MAX-PLANCK-INSTITUT
FÜR PLASMAPHYSIK

JOREK: advancements in MHD Solver

Ihor Holod, 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
- Preconditioner
- Re-implementation of iterative solver
  - Newton’s method
  - BiCGSTAB method
- Further plans

JOREK: overview

- JOREK is an extended nonlinear MHD code used to study large scale plasma instabilities and their control in realistic divertor geometry
  - IPP 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
- Several hybrid kinetic-fluid models are also available, e.g. for ITG turbulence, neutrals, impurities, energetic particles, relativistic runaway electrons

JOREK: MHD solver

Generalized form of MHD equation

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

Linearized implicit time discretization scheme yields

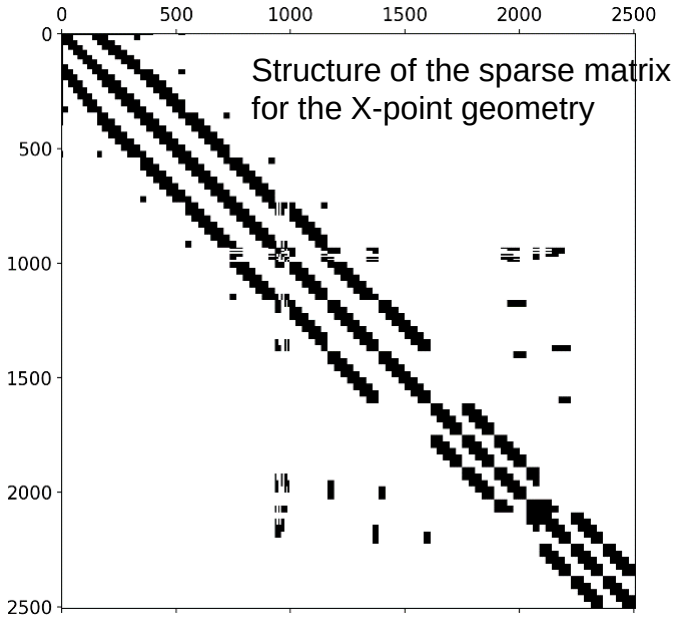
$$\left[(1+\xi)\left(\frac{\partial A}{\partial u}\right)^n - \Delta t\theta\left(\frac{\partial B}{\partial u}\right)^n\right]\delta u^n = \Delta t B^n + \xi\left(\frac{\partial A}{\partial u}\right)^{n-1}\delta u^{n-1} \qquad \delta u^n = u^{n+1} - u^n$$

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

Solver algorithm

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

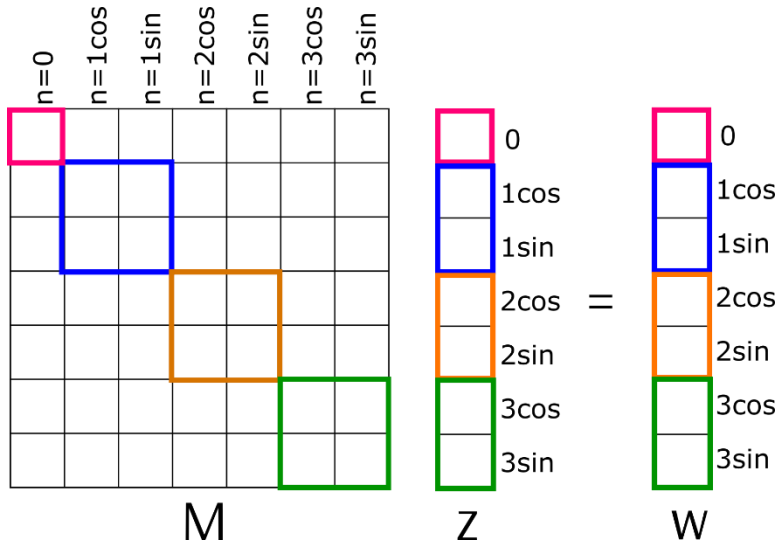
Preconditioner matrix doesn't appear explicitly, only in form of a solution

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*

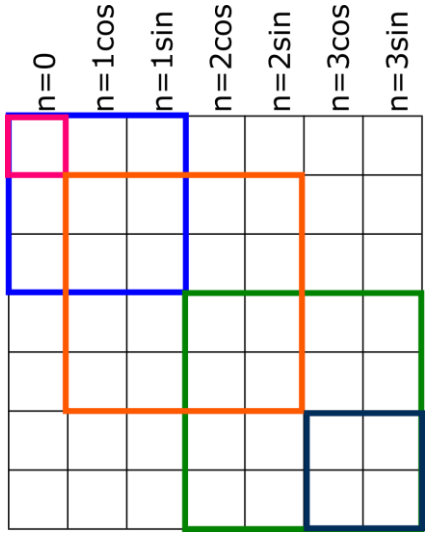
Physics-based preconditioner

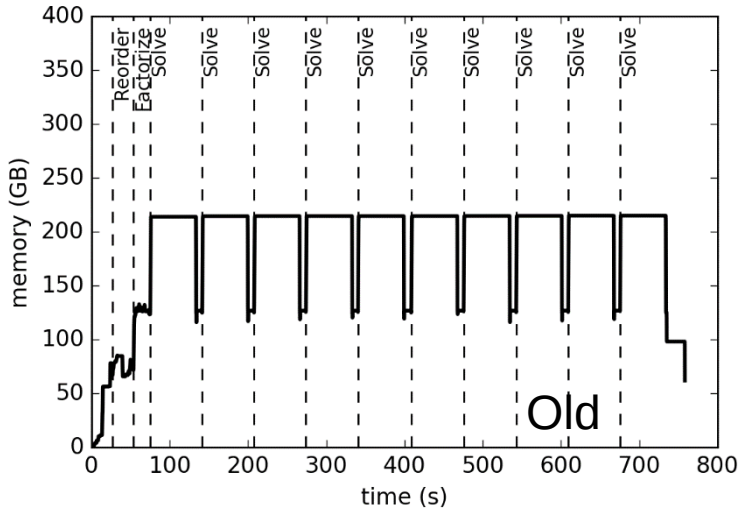
- JOREK preconditioner is bases on decoupling individual toroidal Fourier modes of mode families
  - Full preconditioner matrix is equivalent to the original matrix A with omitted mode coupling
  - Each diagonal block has similar sparsity pattern as A
  - Each diagonal block can be solved independently

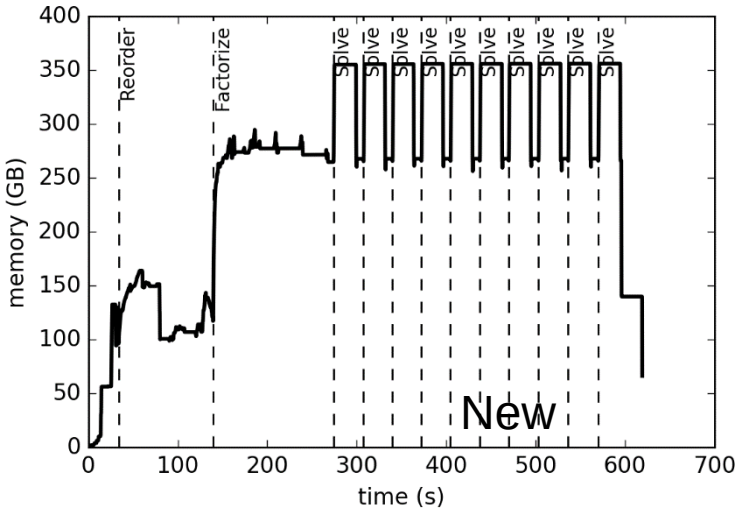


Generalized Preconditioner (mode groups)

- Individual preconditioner block-matrix can include arbitrary harmonics (mode groups)
- ***Convergence may improve significantly with new preconditioner***
- *Increased memory consumption*







Re-implementation of solver subroutines

- There are few places in the code where sparse linear system is solved: Grad-Shafranov equilibrium, direct- and iterative MHD solvers
  - Three choices for external libraries: MUMPS, PaStiX (v.5 and v.6) or STRUMPACK
- Problem with **redundancies**, **complexity** and **non-locality**
- The purpose of this development is creating universal solver interface subroutine which takes the sparse matrix and the right hand side vector as input, and provide the solution vector as the result
- Inside such subroutine two options are implemented for the **direct** and **iterative** approaches.
  - With the iterative approach there is additional solver call for the preconditioner, which is **identical** to the direct one, except the input matrix is distributed over the different sets of MPI tasks
- **New restructured solver allows simplified implementation of new features and improved flexibility in managing computational resources.**

New data-types and interfaces

- New data types
  - Passing structured data as procedure arguments for localization and control vs. global variables
  - Centralized definition of data types allows easy adjustments and addition of new features
  - Type-bound procedures are developed to handle **move**, **copy** and **delete** operations
- New unified solver interface

```
subroutine solve_sparse_system(a_mat, rhs_vec, sol_vec, solver)
type(type_SP_SOLVER)     :: solver
type(type_SP_MATRIX)     :: a_mat
type(type_RHS)           :: rhs_vec, sol_vec
```

  - The direct-solve procedure is hidden in the subroutines specific to different external solver libraries, including standard steps, such as *initialization*, *analysis*, *factorization* and *solve*
  - **No duplication**

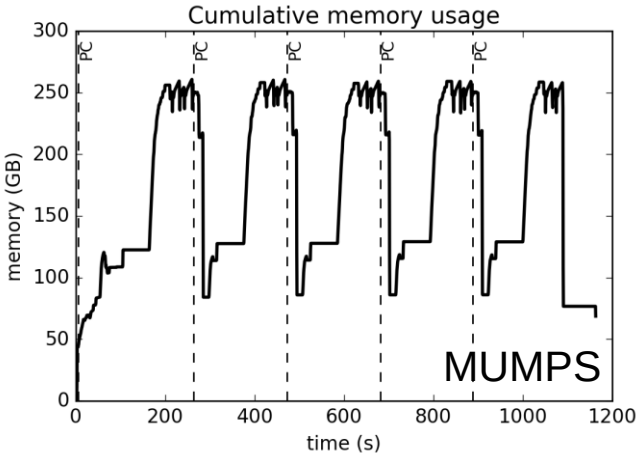
Improving performance with STRUMPACK library

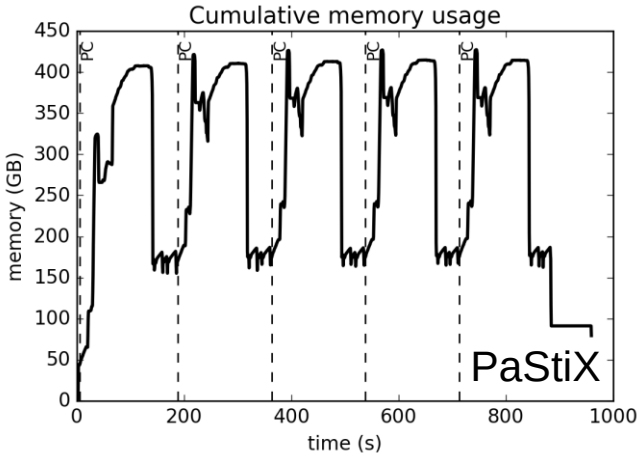
- Using STRUMPACK as direct solver allows fully distributed construction and analysis of preconditioner block-matrices and memory purging from previous LU-factorization
- ELM simulation test case with 2 Fourier modes, using single-mode-family preconditioner.
  - Global stiffness matrix has rank of 3.1M and 2.3B nonzero entries
  - Number of nonzero entries in the preconditioner matrices are 258M for **`n=0`** and 1B for **`n=1`** modes, with the respective ranks of 1M and 2M
- Simulation performed on Marconi-Fusion HPC using 6 computational nodes and 12 OpenMP threads (4 MPI tasks per node)
- The time performance of the iterative solver hasn't changed significantly in the restructured solver

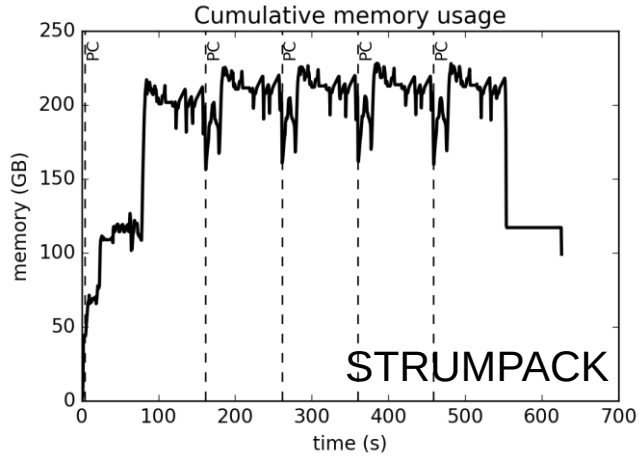
	PaStiX v.5.2.3	STRUMPACK v.7.0.1	MUMPS v.5.2.1
Step with A+F (s)	132	104	170
Step with F (s)	131	59	145
Step with 15 iterations (s)	14.6	11	13.3

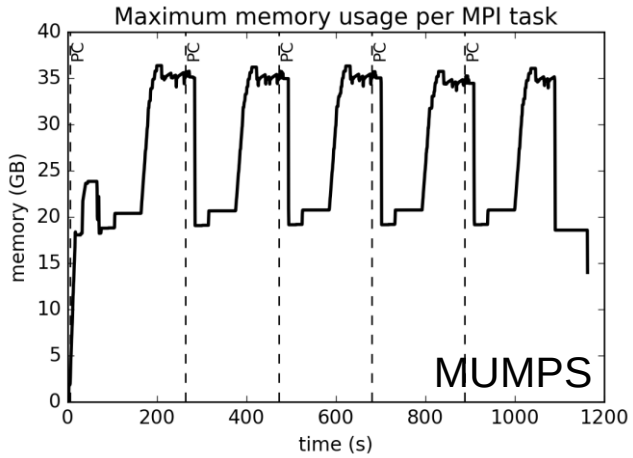
Improvement in memory usage

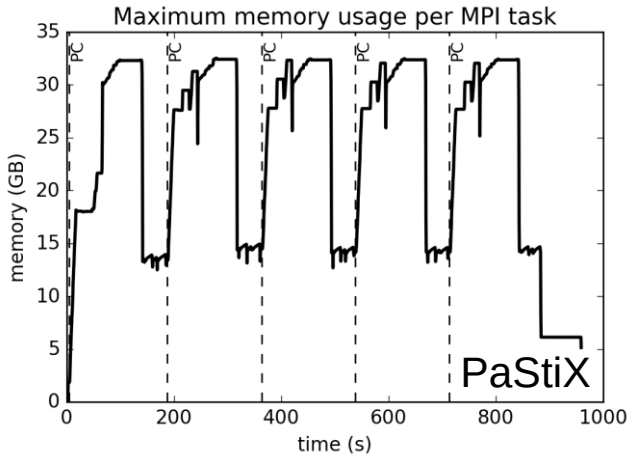
- Global stiffness matrix is distributed among MPI tasks of MPI_COMM_WORLD
- Preconditioner sparse matrices are distributed or cloned (PaStiX) among MPI tasks of MPI_COMM_N
- LU factorized matrices are distributed or cloned among MPI tasks of MPI_COMM_N

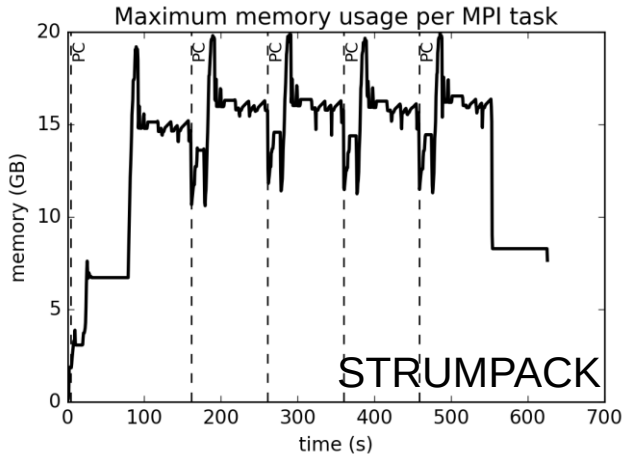












Inexact Newton Solver

- Find solution iteratively, reducing the precision at intermediate steps

$$G(u^{n+1}) = b(u^n, u^{n-1})$$

$$\left.\frac{\partial G}{\partial u}\right|_{u^n} \delta u^{n+1} = -G(u^n) + b(u^n, u^{n-1}) \qquad \delta u^{n+1} = u^{n+1} - u^n$$

$$\left.\frac{\partial G}{\partial u}\right|_{u_{k-1}} \delta u_k = -G(u_{k-1}) + b(u^n, u^{n-1})$$

$$G(u_k) = G(u^n) + \left.\frac{\partial G}{\partial u}\right|_{u^n} \delta u_k^n$$

- Equation to solve

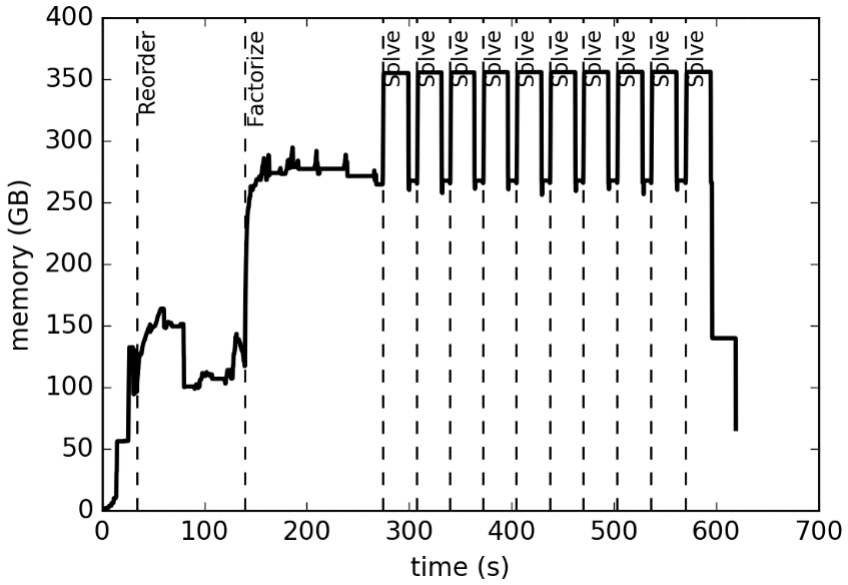
$$J_{k-1}\delta u_k = R_{k-1}$$

$$R_k = R^n - J^n \delta u_k^n \qquad R^n = -G(u^n) + b(u^n, u^{n-1}) \qquad J_k = \left.\frac{\partial G}{\partial u}\right|_{u_k}$$

- Option **`use_newton`** in the input file wraps around the existing sparse solver

BiCGSTAB iterative method in JOREK

- Maximum memory utilization occurs during the iterative part of the MHD solver
- Restarted GMRES method requires allocation of the whole Krylov subspace for the maximum number of iterations before restart
- BiCGSTAB (Bi-Conjugate Gradient Stabilized) method can be used as an alternative to GMRES with significantly lower memory requirements



BiCGSTAB implementation

- BICGSTAB algorithm is similar to the GMRES algorithm in terms of the main operations:
  - Construct preconditioner solution
  - Perform matrix-vector multiplication
- BICGSTAB method is implemented in JOREK as a stand-alone module solvers/mod_bicgstab.f90
- BICGSTAB uses the same control parameters as GMRES for maximum iterations and convergence tolerance (iter_gmres, gmres_tol)
- BICGSTAB can be used as an alternative to GMRES by setting USE_BICGSTAB=1 in the Makefile.inc
- **Cons:**
  - BICGSTAB method can fail to converge
  - Convergence can be worse compared to GMRES
  - For the same number of iterations BICGSTAB has slower performance: every iteration requires 2 calls for the preconditioner solve and for the matrix-vector multiplication
- **Pros:**
  - BICGSTAB requires less memory
  - Convergence can be better compared to GMRES

Further developments

- Targeting GPU accelerated MHD solver
  - GPU acceleration/optimization of stiffness matrix construction
  - Possibility of using smaller preconditioner at the expense of increased computation cost
- Improving convergence of iterative solver
  - Employ ML techniques to provide better initial guess
  - Implement modern algebraic methods to reduce condition number

Porting JOREK to HLRS HAWK

- HAWK production node type Zen2 aka “Rome”
  - 2x**AMD EPYC** 7742 processors (64 cores @ 2.25GHz, AVX2)
  - DRAM: 256GB @ 380GB/s
- Available compilers:
  - AOCC is the AMD Optimized C/C++ Compiler based on LLVM
  - Intel
  - GNU
- MPI Libraries: MPT, OpenMPI
- JOREK and solver libraries are built and tested with different combinations of compiler+MPI
  - Performance of AOCC+MPT and Intel+MPT are comparable when using STRUMPACK
  - With PaSTiX the internal threading fails when AOCC+MPT is used
  - GNU+OpenMPI yields sub-optimal performance

JOREK performance on HAWK

- The size of the test problem used to study strong scaling behavior is relatively modest:
  - System rank 1.4M; 2.6B nonzero entries
  - LU-factorization of 5 sparse matrices of up to 845M non-zero entries each. Corresponding factorized matrices contain up to 5.9B elements.
  - The total memory requirement is ~350GB.
  - Minimum number of nodes for this case is 4.
- The departure from the ideal scaling is due to the increased communication time during matrix construction, as well as relatively poor scaling performance of the sparse solver libraries.
- Both issues could be addressed partially via fine-tuned optimization. Much better scalability is expected for particle-dominated simulations.