Bs, type::deriv2>;`
$\delta\Phi[x, y, z]$	`using dPhi = make_mpi_grid_t<dPhi, mpi_coordinate, type::scalar>;`
$[x, y, z]$ [0,2,2]	`using Shift_coor = make_physics_coordinate_t<Shift, List<boundary::Dirichlet, boundary::periodic, boundary::periodic>, List<0,2,2>>;`

Symbolic implement of GMEC

$$\nabla_{\parallel} = \vec{b} \cdot \nabla$$

```
auto Nabla_p = b*Nabla
```

$$\delta\bar{\omega} = \nabla \cdot \frac{1}{v_A^2} \nabla_{\perp} \delta\Phi$$

```
auto dw = Div*(_va2*Nabla*dPhi)
```

$$\delta J_{\parallel} = -\frac{1}{B} \nabla \cdot \left(B^2 \nabla_{\perp} \left(\frac{\delta A_{\parallel}}{B} \right) \right)$$

```
auto dJp = - Div*(Bs2*Nabla*(dA/Bs))/Bs
```

$$\delta\vec{B} = \nabla \times \left(\delta A_{\parallel} \vec{b} \right)$$

```
auto dB = Cross(Nabla,dA*b)
```

$$\vec{v}_{*i} = \frac{1}{enB} \vec{b} \times \nabla \mathrm{P_i}$$

```
auto v_star = Cross(b,Nabla*Pi)
/(e*n*Bs)
```

$$\vec{v}_{E\times B} = \frac{1}{B} \vec{b} \times \nabla\delta\Phi$$

```
auto v_EB = Cross(b,Nabla*Phi)/Bs
```

$$\vec{v}_d = \frac{1}{B} \vec{b} \times \vec{\kappa}$$

```
auto v_d = Cross(b,kappa)/Bs
```

$$\frac{\partial}{\partial t} \delta\bar{\omega} = -\vec{v}_{*i} \cdot \nabla\delta\bar{\omega} \quad + \delta\vec{B} \cdot \nabla \left(\frac{\mathrm{J}_{\parallel}}{B} \right) \quad + \vec{B} \cdot \nabla \left(\frac{\delta J_{\parallel}}{B} \right) \quad + 2\vec{v}_d \cdot \nabla_{\perp} \delta P$$

```
constexpr auto dw_dt = -v_star*Nabla*dw + dB*Nabla*(Jp/Bs) + Bs*b*Nabla*(dJ/Bs) + Ratio<2>{}*v_d*Nabla*dP
```

$$\frac{\partial}{\partial t} \delta A_{\parallel} = -\nabla_{\parallel} \delta\Phi \quad - \eta \delta J_{\parallel} \quad + \frac{P_e}{e n_e^2 B} \delta\vec{B} \cdot \nabla n_e \quad + \frac{P_e}{e n_e^2} \vec{b} \cdot \nabla \delta n_e$$

```
constexpr auto dA_dt = -Nabla_p*dPhi - eta*dJp + Pe/(e*ne2*Bs)*dB*Nabla*ne + Pe/(e*ne2)*b*Nabla*dne
```

$$\frac{\partial}{\partial t} \delta P = -\vec{v}_{E\times B} \cdot \nabla (P_i + P_e) - 2\Gamma (P_i + P_e) \vec{v}_d \cdot \nabla\delta\Phi$$

```
constexpr auto dP_dt = -v_EB*Nabla*(Pi+Pe) -
Ratio<2>{}*gamma*(Pi+Pe)*v_d*Nabla*dPhi
```

$$\frac{\partial}{\partial t} \delta n_e = -\vec{v}_{E\times B} \cdot \nabla n_e$$

```
constexpr auto dne_dt = -v_EB*Nabla*ne
```

Time integration for GMEC

GMEC MHD Equation

$$\frac{\partial}{\partial t}\delta\bar{\omega} + \vec{v}_{*i}\cdot\nabla\delta\bar{\omega} = \delta\vec{B}\cdot\nabla\left(\frac{\mu_0 J_\parallel}{B}\right) + \vec{B}\cdot\nabla\left(\frac{\mu_0\delta J_\parallel}{B}\right) + \frac{2\mu_0}{B}\vec{b}\times\vec{\kappa}\cdot\nabla_\perp(\delta P + \delta P_h)$$

$$\frac{\partial}{\partial t}\delta A_\parallel = -\partial_\parallel\delta\phi - \eta_\parallel\delta J_\parallel + \frac{P_e}{en_e^2 B}\delta\vec{B}\cdot\nabla n_e + \frac{P_e}{en_e^2}\vec{b}\cdot\nabla\delta n_e$$

$$\frac{\partial}{\partial t}\delta P = -\vec{v}_{E\times B}\cdot\nabla P - \frac{2\Gamma P}{B}\vec{b}\times\vec{\kappa}\cdot\nabla\delta\Phi$$

$$\frac{\partial}{\partial t}\delta n_e = -\vec{v}_{E\times B}\cdot\nabla n_e$$

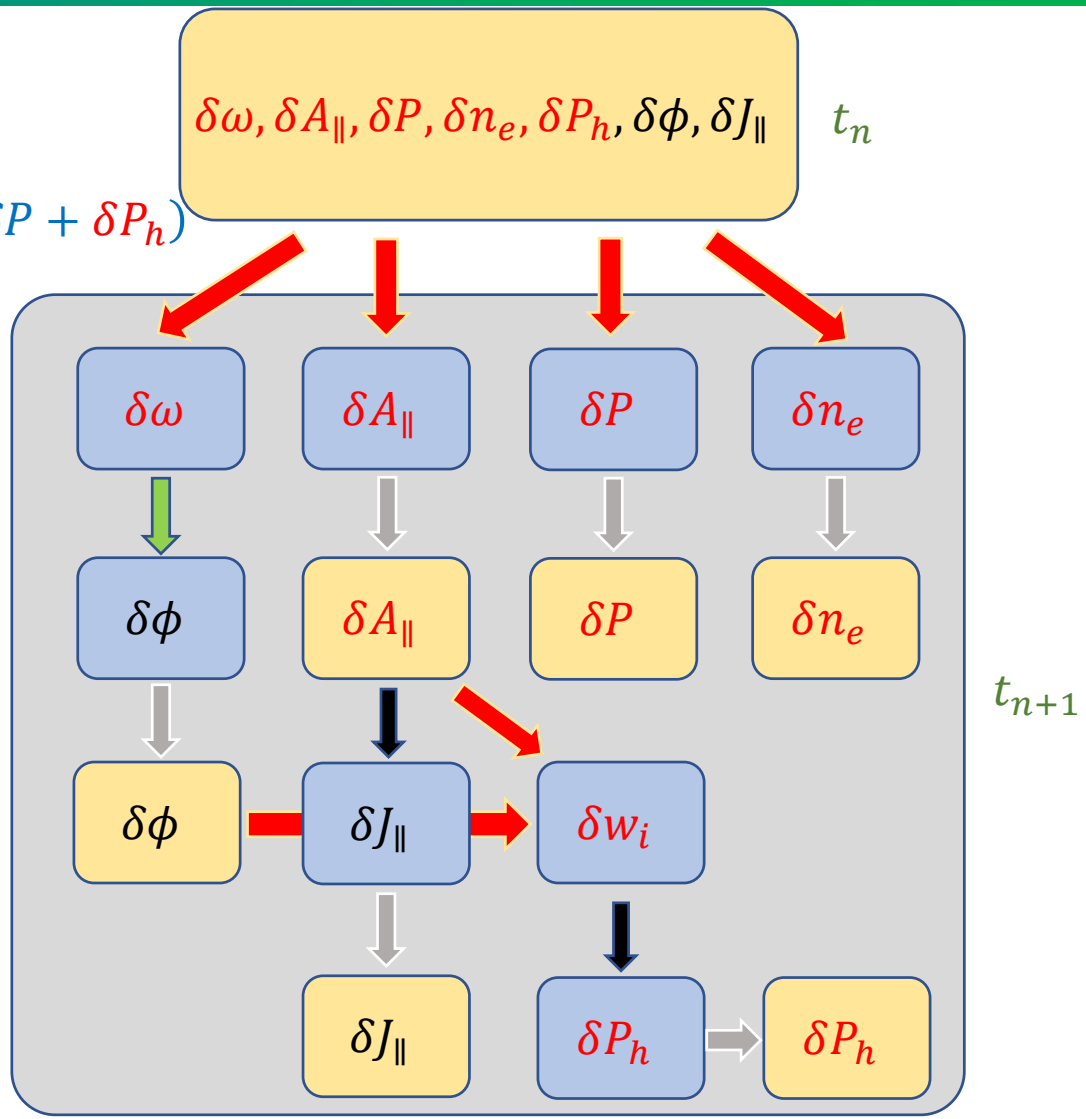
$$\delta\bar{\omega} = \nabla\cdot\frac{1}{v_A^2}\nabla_\perp\delta\phi \qquad \delta J_\parallel = -\frac{1}{B}\nabla\cdot\left(B^2\nabla_\perp\left(\frac{\delta A_\parallel}{B}\right)\right)$$

grid without ghost

grid with ghost

time advance

ghost grid update



Flow chart

Gyrokinetic-MHD hybrid model (GMEC)

PIC method of GMEC

$$f = f_0 + \delta f$$
$$w \equiv \frac{\delta f}{g}$$
$$\frac{dw_i}{dt} = -\left[\frac{f_i(t=0)}{g_i(t=0)} - w_i\right]\frac{1}{f_0}\left(\frac{dP_\phi}{dt}\frac{\partial f_0}{\partial P_\phi} + \frac{dE}{dt}\frac{\partial f_0}{\partial E}\right)$$

Where

$$\frac{dE}{dt} = \frac{d\boldsymbol{X}}{dt}\cdot\mu\nabla B + \frac{dv_\parallel}{dt}mv_\parallel$$

$$\frac{dP_\phi}{dt} = \frac{d\boldsymbol{X}}{dt}\cdot\nabla P_\phi + \frac{dv_\parallel}{dt}\frac{\partial P_\phi}{\partial v_\parallel}$$

$$E = \frac{1}{2}mv_\parallel^2, \qquad P_\phi = qg\rho_\parallel - q\psi_p$$

Gyro-center $\boldsymbol{X}, v_\parallel$

Gyro-kinetic equations

$$\frac{d\boldsymbol{X}}{dt} = \frac{1}{B^{**}}\left\{v_\parallel\boldsymbol{B}^* - \boldsymbol{b}\times\left[\langle\boldsymbol{E}\rangle - \frac{\mu}{e}\nabla(B + \langle\delta B\rangle)\right]\right\}$$
$$\frac{dv_\parallel}{dt} = \frac{e}{mB^{**}}\boldsymbol{B}^*\cdot\left[\langle\boldsymbol{E}\rangle - \frac{\mu}{e}\nabla(B + \langle\delta B\rangle)\right]$$

where

$$\boldsymbol{B}^* = \boldsymbol{B} + \langle\delta\boldsymbol{B}\rangle + \frac{mv_\parallel}{e}\nabla\times\boldsymbol{b}, \qquad B^{**} = \boldsymbol{B}^*\cdot\boldsymbol{b}$$

$$\delta\boldsymbol{B} = \nabla\times(\delta A_\parallel\boldsymbol{b}), \qquad \boldsymbol{E} = -\nabla\delta\phi - \frac{\partial\delta A_\parallel}{\partial t}\boldsymbol{b}$$

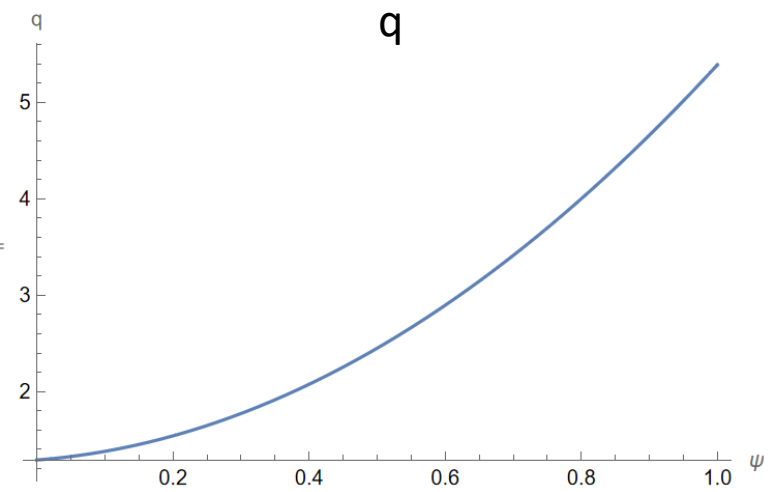
4 points average

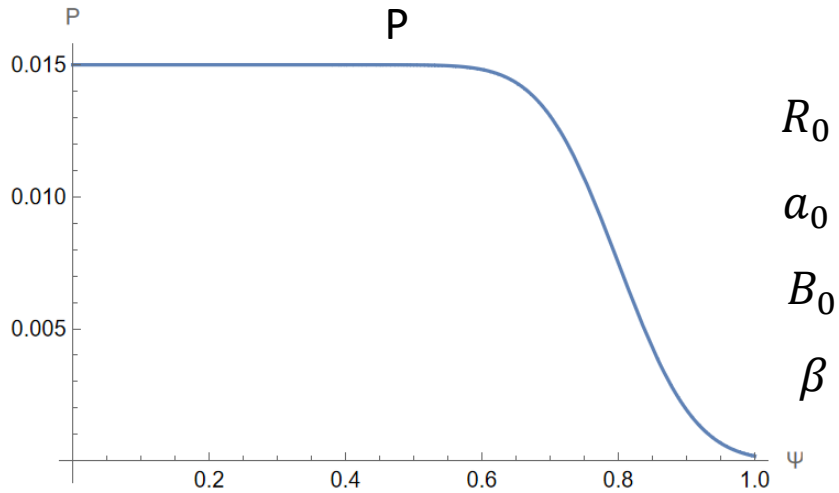
Pressure coupling

$$\delta P_\parallel = \iiint m\,v_\parallel^2\delta f d^3v = \frac{1}{N_p}\sum_i^{N_p} mv_{\parallel,i}^2 w_i\frac{1}{J}\delta(x-x_i)\delta(y-y_i)\delta(z-z_i)$$

$$\delta P_\perp = \iiint \frac{1}{2}mv_\perp^2\delta f d^3v = \frac{1}{N_p}\sum_i^{N_p}\frac{1}{2}mv_{\perp,i}^2 w_i\frac{1}{J}\delta(x-x_i)\delta(y-y_i)\delta(z-z_i)$$

Ideal ballooning mode in CFETR





China Fusion Engineering Test Reactor

$R_0 = 7.2m$

$a_0 = 2.2m$

$B_0 = 6.5T$

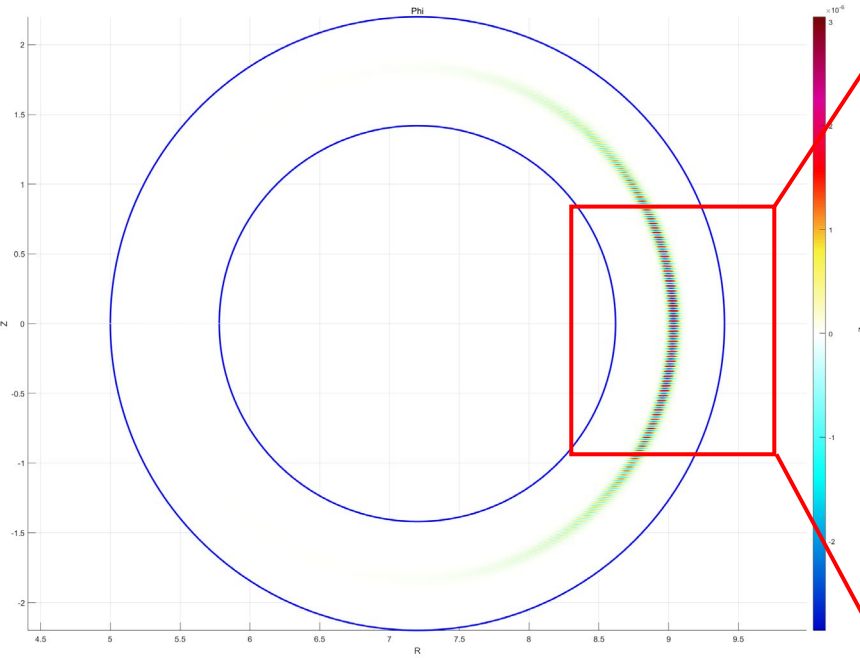
$\beta = 0.03$

$N_x = 1024, N_y = 256, N_z = 16$

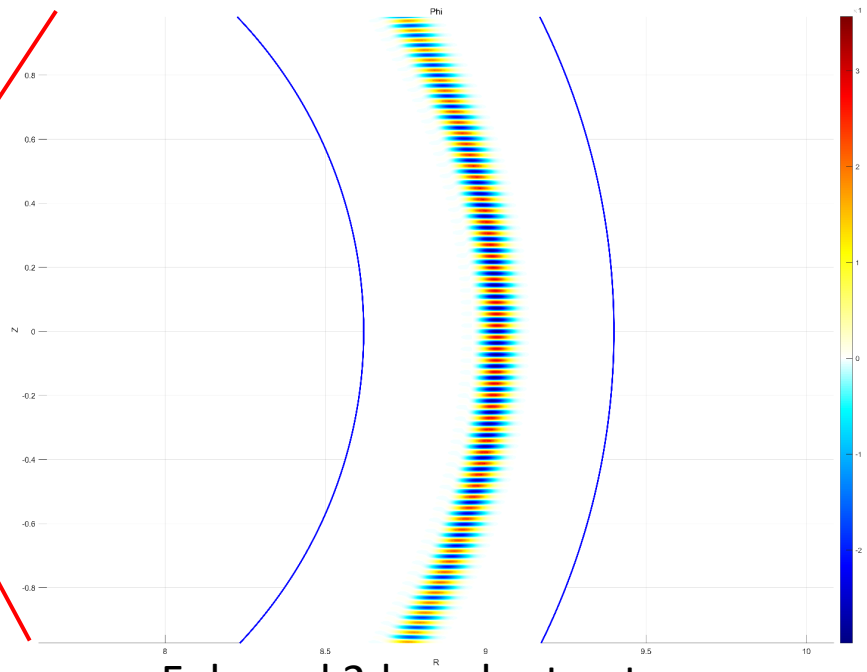
$n = 100$

$dt = 0.02a_0/v_A, N = 30000$

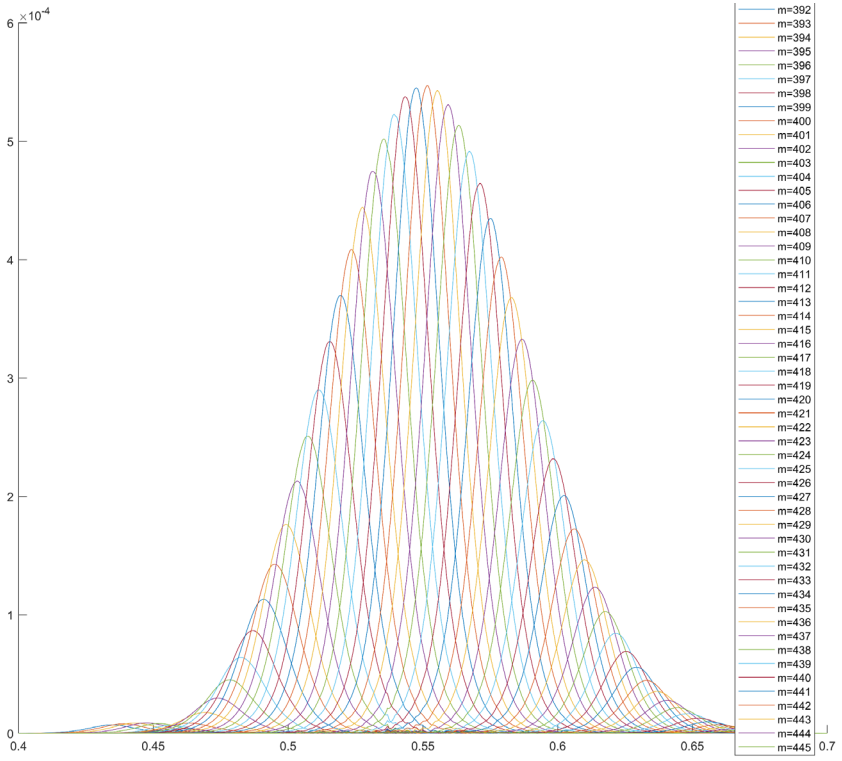
448 cores, 2.2 hours



2d mode structure

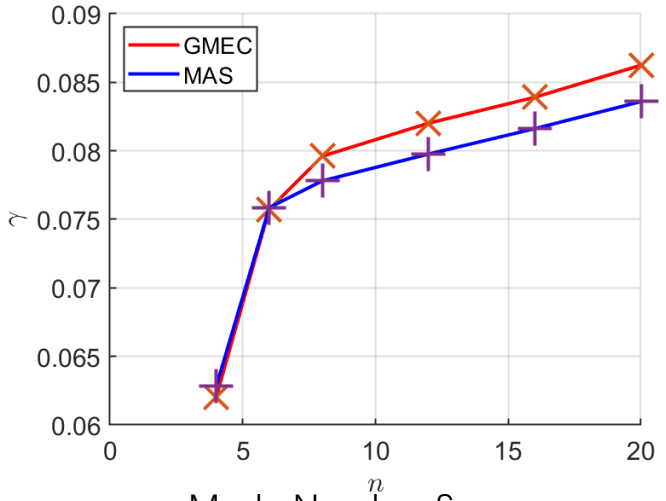


Enlarged 2d mode structure

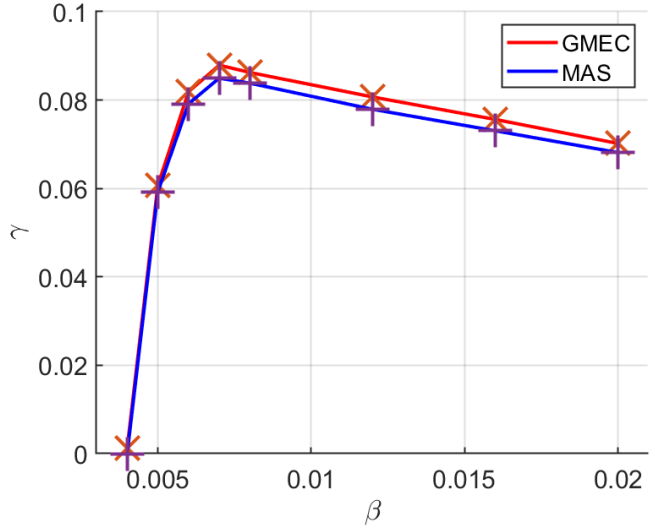


Radial mode structure

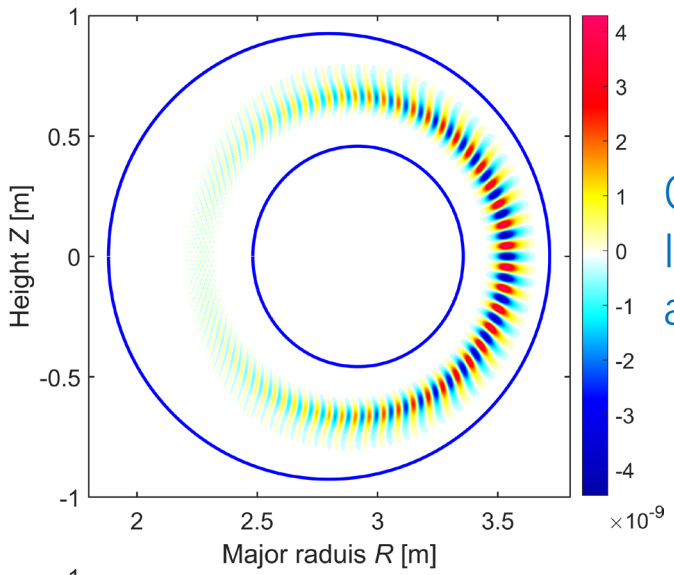
Benchmark with MAS: ideal ballooning mode



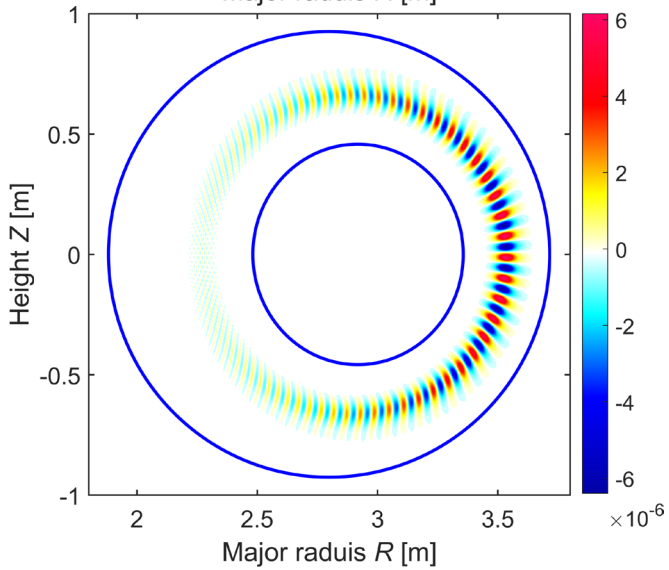
Mode Number Scan



β Scan

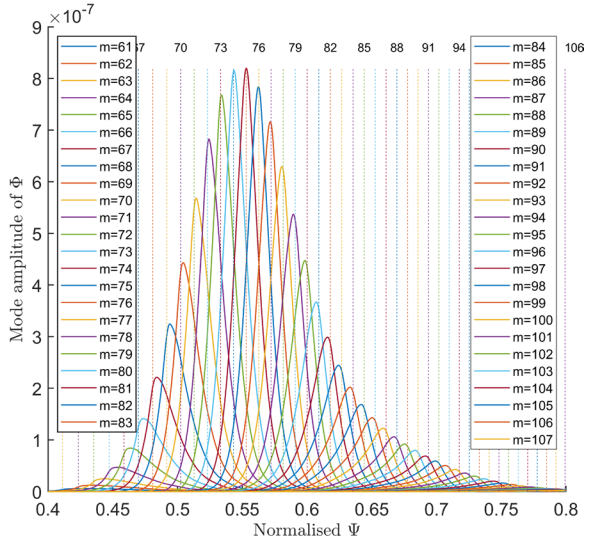


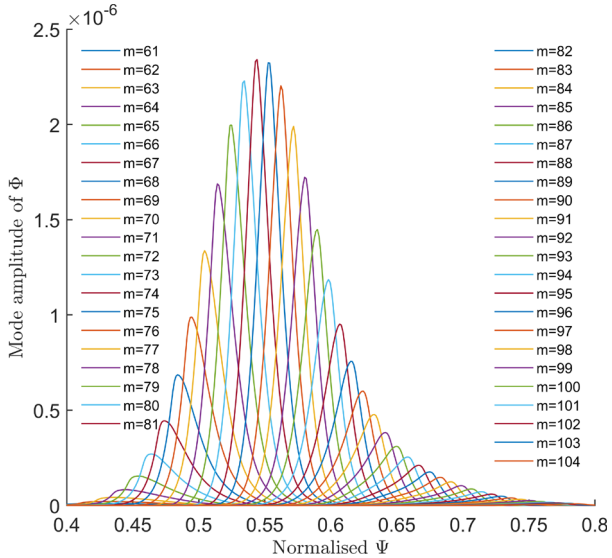
GMEC
Initial value
approach



MAS
Eigenvalue
approach

J. Bao
2022 NF



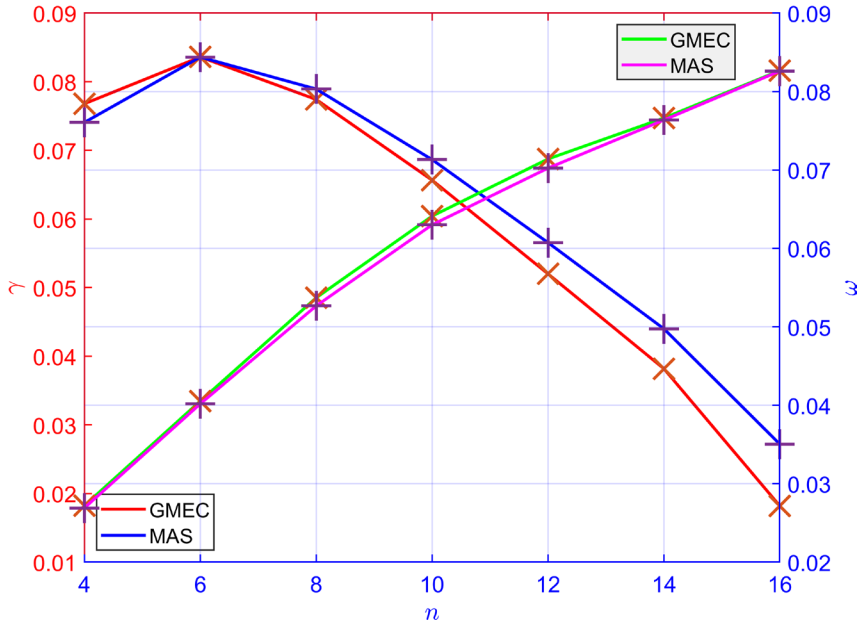


No $\eta_{\parallel}$, no numerical diffusion term

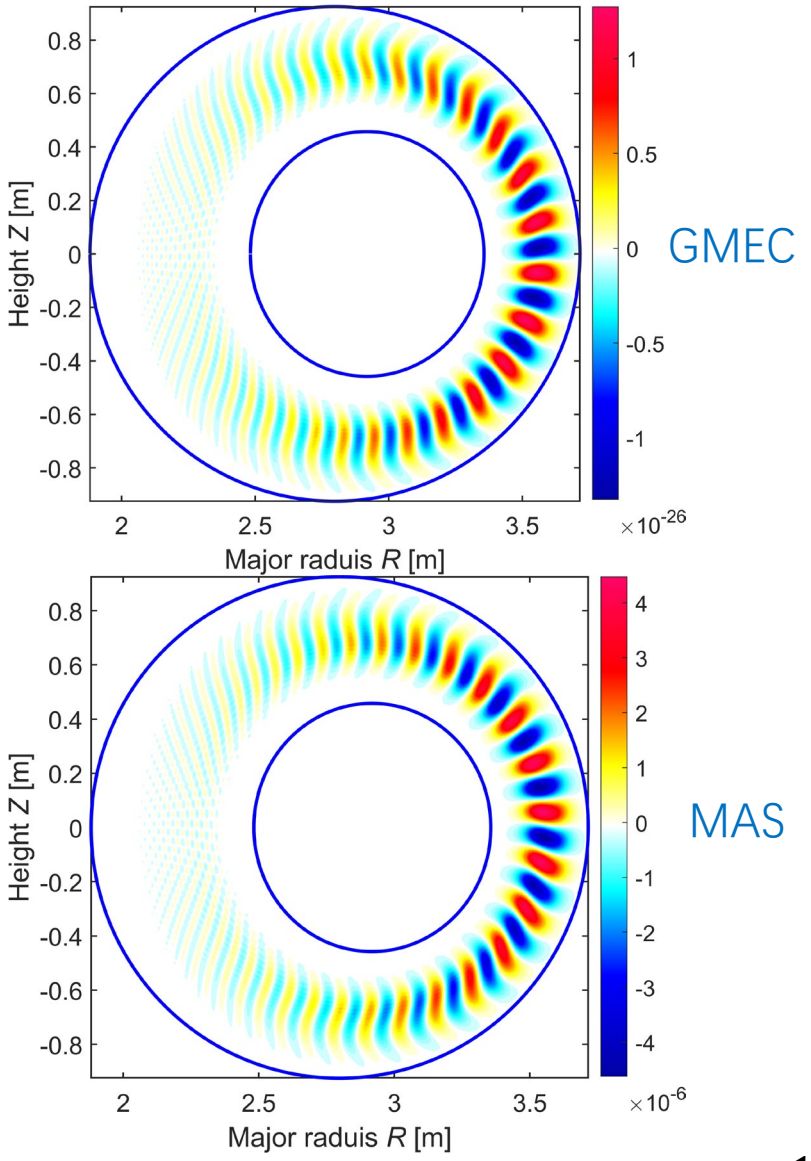
Relative difference with MAS is less than 4%

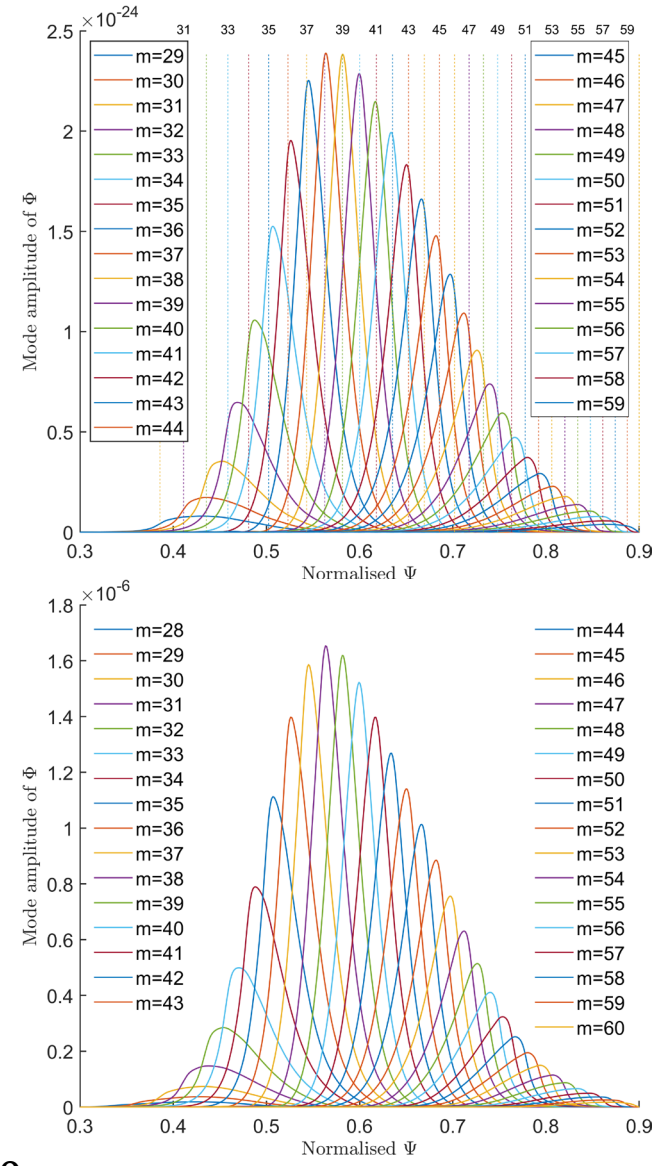
$n = 20$ Ideal ballooning mode, GMEC costs 15 seconds using 448 cores

Benchmark with MAS: diamagnetic drift effects



Relative differences for ω are less than 1% and those for γ are about 10%

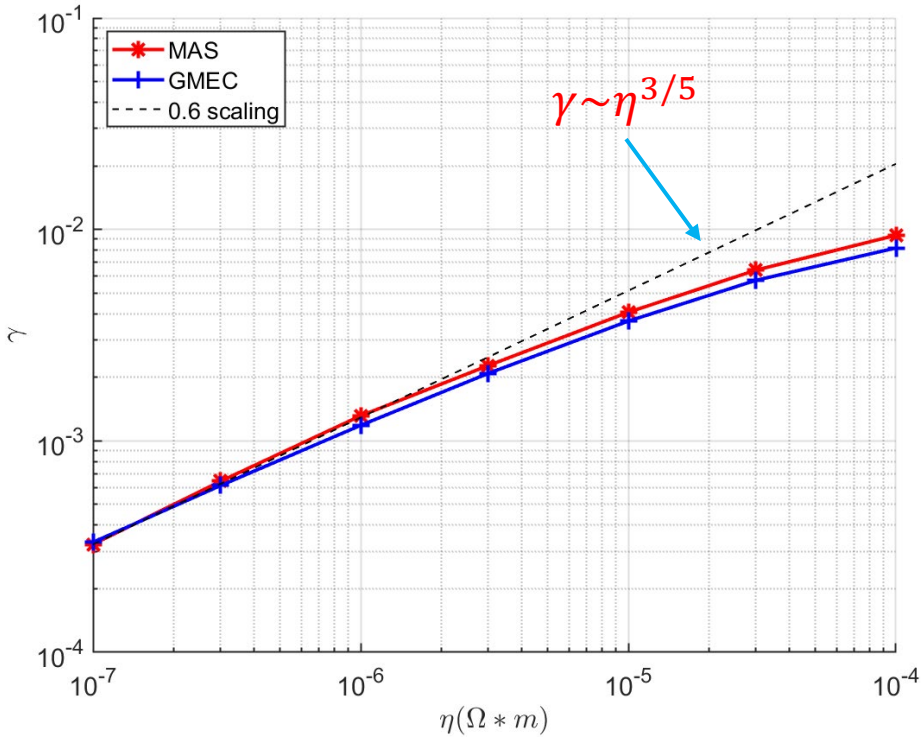




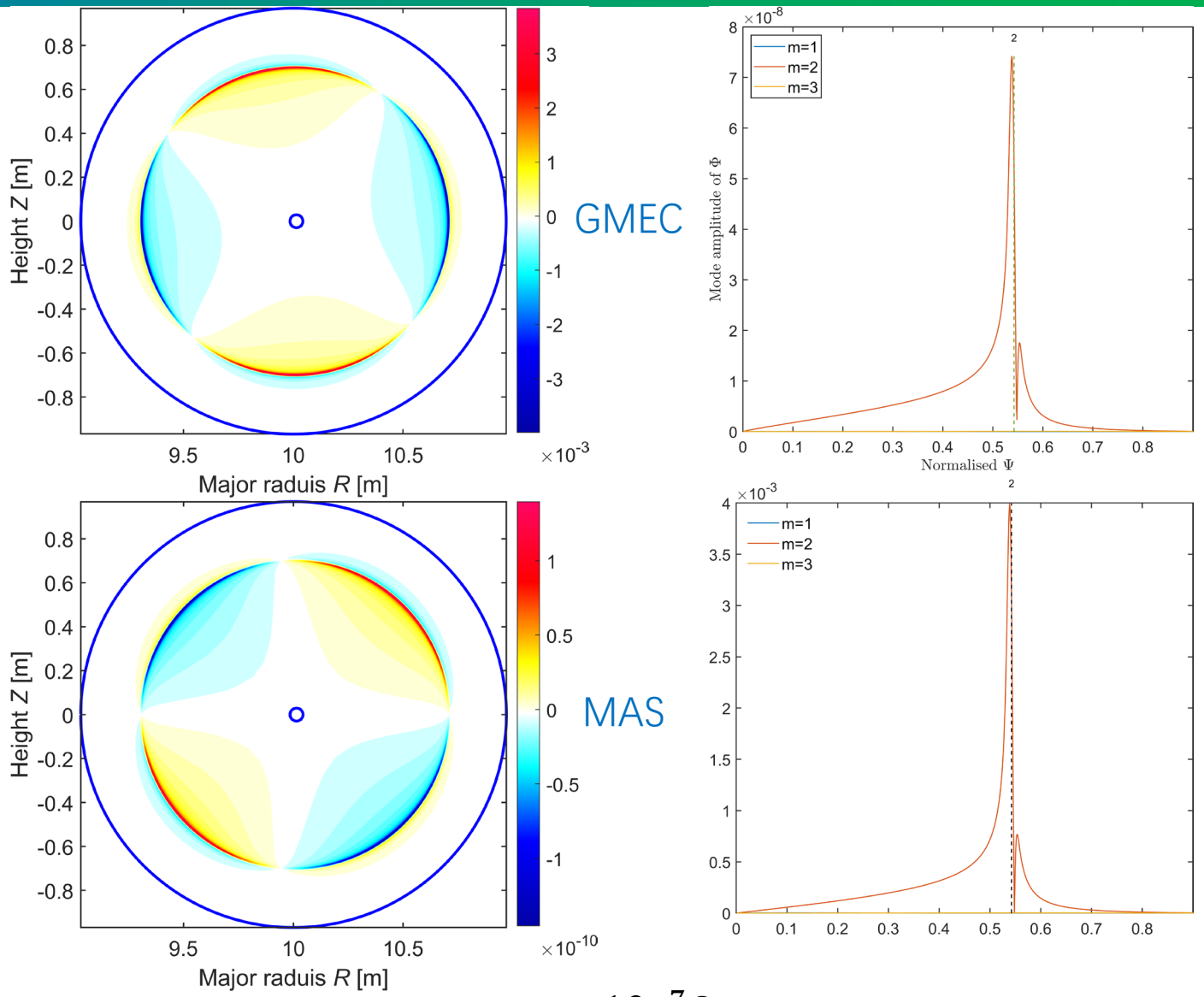
$n = 10$

Benchmark with MAS: tearing mode

$m/n = 2/1$ tearing mode



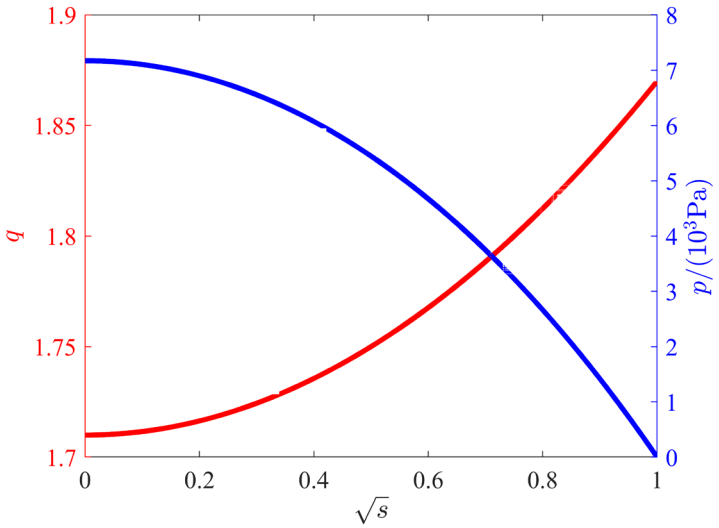
Relative differences with MAS are less than 15%

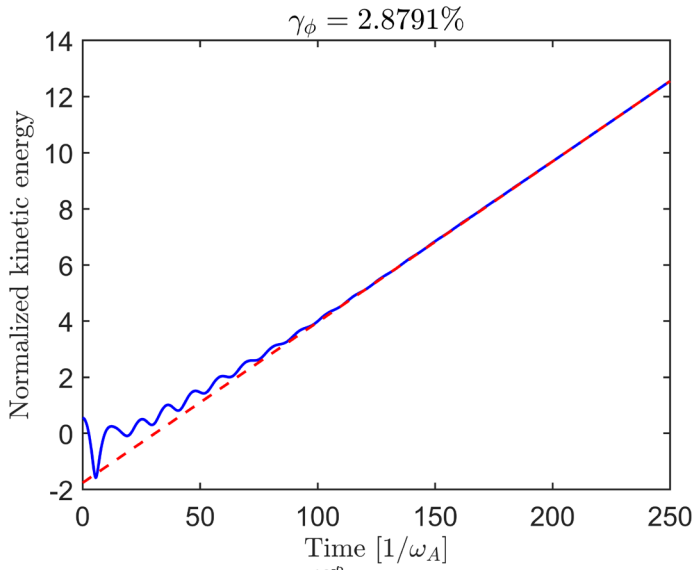


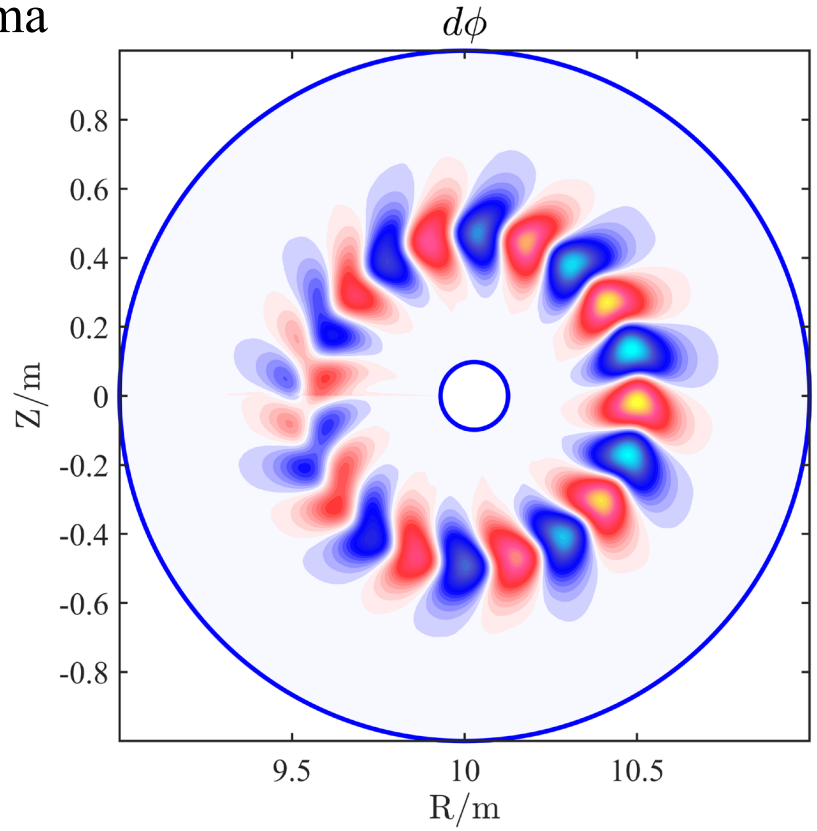
$\eta = 10^{-7}\Omega \cdot m$

Toroidal Alfven Eigenmode (TAE)

A TAE mode with m = 10, 11 and n = 6 is calculated in a hydrogen plasma







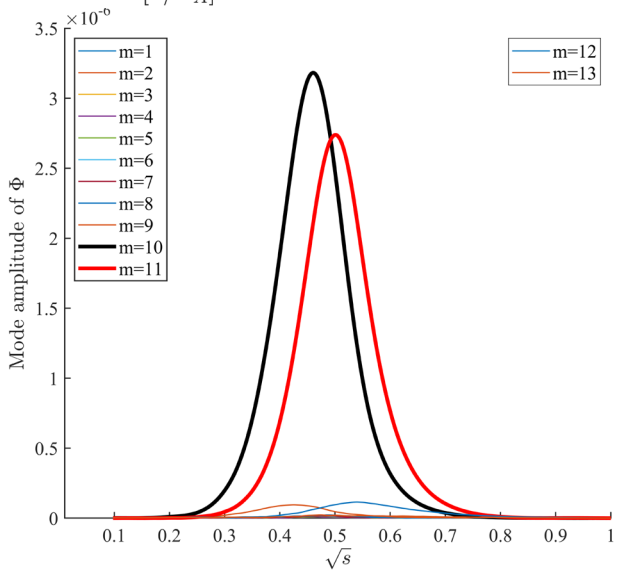
$$n(s) = n_0 c_3 \exp\left(-\frac{c_2}{c_1}\tanh\frac{\sqrt{s}-c_0}{c_2}\right)$$

$$n_0 = 1.44131 \cdot 10^{17}\mathrm{m}^{-3}, c_0 = 0.49123$$

$$c_1 = 0.298, c_2 = 0.199, c_3 = 0.521$$

$$R = 10\ m, a = 1\ m$$

Maxwell distribution



$$(N_x, N_y, N_z) = (256,64,16)$$

$N_{pc} = 64$, $N_{particle} = 16{,}777{,}216$

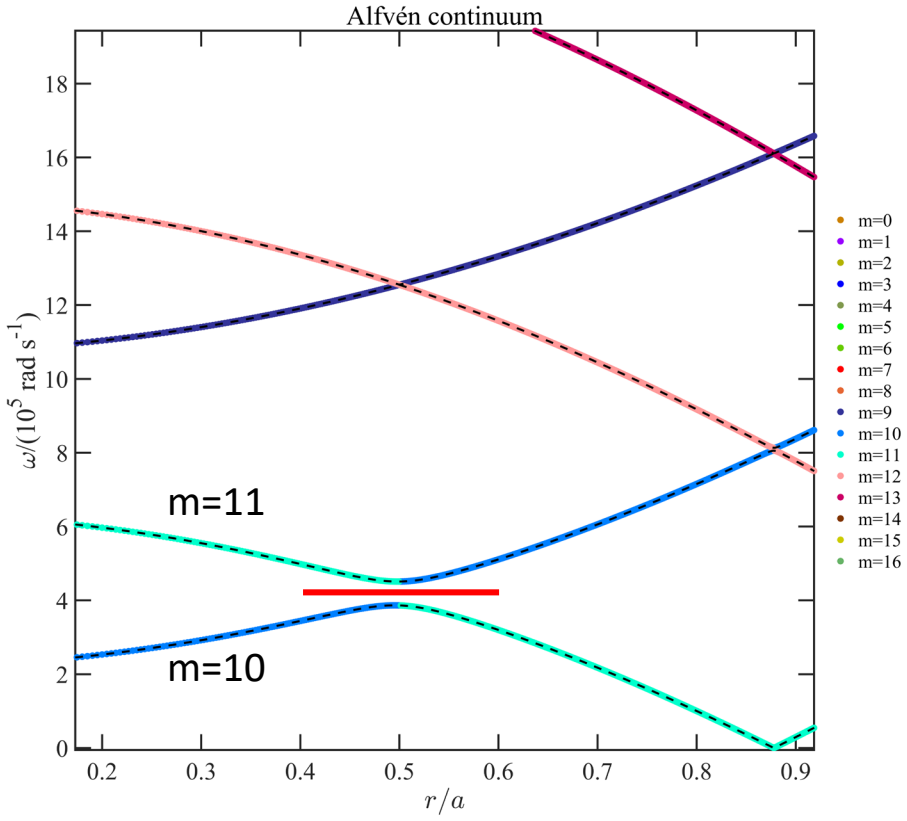
$N_{step} = 25{,}000$, $\Delta t = 0.01 R_0/v_A$, RK4

448 cores, 25 minutes, wo FLR (0.06s)

54 minutes, with FLR

A. Könies *et al* 2018 *Nucl. Fusion* **58** 126027

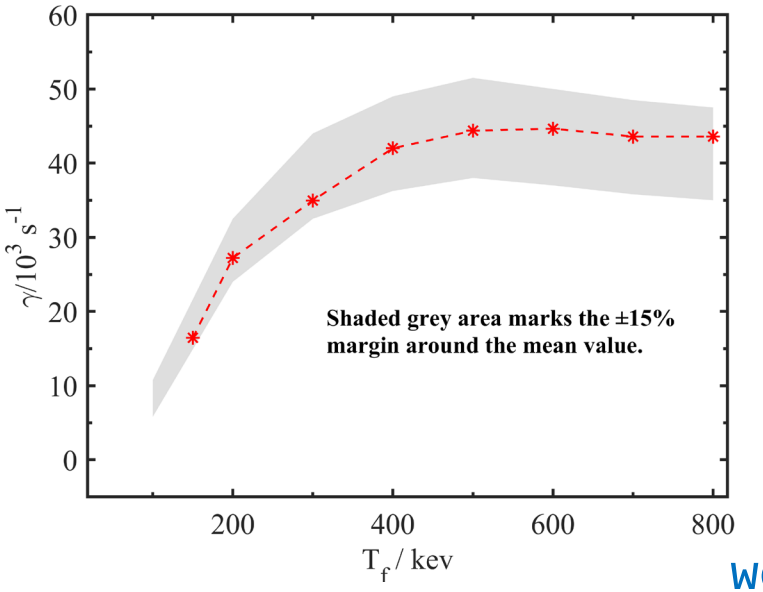
Benchmark: n=6 TAE

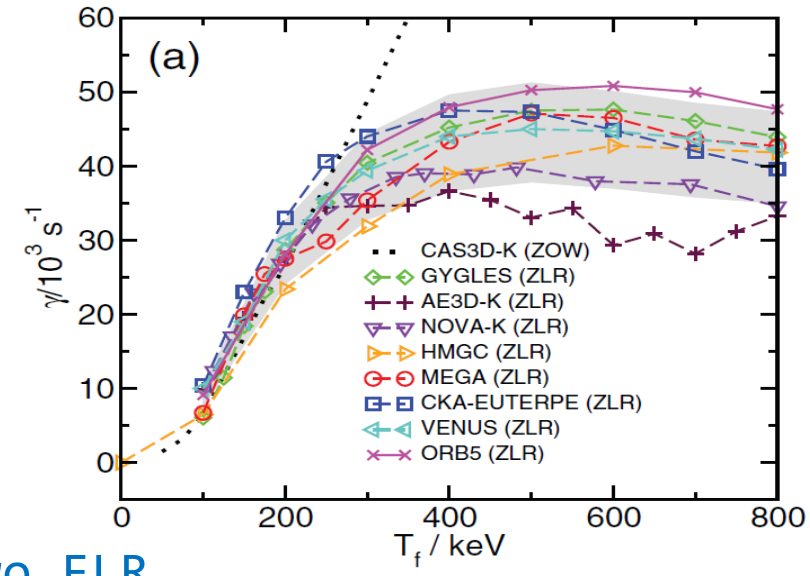


Continuum (MAS)
Gap range: [3.86,4.52]

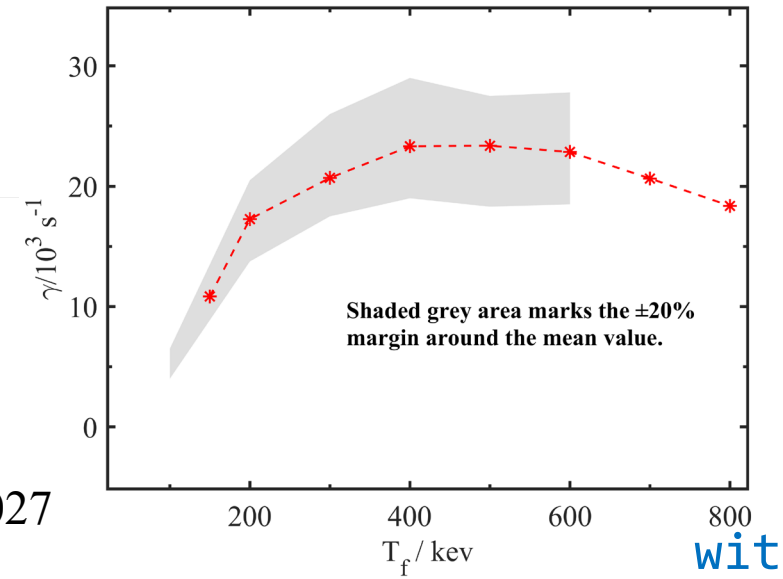
Maxwell distribution

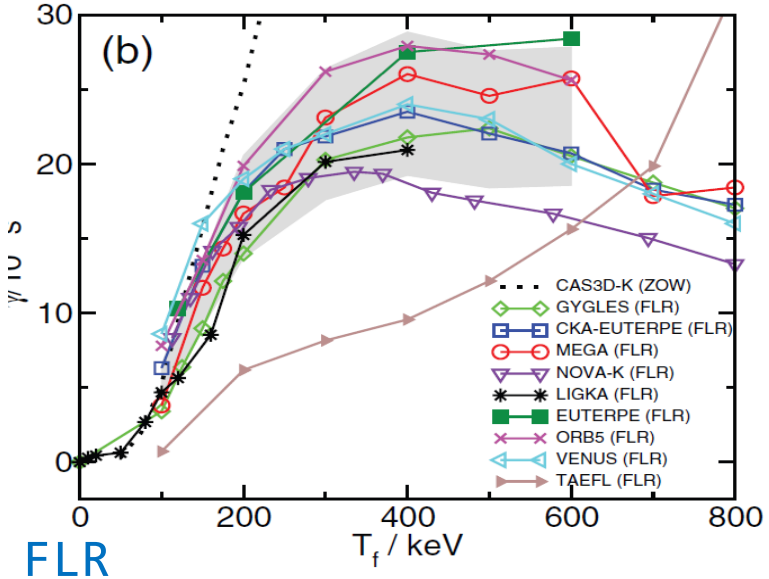
A. Könies *et al* 2018 *Nucl. Fusion* **58** 126027





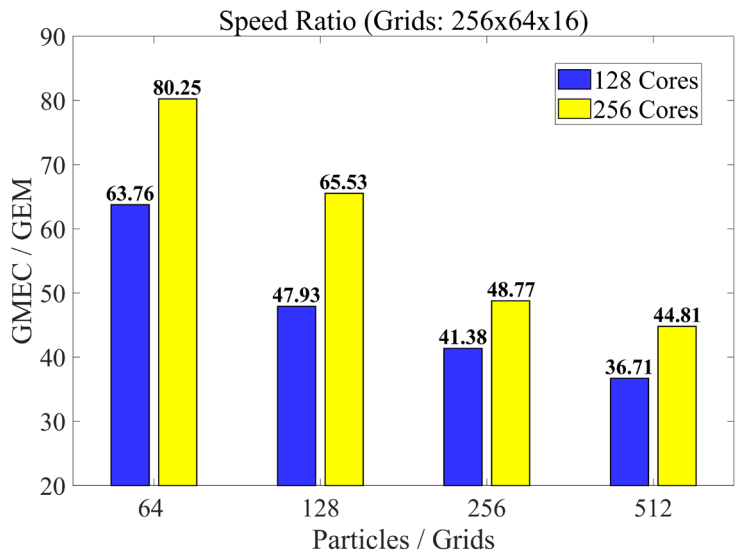
wo FLR

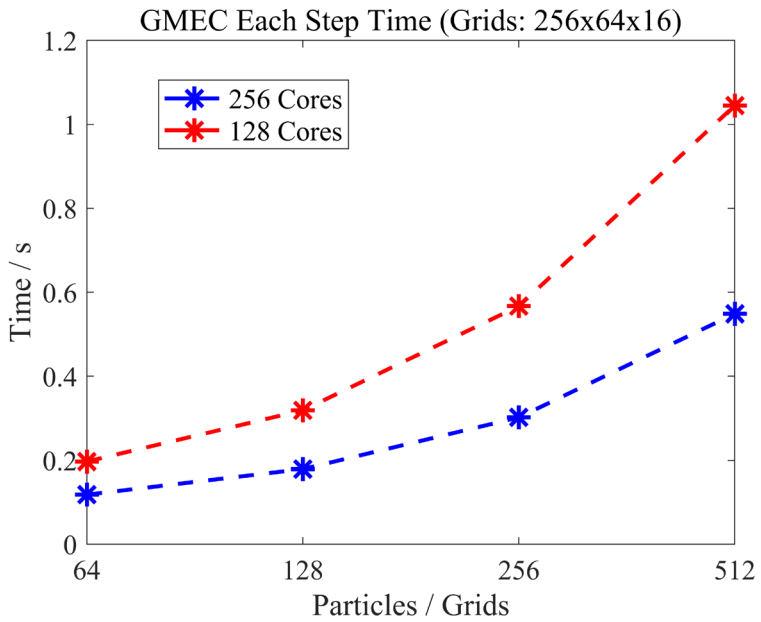




with FLR

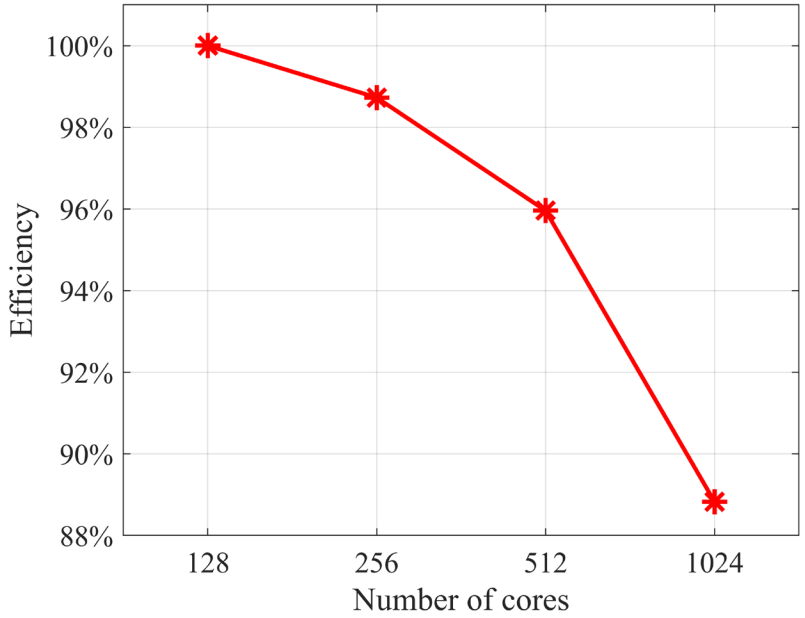
GMEC is much faster than typical codes





Runge-Kutta: 2 order
Gyro-Average: 4 points

$N = 134{,}217{,}728$



Parallel efficiency decrease with the number of cores due to the increase of ghost grid ratio.

Outline

1. Background
2. Compile-time Symbolic Solver CSS
3. Optimization for fluid and particle simulations
4. Shifted metric method
5. Gyrokinetic MHD hybrid code GMEC
6. Field and particle code FP3D
7. Conclusion

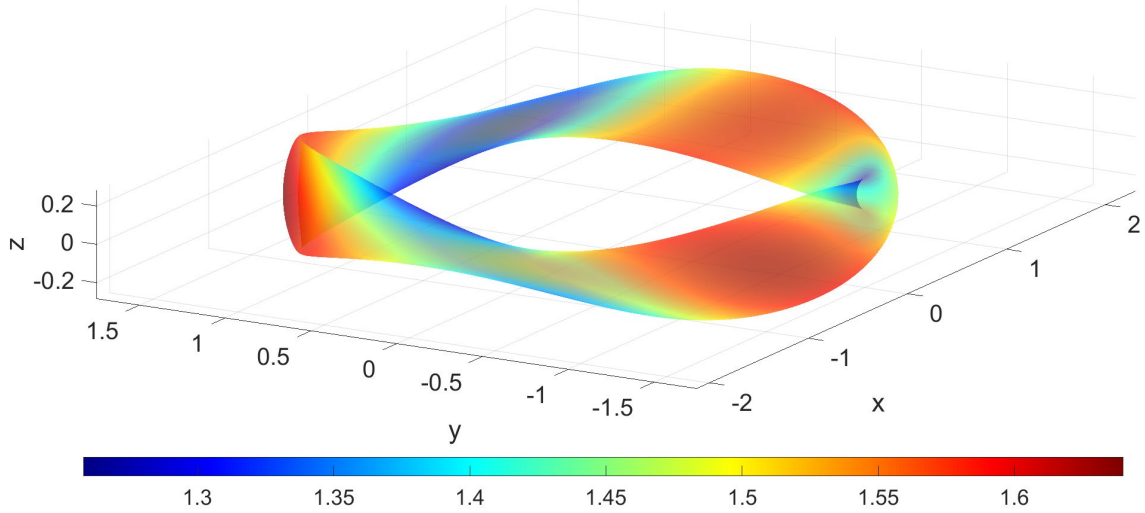
P. Y. Jiang, Z. C. Feng, G. D. Yu, G. Y. Fu. (2023). *FP3D: A code for calculating 3D magnetic field and particle motion.* Arxiv

FP3D: magnetic field and surface

Magnetic field

$$\vec{B}(\vec{r}) = \int \frac{4\pi}{\mu_0} \frac{I d\vec{l} \times \hat{e}_r}{r^2}$$

$$= \sum_i \frac{4\pi}{\mu_0} \frac{I_i \Delta\vec{r} \times \hat{e}_{r,\bar{r}}}{\bar{r}^2}$$



trace field line for NCSX

Trace field line

Equal length

$$\frac{d\vec{r}}{dl} = \hat{b}$$

Equal ϕ

$$\frac{d\vec{r}}{d\phi} = \frac{\vec{B}}{\vec{B} \cdot \nabla\phi}$$

Rotation transform ι

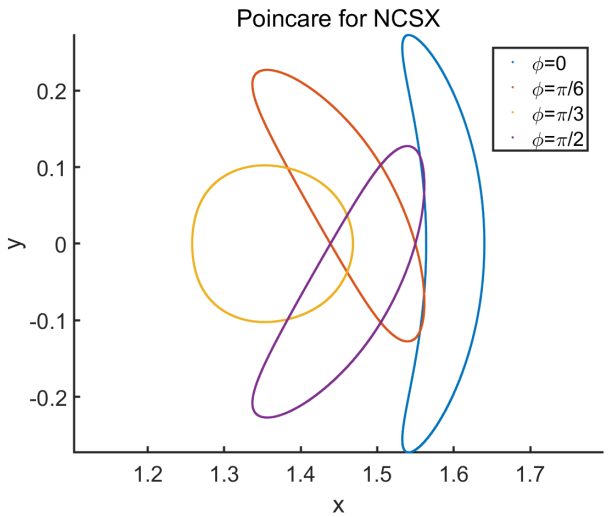
$$\frac{dr}{d\phi} = b_r(r, \phi, z)$$

$$\frac{dz}{d\phi} = b_z(r, \phi, z)$$

$$\frac{dr_a}{d\phi} = b_r(r_a, \phi, z_a)$$

$$\frac{dz_a}{d\phi} = b_z(r_a, \phi, z_a)$$

$$\frac{d\theta}{d\phi} = \begin{pmatrix} (z'(\phi) - z_a'(\phi))(r - r_a) \\ -(r'(\phi) - r_a'(\phi))(z - z_a) \end{pmatrix} / \rho^2$$



FP3D: particle equations implement

Guiding-center equations

$$\frac{\partial}{\partial t}\vec{X} = \frac{1}{B_{\parallel}^{*}}\left(U\vec{B}^{*} - \vec{b}\times\vec{E}^{*}\right)$$

$$\mathrm{m}\dot{\mathrm{U}}_{\parallel} = \frac{\vec{\mathrm{B}}^{\dagger}\cdot e\vec{\mathrm{E}}^{\dagger}}{\hat{b}\cdot\vec{\mathrm{B}}^{\dagger}}$$

where

$$\vec{\mathrm{B}}^{\dagger} = \vec{B} + \left(\frac{\epsilon mc}{e}\right)U_{\parallel}\nabla\times\hat{\mathrm{b}}$$

$$\vec{\mathrm{E}}^{\dagger} = \vec{E} + \frac{\mu\nabla\mathrm{B}}{\mathrm{e}\Gamma} - \frac{\epsilon m}{e}U_{\parallel}\frac{\partial\hat{\mathrm{b}}}{\partial\tau}$$

$$\vec{E}^{*} = \vec{E} - \frac{\mu_0}{q}\nabla B$$

```
auto E_star_cov = E_cov-mu/q*(Nabla*Bs)
```

$$\vec{B}^{*} = \vec{B} + \frac{mU}{q}\nabla\times\vec{b}$$

```
auto B_star_con = B_con+m/q*U*Cross(Nabla*b_cov)
```

$$B_{\parallel}^{*} = \vec{b}\cdot\vec{B}^{*}$$

```
auto B_star_p = B_star_con*bcov;
```

$$\frac{\partial}{\partial t}\vec{X} = \frac{1}{B_{\parallel}^{*}}\left(U\vec{B}^{*} - \vec{b}\times\vec{E}^{*}\right)$$

```
constexpr auto dX_dt =(U* B_star_con-Cross(b_cov, B_star_s ))/ B_star_p
```

$$\frac{\partial}{\partial t}U = \frac{q}{m}\frac{1}{B_{\parallel}^{*}}\vec{B}^{*}\cdot\vec{E}^{*}$$

```
constexpr auto dU_dt =b_cov* B_star_con/ B_star_p
```

Monte-Carlo collision

$$\lambda_{new} = \lambda_{old}(1 - 2\nu_d\Delta t) \pm \sqrt{(1-\lambda_{old}^2)2\nu_d\Delta t}$$

where

$$\lambda = v_{\parallel}/v$$

B-spline interpolation order: 8

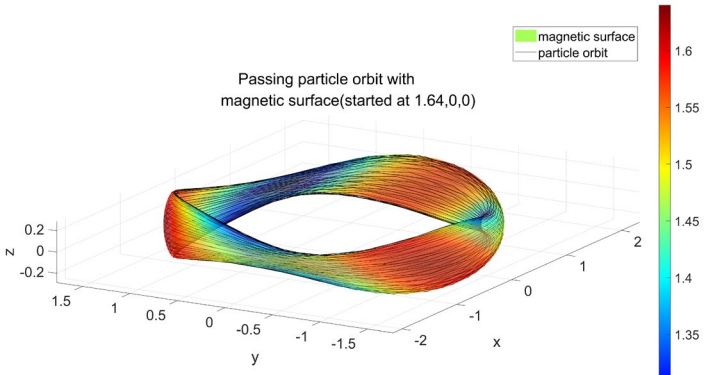
Symbolic grid:

B, B^i, b_i, E_i

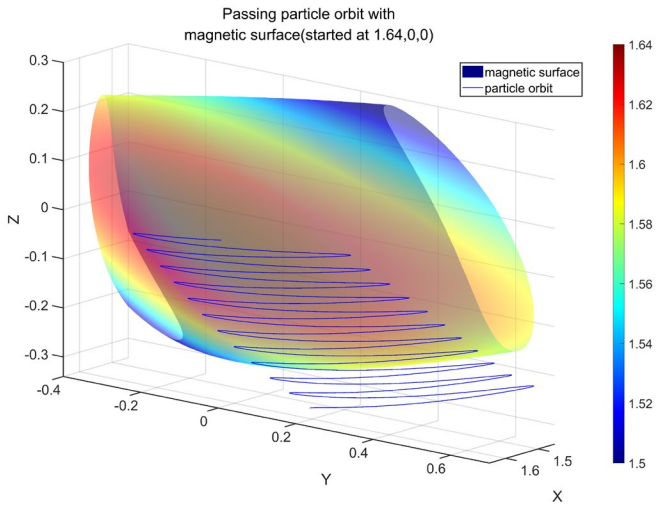
Choose appropriate vector form to avoid metric tensor.

FP3D: particle orbits

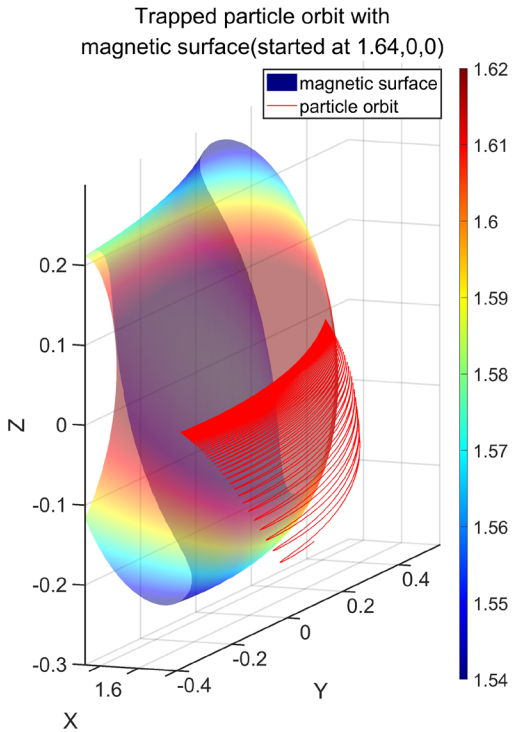
Particle orbits in NCSX stellarator

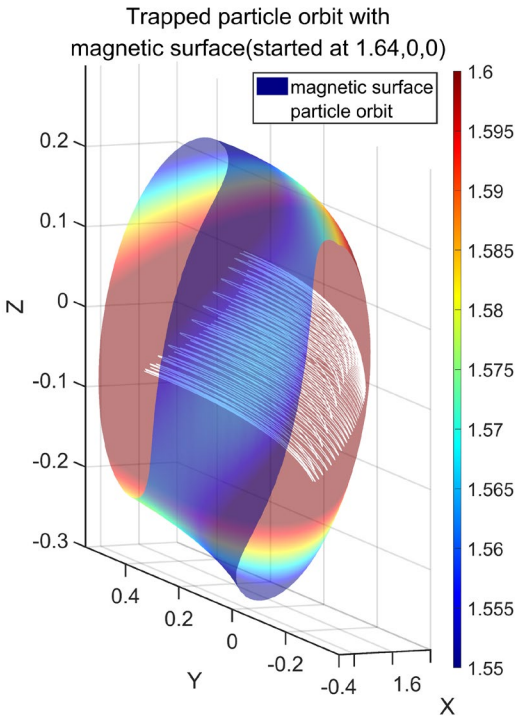


passing particle



trapped particle





Particle with radial electric field

$$\vec{E} = \Phi'(\psi)\nabla\psi$$

FP3D: neoclassical transport

Compare with theory

Collisionless D_l

$$D_l = \frac{3}{8} I_1 \nu \rho^2 \frac{q^2}{\epsilon^2}$$

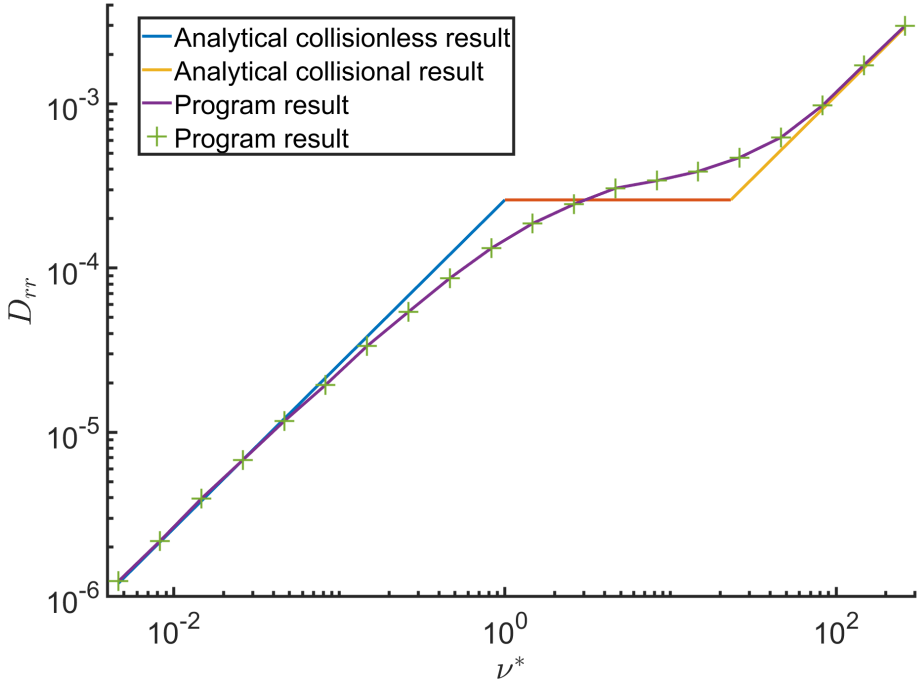
Collisional D_c

$$D_c = \nu q^2 \rho^2$$

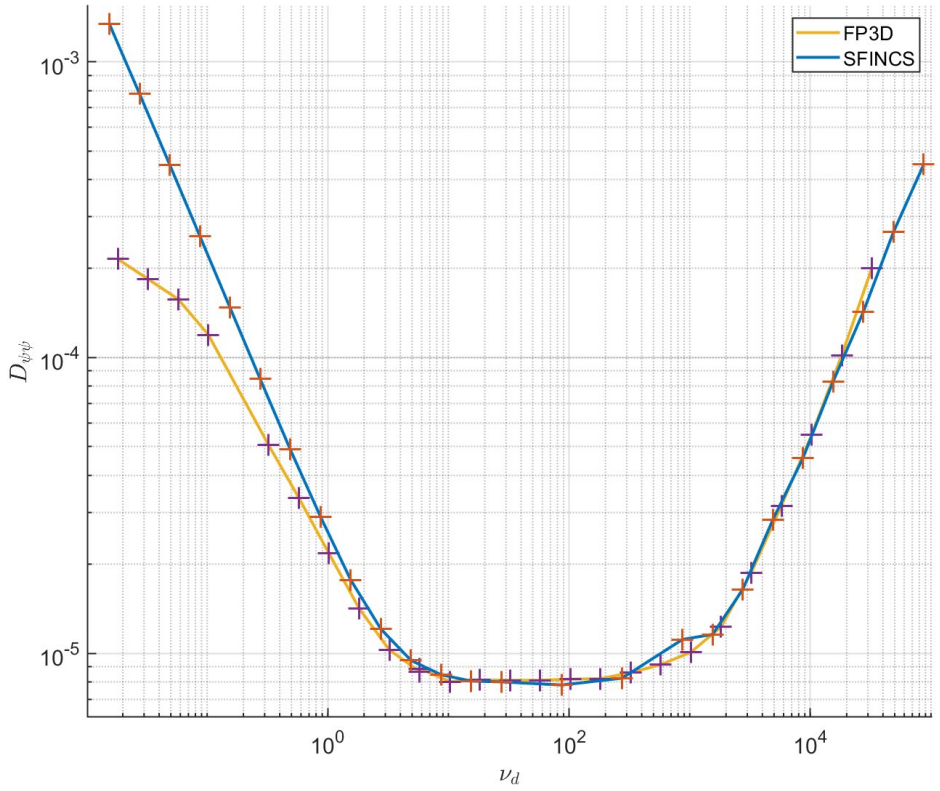
where

$$\rho = m v_{th} / e B_0$$

$$I_1 = 1.38\sqrt{2\epsilon}$$



Transport coefficient in circular Tokamak



Transport coefficient in NCSX

Compare with SFINCS

Calculated by SFINCS

$$\frac{Ze(G+\iota I)}{ncTG}\left\langle \int d^3v f \vec{v}_m \cdot \nabla\psi \right\rangle = L_{11}\frac{GTc}{ZeB_0 v_i}\left[\frac{1}{n}\frac{dn}{d\psi} + \frac{Ze}{T}\frac{d\Phi}{d\psi} - \frac{3}{2T}\frac{dT}{d\psi}\right]$$

$$D_{\psi\psi} = \frac{G^2T^2c^2}{Z^2(G+\iota I)B_0 v_i} L_{11}$$

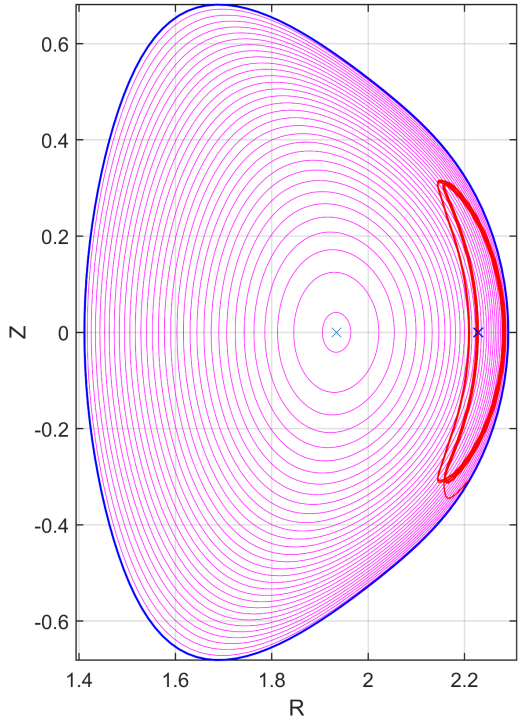
M. Landreman, H. M. Smith, A. Mollén, P. Helander; *Comparison of particle trajectories and collision operators for collisional transport in nonaxisymmetric.* plasmas. Phys. Plasmas 2014.

FP3D: ripple losses in EAST tokamak

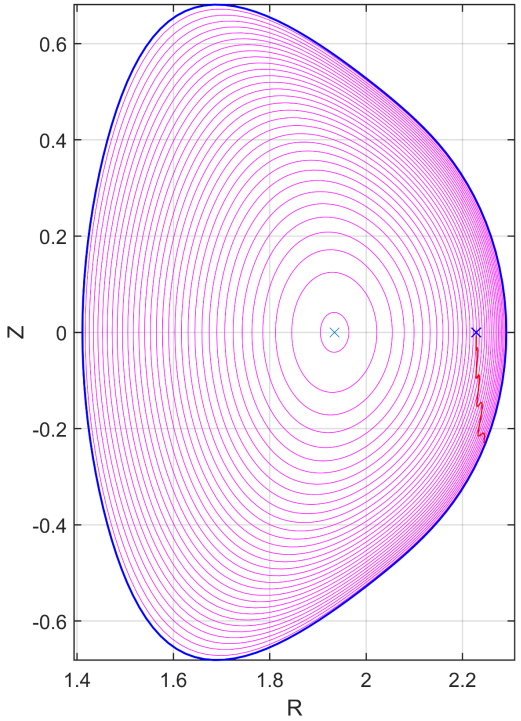
Particle orbit in EAST tokamak with ripple magnetic field

$$\delta B_{\phi}^{rip} = -B_0 R_0 \delta(R,Z)\cos(N\phi)$$

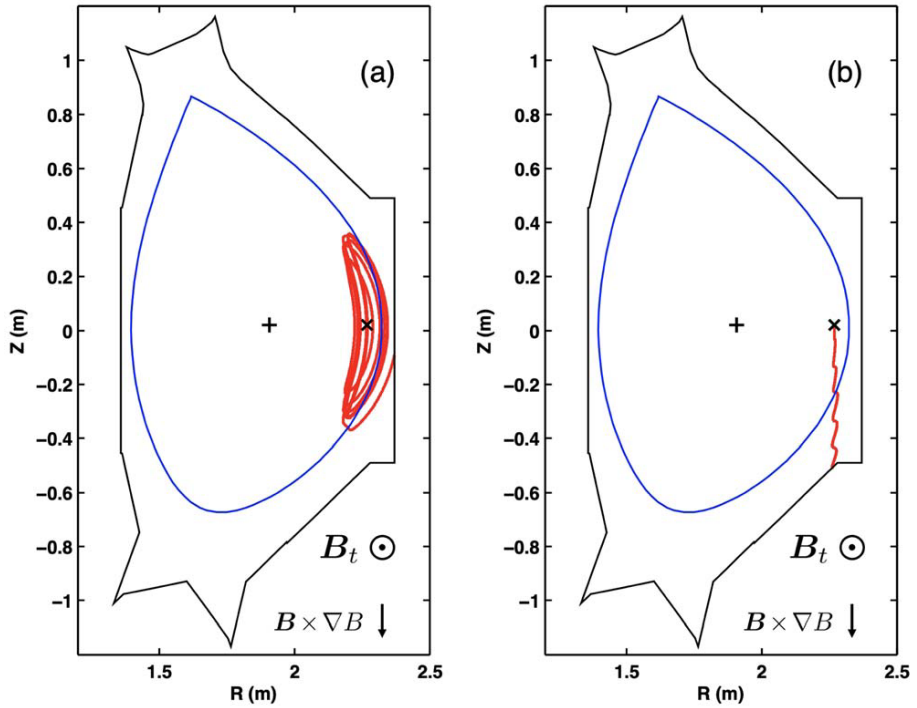
$$\delta(R,Z) = \delta_0 Exp\left[\frac{1}{w_{rip}}\sqrt{\left(R-R_{rip}\right)^2 + b_{rip}Z^2}\right]$$



Shallow trapped particle, chaos loss



Deep trapped particle, direct loss



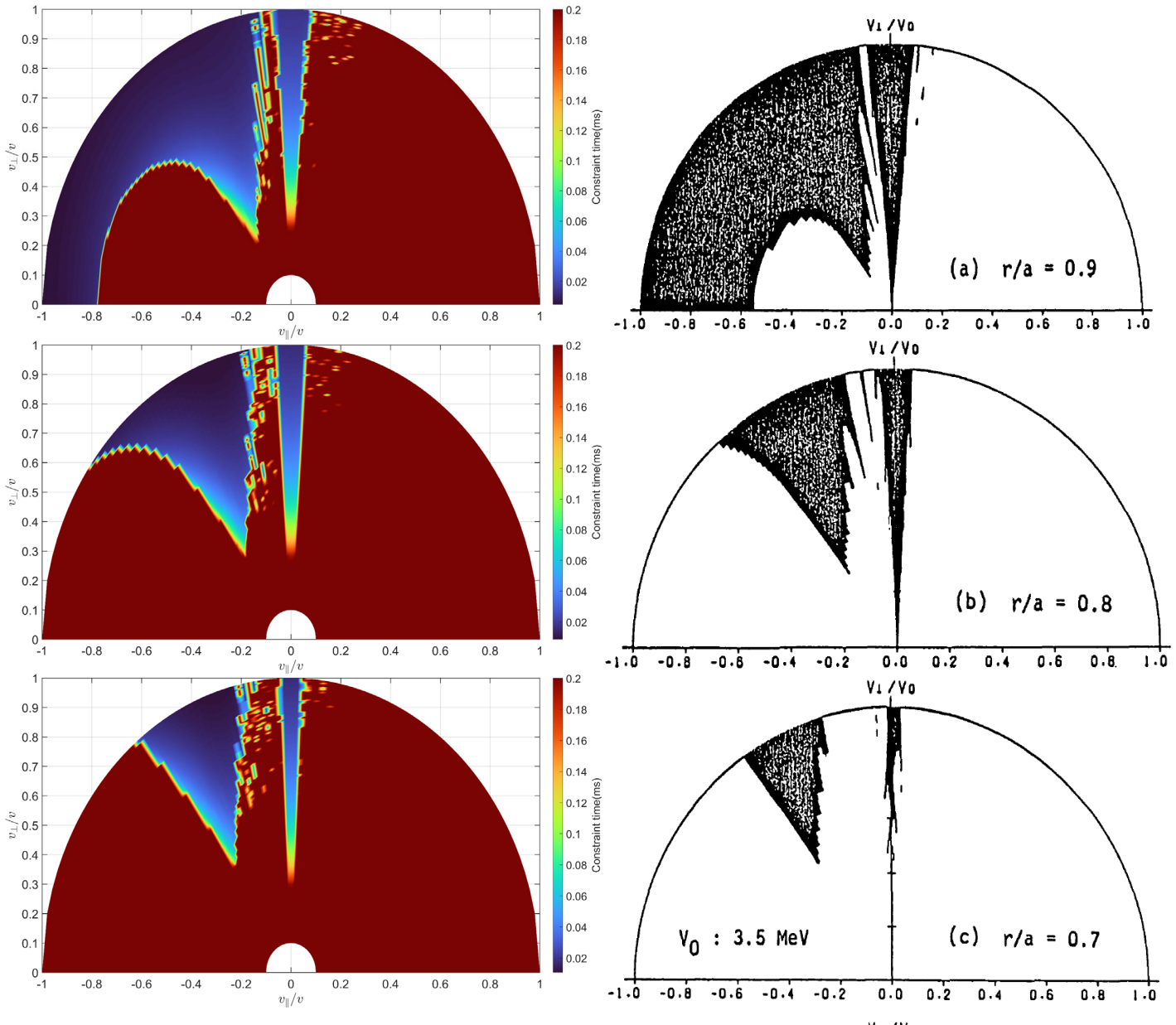
Yingfeng Xu, et al (2021). *Simulations of NBI fast ion loss in the presence of toroidal field ripple on EAST.* Plasma Science and Technology

Ripple losses in EAST tokamak

$\bar{\psi} = 0.5$

$\bar{\psi} = 0.4$

$\bar{\psi} = 0.3$



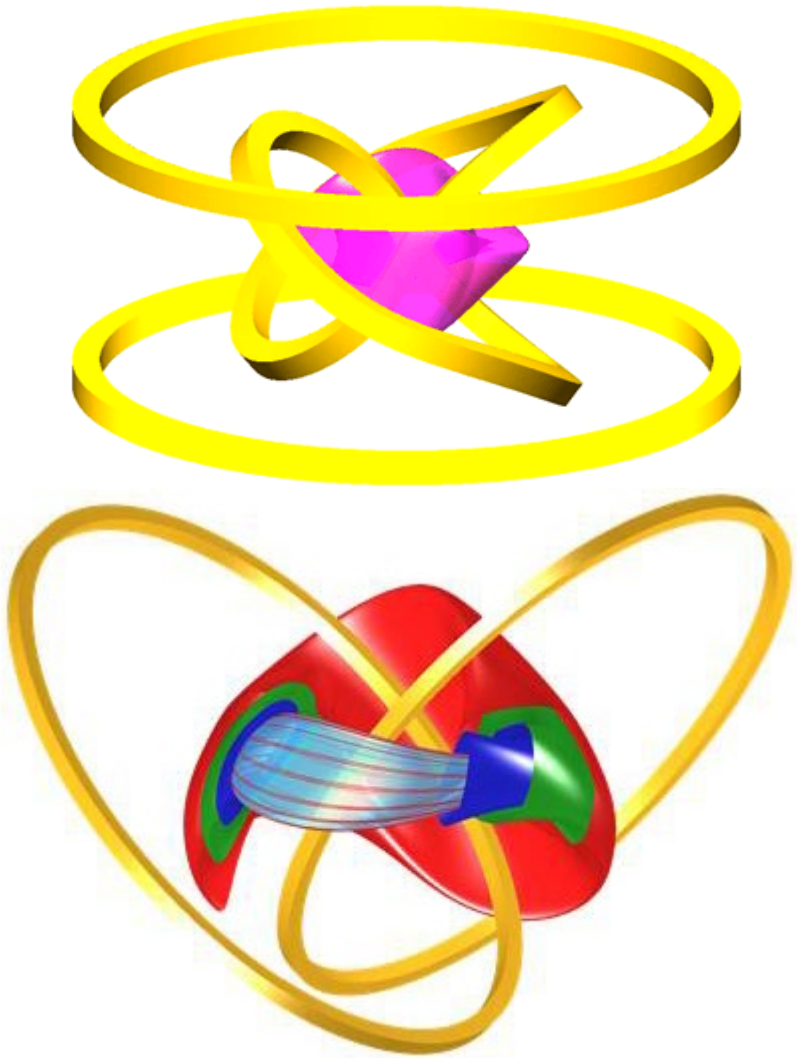
Particle constraint time distribution.

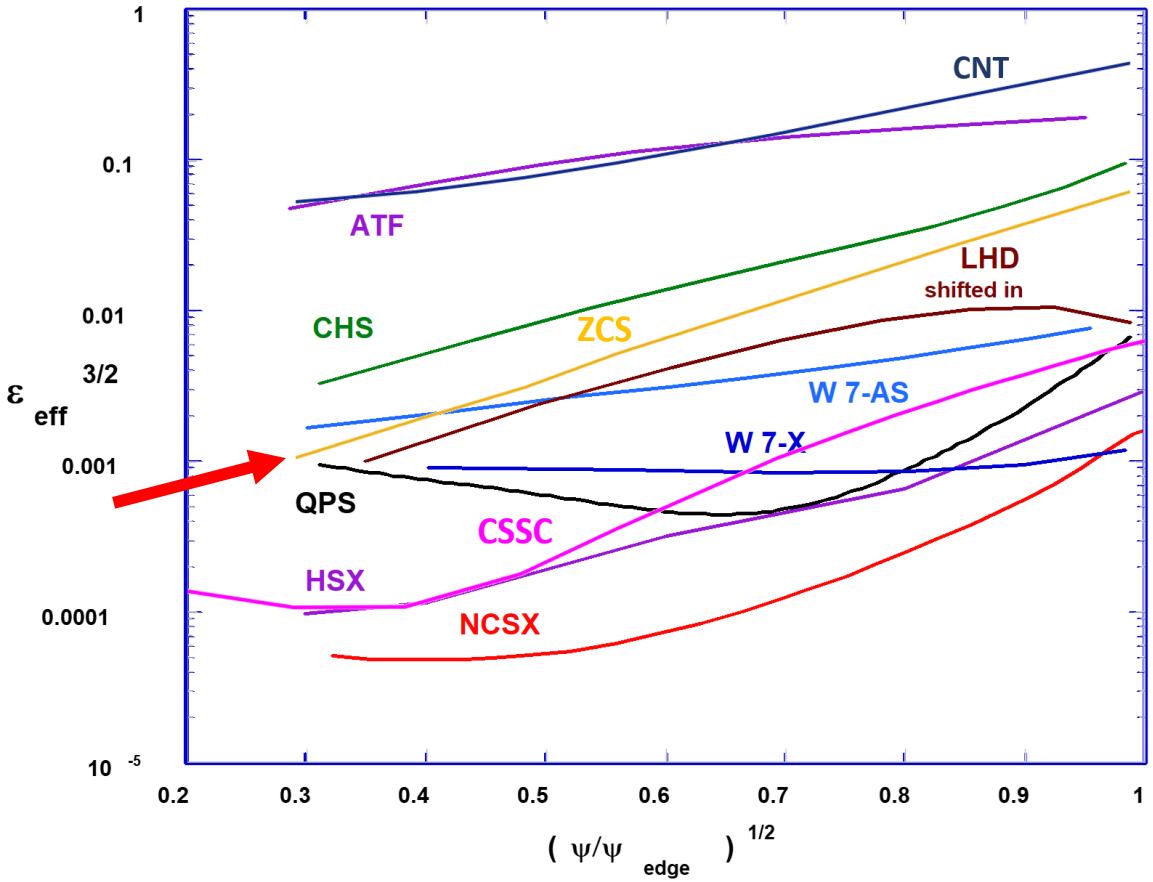
The initial positions are uniform in given magnetic surface.

K. Tani, et al (1983). *Ripple loss of suprathermal alpha particles during slowing-down in a tokamak reactor.* Nuclear Fusion

Optimization for stellarators

Zhejiang university Compact Stellarator



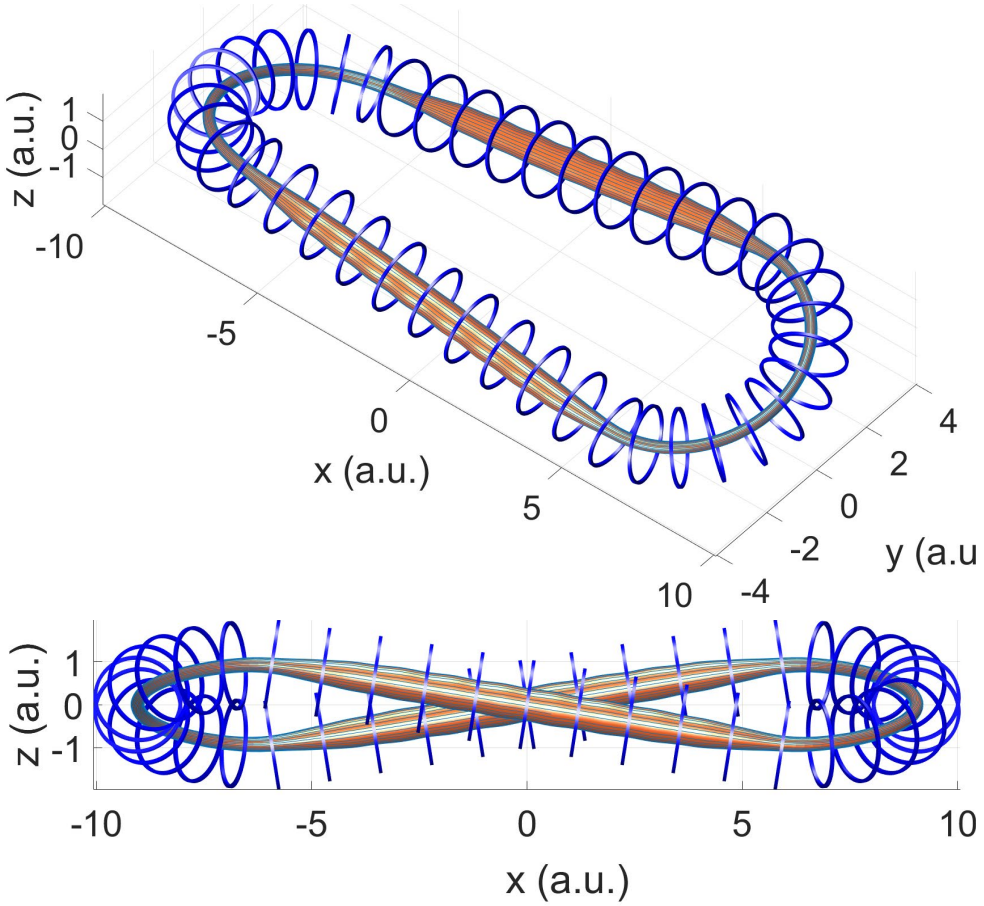


ZCS composed with only 4 simple coils. With highly optimization, the neoclassical transport coefficient of ZCS is close to some stellarators with complex coils.

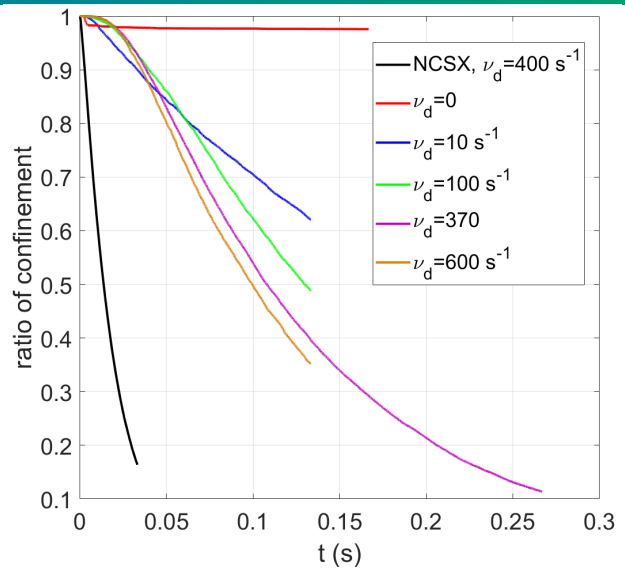
Yu, G., Feng, Z., **Jiang, P.**, & Fu, G. *Existence of an optimized stellarator with simple coils.* Journal of Plasma Physics

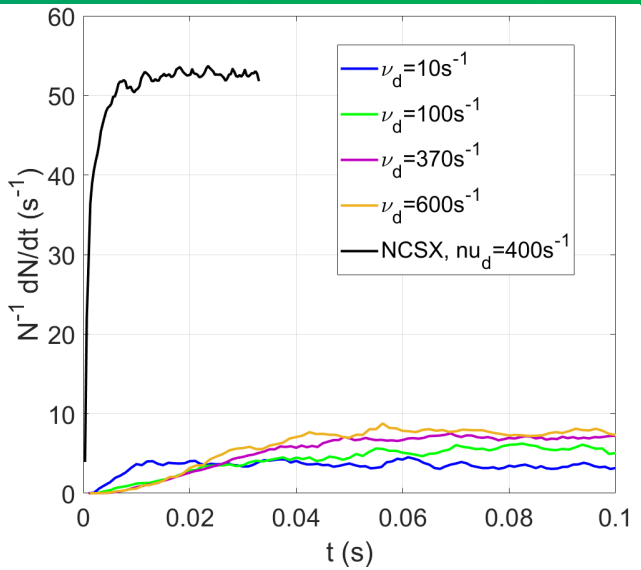
Optimization for stellarators

Linked mirrors stellarator

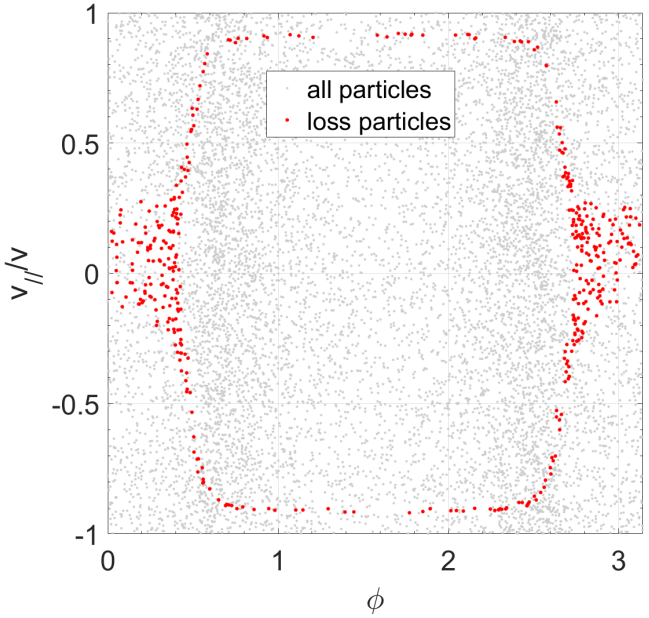


42 circular coils





Particle loss rate



Loss particle distribution

Feng, Z., Yu, G., **Jiang, P.**, & Fu, G. (2022). *Proposal of a linked mirror configuration for magnetic confinement experiments.* Nuclear Fusion

Outline

1. Background
2. Compile-time Symbolic Solver CSS
3. Optimization for fluid and particle simulations
4. Shifted metric method
5. Gyrokinetic MHD hybrid code GMEC
6. Field and particle code FP3D
7. Conclusion

Conclusion

- Compile-time Symbolic Solver (CSS) is a general-purpose finite difference framework for both PDEs and ODEs. It is also an instruction optimization framework to achieve faster speed than conventional codes.
- It supports:
  - arbitrary equations in curvilinear coordinate systems.
  - arbitrary coordinates with arbitrary boundary conditions
  - N-dimensional grids with TBB and MPI hybrid parallel scheme in arbitrary dimensions.

- CSS is used to generate and solve GMEC equations and FP3D equations.
- Benchmarks between GMEC and MAS for IBM, TM, BM with drift terms are done successfully. For TAE, GMEC agree well with others codes.
- FP3D is used to calculate magnetic surface, particle orbits, neoclassical transport coefficient and ripple losses, and benchmark with others codes successfully.

Prospect for GMEC

- With the high efficiency, GMEC is a powerful tool for investigate energetic particle-driven Alfven instabilities in burning plasmas

- We will add nonlinear FLR term soon to study the fully nonlinear dynamics of the Alfven instabilities driven by energetic particles.

- GMEC can be extended easily to 3D equilibria for stellarator applications. $g^{ij}[x, y, z]$

- With the high efficiency, GMEC can potentially be extended to full-f method with good calculation speed.

Prospect for CSS

Possible future application:

- 5D Vlasov equation using continuum method. $f[x, y, z, v_{\parallel}, \mu]$
- Eigenvalue codes with high efficiency.
- Finite element codes.
- CSS is easy to implement on GPU to get higher performances than CPU.

Thank you

Thank You
For Your Attention

Merging of coefficients

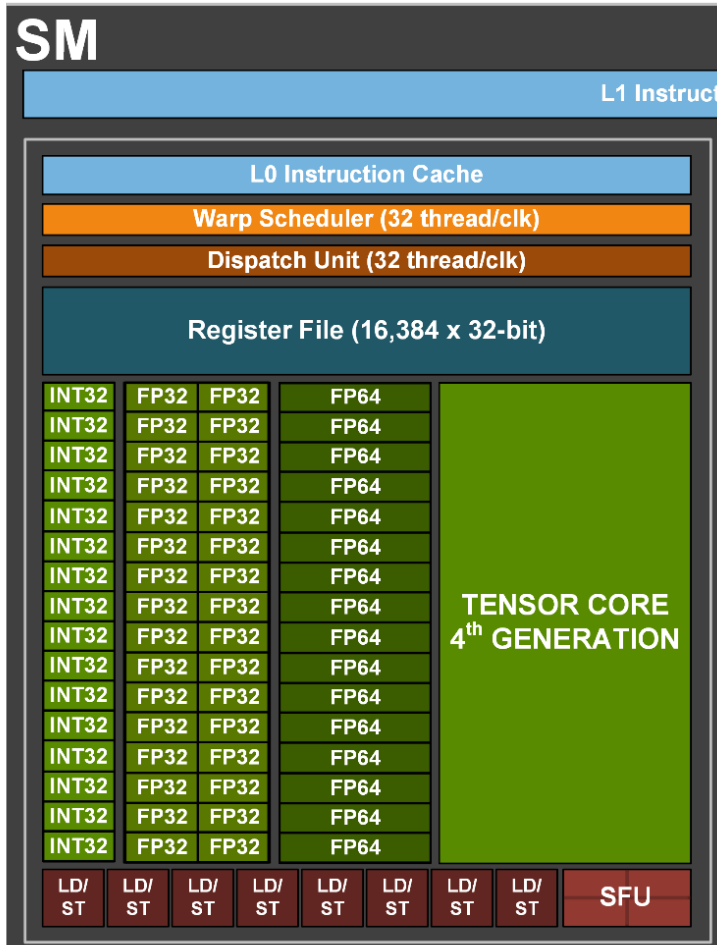
```
auto laplace = div(grad(f));
cout << "Div(Grad(phi))" << endl;
cout << laplace << endl;
```

```
Div(Grad(phi))

DJ/Dz * gzx * Dphi/Dx / J + DJ/Dz * gzy * Dphi/Dy / J + DJ/Dz * gzz * Dphi/Dz /
J + DJ/Dy * gyx * Dphi/Dx / J + DJ/Dy * gyy * Dphi/Dy / J + DJ/Dy * gyz * Dphi/D
z / J + DJ/Dx * gxx * Dphi/Dx / J + DJ/Dx * gxy * Dphi/Dy / J + DJ/Dx * gxz * Dp
hi/Dz / J + gxx * D^2phi/Dx^2 + Dgxx/Dx * Dphi/Dx + gxy * D^2phi/DxDy + Dgxy/Dx
* Dphi/Dy + gxz * D^2phi/DxDz + Dgxz/Dx * Dphi/Dz + gyx * D^2phi/DxDy + Dgyx/Dy
* Dphi/Dx + gyy * D^2phi/Dy^2 + Dgyy/Dy * Dphi/Dy + gyz * D^2phi/DyDz + Dgyz/Dy
* Dphi/Dz + gzx * D^2phi/DxDz + Dgzx/Dz * Dphi/Dx + gzy * D^2phi/DyDz + Dgzy/Dz
* Dphi/Dy + gzz * D^2phi/Dz^2 + Dgzz/Dz * Dphi/Dz
```

```
<Phi[3,0,-3] : 1/24gxz/(G_x*G_z)>
<Phi[0,0,-2] : 1/12Dgxz/Dx/(G_z) + -1/12gzz/(G_z^2) + 1/12Dgzz/Dz/(G_z) + 1/12gxz*DJ/Dx/(G_z*J) + 1/12gzz*DJ/Dz/(G_z*J)>
<Phi[2,0,-2] : -1/6gxz/(G_x*G_z)>
<Phi[-1,0,-1] : 11/24gxz/(G_x*G_z)>
<Phi[0,0,-1] : -2/3Dgxz/Dx/(G_z) + 4/3gzz/(G_z^2) + -2/3Dgzz/Dz/(G_z) + -2/3gxz*DJ/Dx/(G_z*J) + -2/3gzz*DJ/Dz/(G_z*J)>
<Phi[1,0,-1] : -1/4gxz/(G_x*G_z)>
<Phi[-1,0,0] : 11/12gxx/(G_x^2) + -1/4Dgxx/Dx/(G_x) + -1/4Dgxz/Dz/(G_x) + -1/4gxx*DJ/Dx/(G_x*J) + -1/4gxz*DJ/Dz/(G_x*J)>
<Phi : -5/3gxx/(G_x^2) + -5/6Dgxx/Dx/(G_x) + -5/6Dgxz/Dz/(G_x) + -5/2gzz/(G_z^2) + -5/6gxx*DJ/Dx/(G_x*J) + -5/6gxz*DJ/Dz/(G_x*J)>
<Phi[1,0,0] : 1/2gxx/(G_x^2) + 3/2Dgxx/Dx/(G_x) + 3/2Dgxz/Dz/(G_x) + 3/2gxx*DJ/Dx/(G_x*J) + 3/2gxz*DJ/Dz/(G_x*J)>
<Phi[2,0,0] : 1/3gxx/(G_x^2) + -1/2Dgxx/Dx/(G_x) + -1/2Dgxz/Dz/(G_x) + -1/2gxx*DJ/Dx/(G_x*J) + -1/2gxz*DJ/Dz/(G_x*J)>
<Phi[3,0,0] : -1/12gxx/(G_x^2) + 1/12Dgxx/Dx/(G_x) + 1/12Dgxz/Dz/(G_x) + 1/12gxx*DJ/Dx/(G_x*J) + 1/12gxz*DJ/Dz/(G_x*J)>
<Phi[-1,0,1] : -11/24gxz/(G_x*G_z)>
<Phi[0,0,1] : 2/3Dgxz/Dx/(G_z) + 4/3gzz/(G_z^2) + 2/3Dgzz/Dz/(G_z) + 2/3gxz*DJ/Dx/(G_z*J) + 2/3gzz*DJ/Dz/(G_z*J)>
<Phi[1,0,1] : 1/4gxz/(G_x*G_z)>
<Phi[0,0,2] : -1/12Dgxz/Dx/(G_z) + -1/12gzz/(G_z^2) + -1/12Dgzz/Dz/(G_z) + -1/12gxz*DJ/Dx/(G_z*J) + -1/12gzz*DJ/Dz/(G_z*J)>
<Phi[2,0,2] : 1/6gxz/(G_x*G_z)>
<Phi[3,0,3] : -1/24gxz/(G_x*G_z)>
```

Compile-time Red-black tree

```
// balance
template<typename X, typename LX, rbtree LL, rbtree LR, typename RX, rbtree RL, rbtree RR> struct balance<X, Node<Red,
 LX, LL, LR>, Node<Red, RX, RL, RR>> {
    using type = Node<Red, X, Node<Black, LX, LL, LR>, Node<Black, RX, RL, RR>>;
};
template<typename X, typename LX, typename LLX, rbtree LLL, rbtree LLR, rbtree LR, rbtree R> struct balance<X,
 Node<Red, LX, Node<Red, LLX, LLL, LLR>, LR>, R> {
    using type = Node<Red, LX, Node<Black, LLX, LLL, LLR>, Node<Black, X, LR, R>>;
};
template<typename X, typename LX, rbtree LL, typename LRX, rbtree LRL, rbtree LRR, rbtree R> struct balance<X,
 Node<Red, LX, LL, Node<Red, LRX, LRL, LRR>>, R> {
    using type = Node<Red, LRX, Node<Black, LX, LL, LRL>, Node<Black, X, LRR, R>>;
};
template<typename X, rbtree L, typename RX, rbtree RL, typename RRX, rbtree RRL, rbtree RRR> struct balance<X, L,
 Node<Red, RX, RL, Node<Red, RRX, RRL, RRR>>> {
    using type = Node<Red, RX, Node<Black, X, L, RL>, Node<Black, RRX, RRL, RRR>>;
};
template<typename X, rbtree L, typename RX, typename RLX, rbtree RLL, rbtree RLR, rbtree RR> struct balance<X, L,
 Node<Red, RX, Node<Red, RLX, RLL, RLR>, RR>> {
    using type = Node<Red, RLX, Node<Black, X, L, RLL>, Node<Black, RX, RLR, RR>>;
};
```

GPU structure