MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

JOREK: Advancements in GPU Porting of MHD Solver

Ihor Holod and Patrik Rac
IPP-Garching

EUROfusion

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Outline

- **Motivation**
- **Overview of JOREK code**
- **MHD Solver Algorithm**
- **GPU Porting of Stiffness Matrix Construction**
- **GPU Porting of Iterative Solver**
- **Summary**

JOREK: overview

- **JOREK is an extended nonlinear MHD code used to study large scale plasma instabilities and their control in realistic divertor geometry**
  - **IPP-Garching is hosting one of the main hubs for the code development in the European and international community**
  - **JOREK is written in modern FORTRAN with MPI/OpenMP hybrid parallelization**
- **Several models are implemented in JOREK with different sets of physical quantities, including full-MHD, and various implementations of reduced MHD models**
  - **MHD equations in weak form are spatially discretized on continuous 2D isoparametric Bezier finite element grid in poloidal plane, combined with a toroidal Fourier expansion**
  - **Implicit time integration scheme allows large time stepping for realistic simulations**
- **Several hybrid kinetic-fluid models are also available, e.g. for ITG turbulence, neutrals, impurities, energetic particles, relativistic runaway electrons**

JOREK: MHD solver and global sparse matrix

Generalized form of MHD equation

$$\frac{\partial A(u)}{\partial t} = B(u,t)$$

Linearized implicit time discretization scheme yields

$$\left[(1+\xi)\left(\frac{\partial A}{\partial u}\right)^n - \Delta t\theta\left(\frac{\partial B}{\partial u}\right)^n\right]\delta u^n = \Delta t B^n + \xi\left(\frac{\partial A}{\partial u}\right)^{n-1}\delta u^{n-1}$$

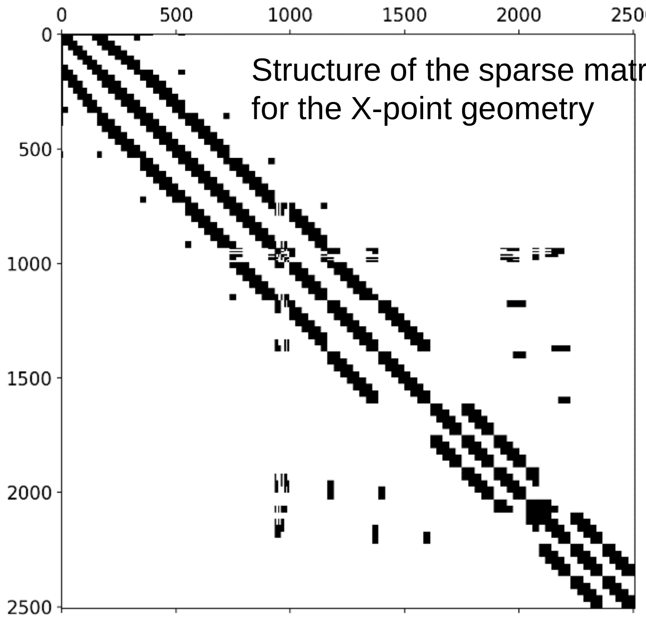
$$\delta u^n = u^{n+1} - u^n$$

Linear system of algebraic equations

$$Ax = b$$

A is a sparse matrix, typically large and ill-conditioned

Example: 30K nodes; 8 physical variables; 4 dof per node; 21 toroidal harmonics: matrix dimension 40 million with 500 billion non-zero elements – requires 8 TB of memory for storage



Structure of the sparse matrix for the X-point geometry

Physics-based preconditioner

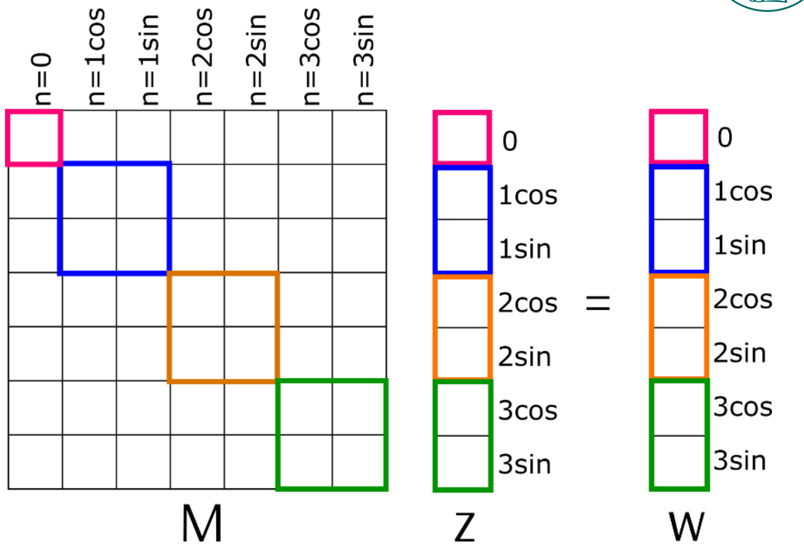
- Direct LU factorization is (usually) prohibitively expensive
- Iterative GMRES method with (left) preconditioning is used

Preconditioned system to be solved: $M^{-1}Ax = M^{-1}b$

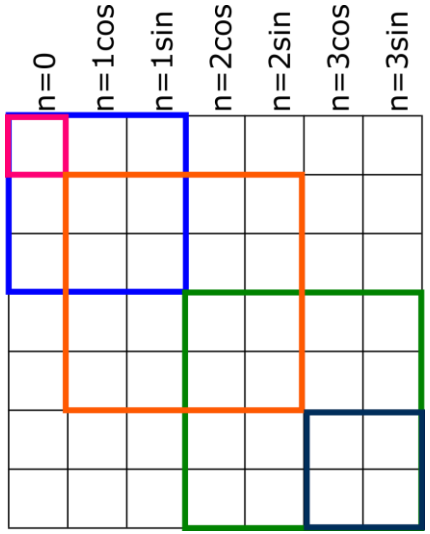
Product $M^{-1}A$ should have low condition number

Solution $z = M^{-1}w$ should be easy to find

Preconditioner matrix doesn't appear explicitly, only in form of a solution



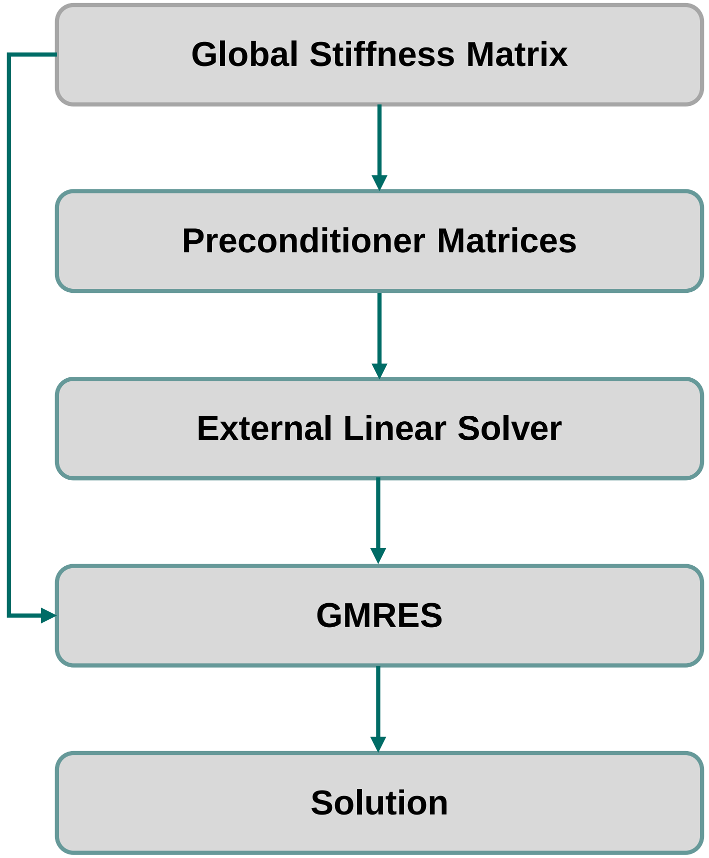
- JOREK preconditioner is bases on decoupling individual toroidal Fourier modes of mode families
    - Full preconditioner matrix is equivalent to the original matrix A with omitted mode coupling
    - Each diagonal block has similar sparsity pattern as A
    - Each diagonal block can be solved independently



Solver algorithm

Solver algorithm:

- Construct global stiffness matrix and RHS – *every time step*
- Construct/distribute preconditioner matrix – ***once per several steps***
- Analyze/build elimination graph – *once per simulation run*
- Perform LU factorization – ***once per several steps***
- Perform GMRES/BICGSTAB iterations – *every step*
  - Find solution for preconditioner matrix – *every iteration*



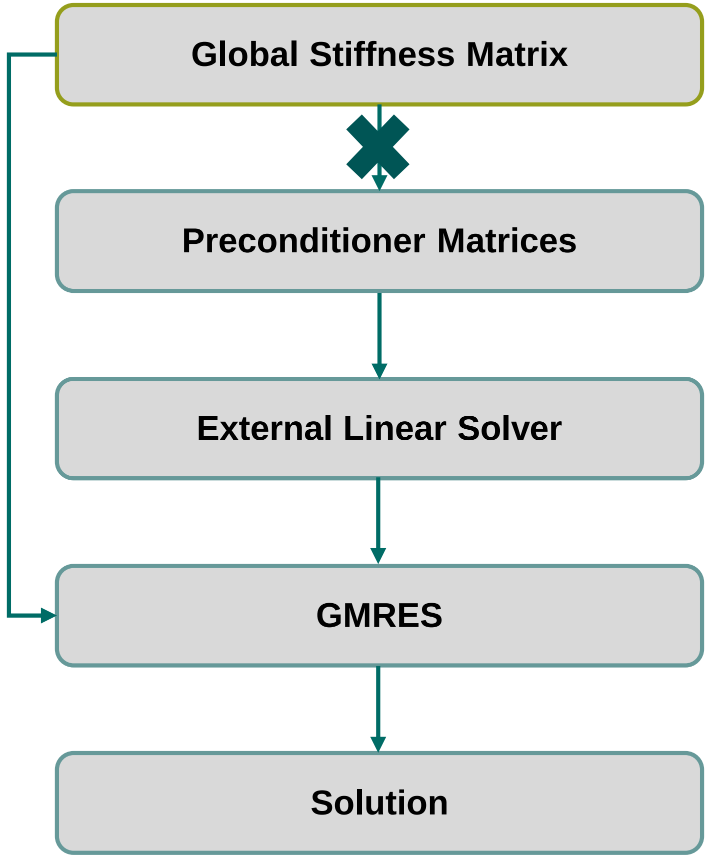
GPU Acceleration Strategy

Acceleration of Preconditioner Solver

- Vendor specific

Acceleration of Matrix Construction/Iterative Solver

- Global Matrix constructed/residing on GPU
- Matrix-vector product in iterative cycle calculated on GPU
- Direct construction used for Preconditioner matrices



GPU Accelerated Matrix Construction

CPU Approach

- Finite Elements distributed among MPI tasks and OpenMP threads
- Element matrices are computed using SIMD vectorization

GPU Approach using OpenMP Offloading

- Element distributed among MPI tasks and OpenMP *teams*
- Element matrices are computed in batches of many elements
- Loop over elements and internal loops are distributed over *teams* and *SM threads*
  - Loop restructuring was necessary to obtain good performance on accelerators

Optimized FFT Libraries

- The Fast Fourier Transform is performed using *CuFFT/RocFFT*
- The transforms for multiple elements are batched for maximum efficiency.

GPU Accelerated Matrix Construction

Element coloring

- Element coloring is used to…
  - …remove synchronization bottleneck
  - …reduce memory cost by element batching.

Performance

- Efficiency is very setup dependent.
- A reasonable setup can lead to a decent speed up of ~2 on HCP Raven node (2x Intel Xeon IceLake-SP 8360Y, 72 cores per node, 4x Nvidia A100)

GPU Accelerated Matrix-Vector Product

Original CPU approach based on matrix block structure:

- **Blocks distributed among OpenMP threads**
- **MKL BLAS used for individual block multiplication**

Better performance achieved using compressed sparse row (CSR) format

- **Row pointers distributed among OpenMP threads**
- **Explicit summation over column indices with SIMD distribution**
- **COO-to-CSR mapping is pre-calculated**

GPU implementation using CSR format

- **Row pointers distributed among OpenMP teams**
- **Explicit summation over column indices with SM thread distribution**

Matrix block-structure (bs=3)

11,12,13 21,22,23 31,32,33	14,15,16 24,25,26 34,35,36

11,12,13,21,22,23,31,32,33,14,15,16,…

Summary

- **GPU offloading of JOREK MHD Solver components has been implemented using OpenMP library**
  - **GPU acceleration of stiffness matrix construction with coloring method**
  - **GPU acceleration of matrix-vector product in the iterative solver**
- **The unification of these components is currently underway**