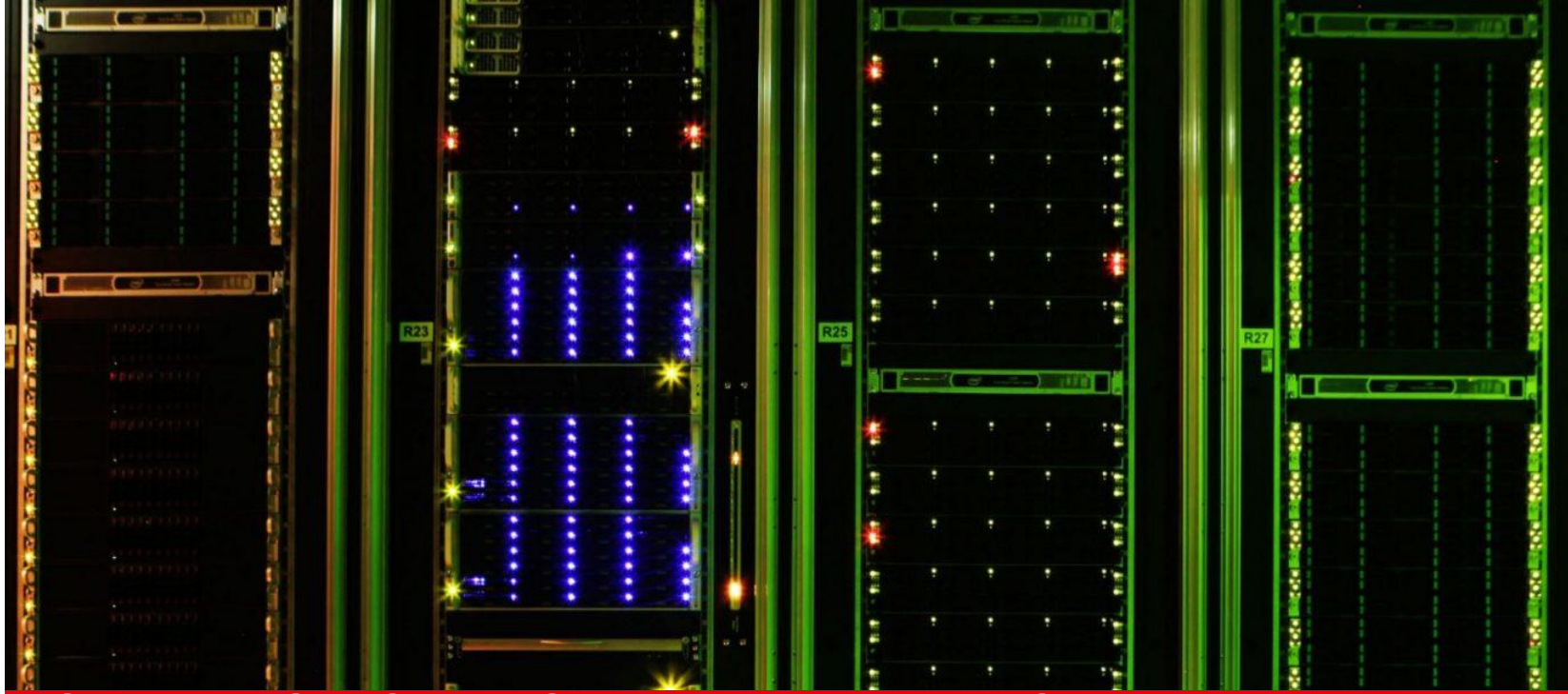
EPFL

EUROfusion



Soledge3X Code Optimization and GPU Production Computing for the WEST Tokamak

M. Peybernes, E. Bourne (EPFL - SCITAS)

H. Bufferand, D. Sales de Oliveira, P. Tamain (Soledge group - CEA)

4th Annual meeting of EUROfusion HPC ACHs – October 2026

SWISS PLASMA CENTER

EPFL SCITAS
Scientific IT and Application Support

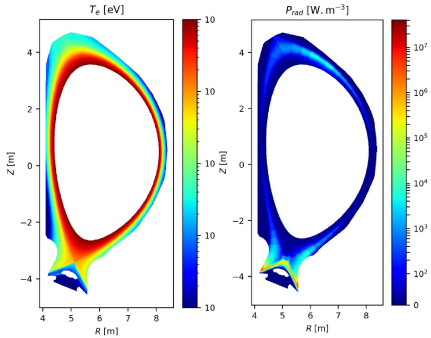
SOLEDGE3X: a versatile fluid code for the edge plasma

- **SOLEDGE3X**: multi-fluid modelling tool for the edge plasma

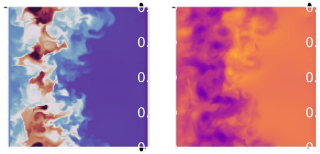
- Key features:
  - **Neutrals** either fluid (embedded) or kinetic (EIRENE)
  - Complete plasma **geometrical flexibility** (arbitrary number of X-points)
  - Usable in **2D or 3D**
  - Usable as **mean-field or self-consistent turbulence** code

- The numerical scheme uses:
  - mix explicit-implicit scheme
  - based on 2D or 3D finite volumes
  - WENO methods for the advection
  - following terms are treated implicitly
    - ➢ Parallel viscosity
    - ➢ Parallel heat conduction
    - ➢ vorticity

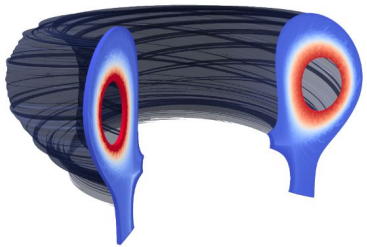
2D mean-field



2D turbulence



3D turbulence



GPU porting strategy

- GPU porting
  - Maintain a single version of the code
  - Ensure code portability and readability
  - Generic pragma for OpenMP/OpenACC
    - Same approach for other codes such as ASCOT5 and CAS3D
  - Open ACC/MP for advection and matrix construction
  - PETSC (with CUDA/HIP) for linear solvers

```
GPU_LOOP_ALL_LEVELS collapse(3)
do ipsi = ipsimin, ipsimax
   do itheta = ithetamin, ithetamax
      do iphi = iphimin, iphimax
    …some work
      end do !iphi
   end do !itheta
end do !ipsi
GPU_END_LOOP_ALL_LEVELS
```

```
#ifndef gpu_commands
#define gpu_commands
#ifdef _OPENMP

#define GPU_MAP_TO_DEVICE !$omp target enter data map(to:
#define GPU_MAP_FROM_DEVICE !$omp target exit data map(from:

#define GPU_ALLOC_ON_DEVICE !$omp target enter data map(alloc:
#define GPU_DELETE_FROM_DEVICE !$omp target exit data map(delete:

#define GPU_LOOP_ALL_LEVELS !$omp target teams distribute parallel do simd
#define GPU_END_LOOP_ALL_LEVELS !$omp end target teams distribute parallel do simd

#define GPU_LOOP_LEVEL_1 !$omp target teams distribute
#define GPU_END_LOOP_LEVEL_1 !$omp end target teams distribute

#define GPU_LOOP_LEVEL_2 !$omp parallel do simd
#define GPU_END_LOOP_LEVEL_2 !$omp end parallel do simd

#elif _OPENACC

#define GPU_MAP_TO_DEVICE !$acc enter data copyin(
#define GPU_MAP_FROM_DEVICE !$acc exit data copyout(

#define GPU_ALLOC_ON_DEVICE !$acc enter data create(
#define GPU_DELETE_FROM_DEVICE !$acc exit data delete(

#define GPU_LOOP_ALL_LEVELS !$acc parallel loop
#define GPU_END_LOOP_ALL_LEVELS !$acc end parallel loop

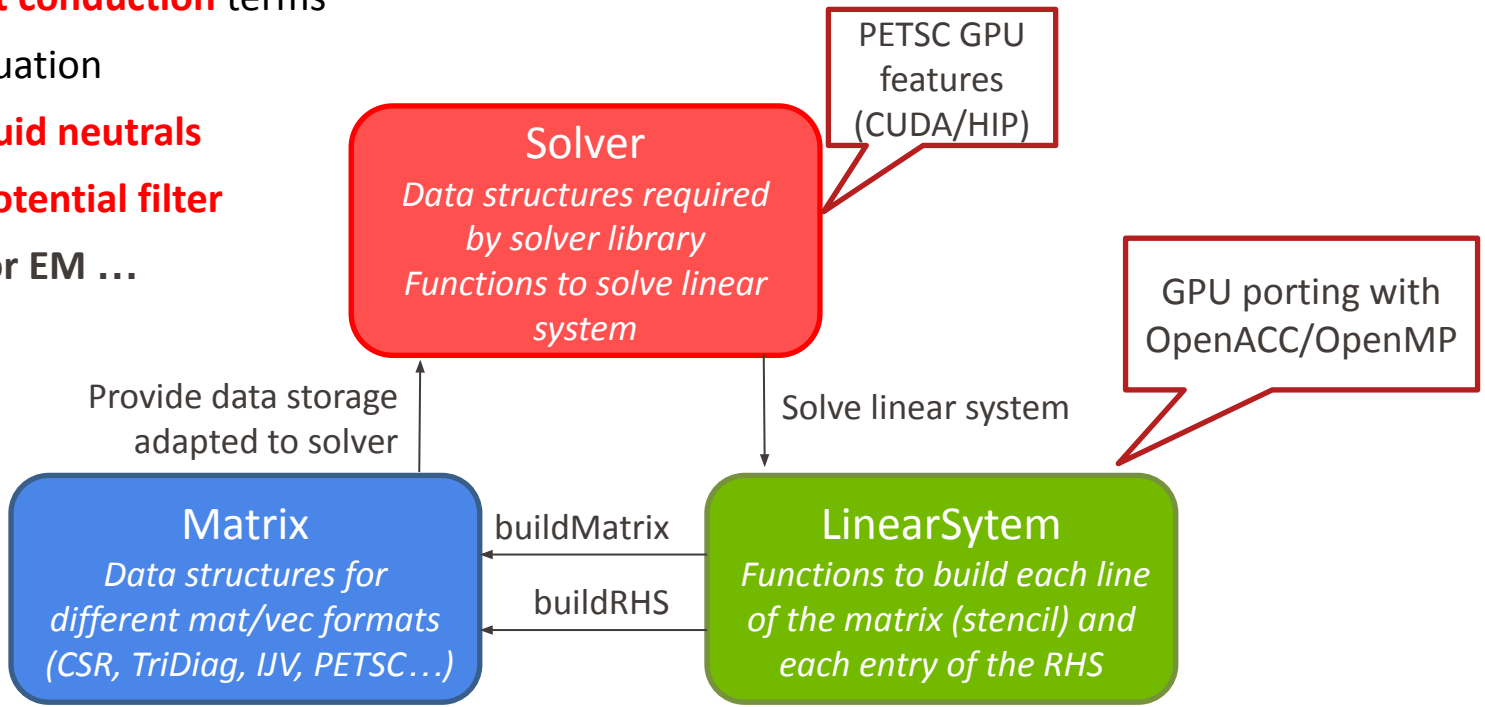
#define GPU_LOOP_LEVEL_1 !$acc parallel loop gang
#define GPU_END_LOOP_LEVEL_1 !$acc end parallel loop

#define GPU_LOOP_LEVEL_2 !$acc loop worker vector
#define GPU_END_LOOP_LEVEL_2

#endif
#endif
```

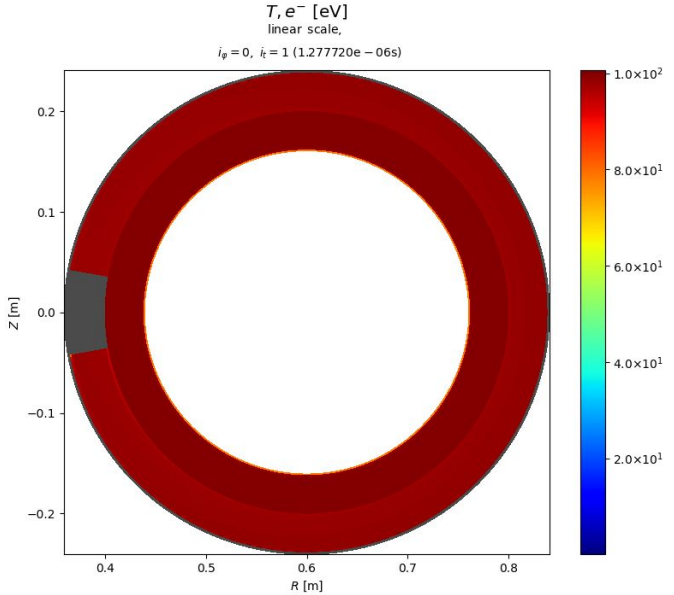
Implicit solvers

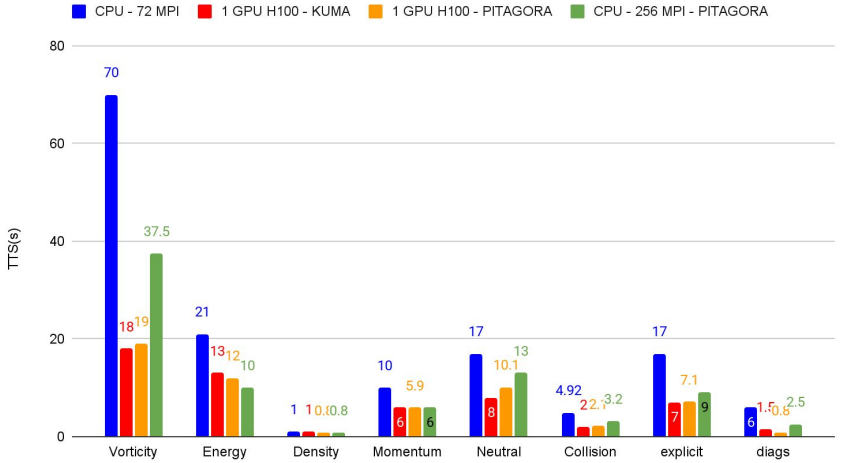
- 5 implicit solvers in SOLEDGE3X :
  - 2D
    - **Parallel viscosity** terms
    - **Parallel heat conduction** terms
  - 3D
    - **Vorticity** equation
    - (optional) **fluid neutrals**
    - (optional) **potential filter**
    - **and more for EM …**

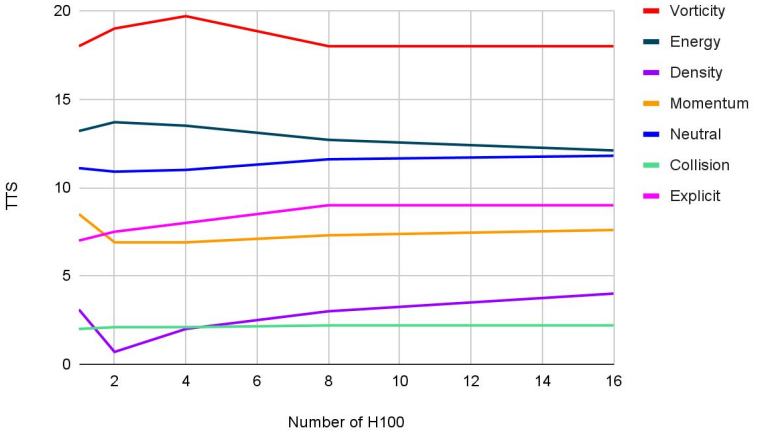


Preliminary results

- Benchmark:
  - **Jed@EPFL**: 2x Platinum 8360Y - 72 MPI
  - **Kuma@EPFL**: H100 - 1 GPU
  - **Pitagora@CINECA**: H100 - 1 GPU
  - **Pitagora@CINECA**: 256 MPI
  - Test case: Npsi = 64, Ntheta = 512, Nphi = 64

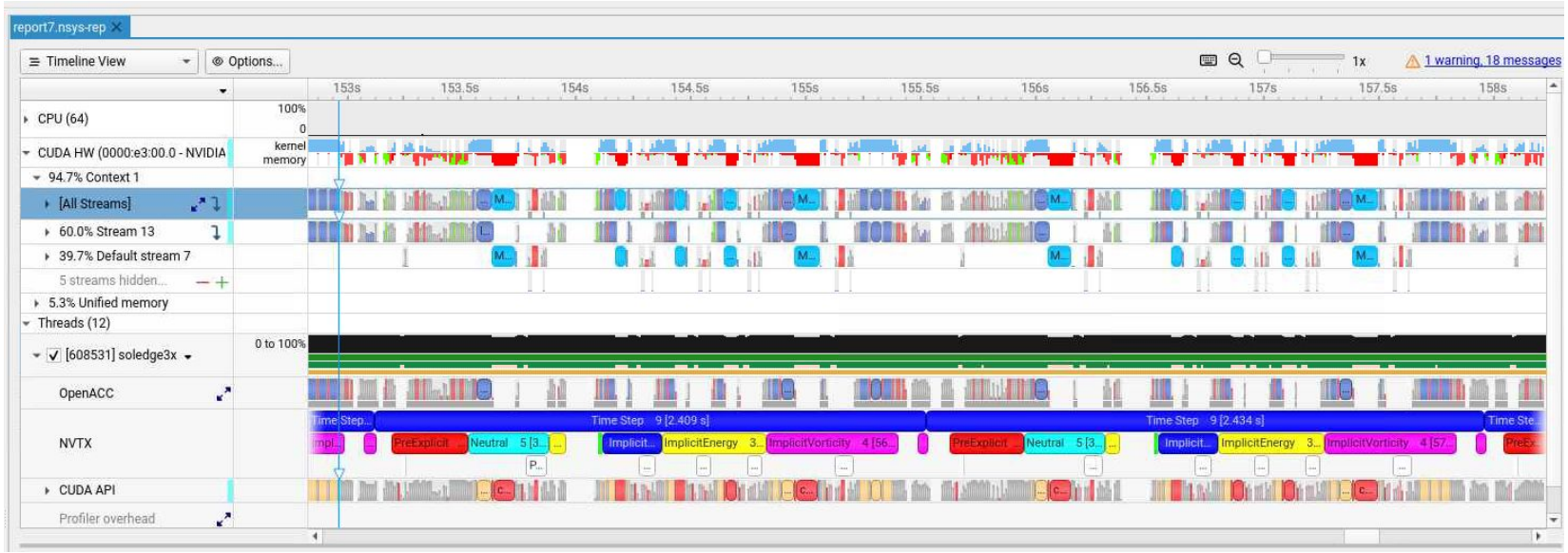






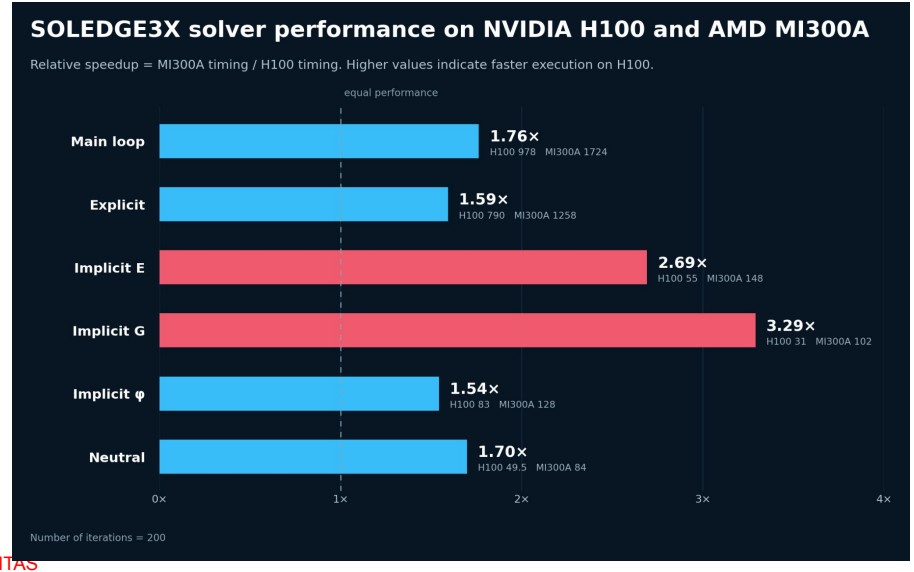
SOLEDGE3X - GPU profiling

- Nsight Systems
- GPU activity
- MPI + GPU interaction



Portability with OpenMP Offload

- OpenMP Offload is now functional
- First runs on AMD GPU MI300A
- AMD compiler
- NVIDIA H100 vs AMD MI300A
- Fix only related to LLVM compilers issues (cpu or gpu)




```
#ifndef gpu_commands
#define gpu_commands
#ifdef _OPENMP

#define GPU_MAP_TO_DEVICE !$omp target enter data map(to:
#define GPU_MAP_FROM_DEVICE !$omp target exit data map(from:

#define GPU_ALLOC_ON_DEVICE !$omp target enter data map(alloc:
#define GPU_DELETE_FROM_DEVICE !$omp target exit data map(delete:

#define GPU_LOOP_ALL_LEVELS !$omp target teams distribute parallel do simd
#define GPU_END_LOOP_ALL_LEVELS !$omp end target teams distribute parallel do simd

#define GPU_LOOP_LEVEL_1 !$omp target teams distribute
#define GPU_END_LOOP_LEVEL_1 !$omp end target teams distribute

#define GPU_LOOP_LEVEL_2 !$omp parallel do simd
#define GPU_END_LOOP_LEVEL_2 !$omp end parallel do simd

#elif _OPENACC

#define GPU_MAP_TO_DEVICE !$acc enter data copyin(
#define GPU_MAP_FROM_DEVICE !$acc exit data copyout(

#define GPU_ALLOC_ON_DEVICE !$acc enter data create(
#define GPU_DELETE_FROM_DEVICE !$acc exit data delete(

#define GPU_LOOP_ALL_LEVELS !$acc parallel loop
#define GPU_END_LOOP_ALL_LEVELS !$acc end parallel loop

#define GPU_LOOP_LEVEL_1 !$acc parallel loop gang
#define GPU_END_LOOP_LEVEL_1 !$acc end parallel loop

#define GPU_LOOP_LEVEL_2 !$acc loop worker vector
#define GPU_END_LOOP_LEVEL_2

#endif
#endif
```

SOLEDGE3X domain decomposition

- 3 level domain decomposition:
  1. Structured zones for magnetic topology
  2. **MPI blocks: prioritized by flux surface** across zones
     - If $N_{MPI} \leq N_{FS}$ each MPI process in charge of a set of FS
     - If $N_{MPI} > N_{FS}$ largest flux surfaces will be shared by a team of MPI processes
  3. **Thread chunks**: no direction priority, aiming at load balance between chunks
     - OpenMP loops are on chunks and species, not on mesh points inside chunks

```
!$OMP DO SCHEDULE(RUNTIME) COLLAPSE(2)
do ichunk = 1, split%Nchunks
   do ispec = 0, Nspecies
      do ipsi = ipsiminWG, ipsimaxWG
         do itheta = ithetaminWG, ithetamaxWG
            do iphi = iphiminWG, iphimaxWG
```

GPU porting

```
GPU_LOOP_ALL_LEVELS collapse(3)
do ipsi = ipsiminWG, ipsimaxWG
   do itheta = ithetaminWG, ithetamaxWG
      do iphi = iphiminWG, iphimaxWG
    …some work
      end do !iphi
   end do !itheta
end do !ipsi
GPU_END_LOOP_ALL_LEVELS
```

MPI domain decomposition evolution

- New configurable MPI domain decomposition

Before / default

MPI blocks: flux-surface prioritized across zones

- Existing behavior preserved when the keyword is not set
- One chunk cannot manage several zones
- Several zones per MPI rank therefore imply several chunks

New option

`zonePrioritizedDecomposition`

- .FALSE. → flux-surface-prioritized decomposition
- .TRUE. → zone-prioritized decomposition
- **Available for both CPU and GPU runs**
- Requires PETSc for all implicit solvers

Why it matters for GPU production computing

GPU execution: one kernel is associated with one chunk. If an MPI rank needs several chunks because it spans several zones, the same kernel pattern is launched several times, which is not ideal for GPU efficiency.

Takeaway: the branch keeps the default CPU/GPU behavior, while enabling a zone-prioritized path when the zone layout is the better decomposition driver.

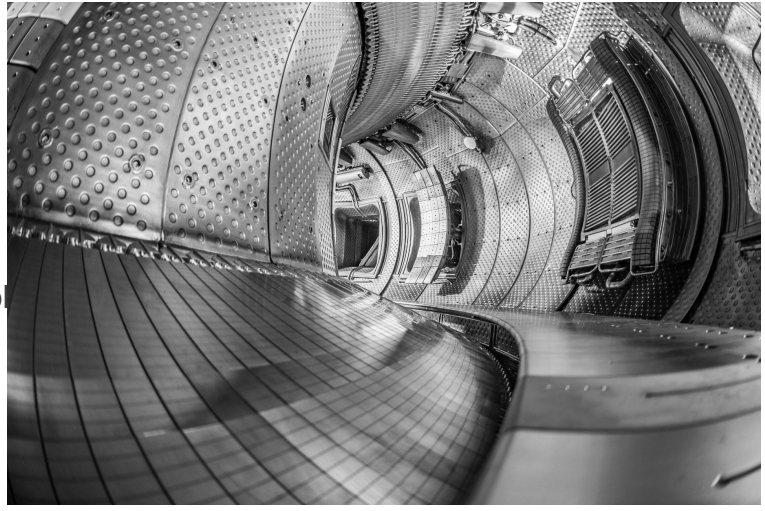
WEST Production test on GPUs

Physical simulation setup

- **Full-size, full-field simulation** with realistic wall geometry representative of a typical **WEST plasma**
- **Plasma parameters:** $I_p = 500$ kA, $B_{0r} = 3.8$ T
- **Target SOL power:** $P_{SOL} = 2$ MW
- **Separatrix density:** $n_{e,sep} = 1.6 \times 10^{19}$ m^{-3}
- **Mesh resolution:** cell width set to one ion-sound Larmor radius at the OMP, approximately 0.2 mm
- **Density and power sources** prescribed as boundary conditions in the inner region
- **Neutrals:** fluid model
- **Turbulence:** still developing, but sufficiently strong to sustain the current radial profiles
- **Plasma and neutral density recycling coefficients:** 0.99

HPC setup

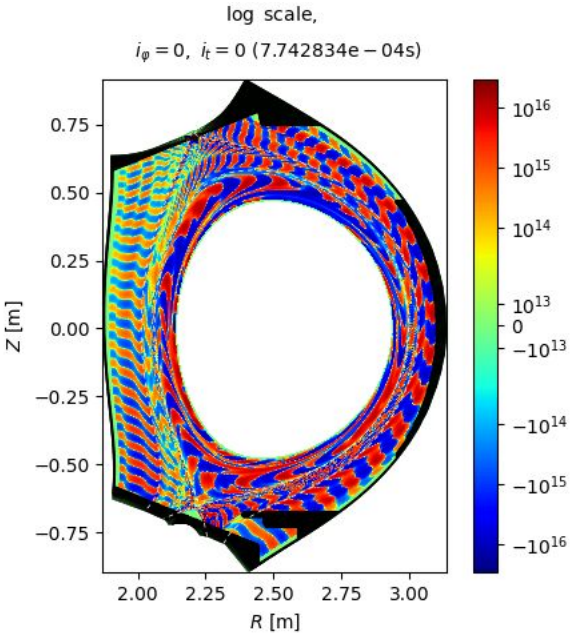
- Pitagora on 8 nodes (32 GPUs)
- Nvhpc compiler
- Petsc
- development for petsc pc/sol each implicit solver
  → identify the good one for each with the miniapp

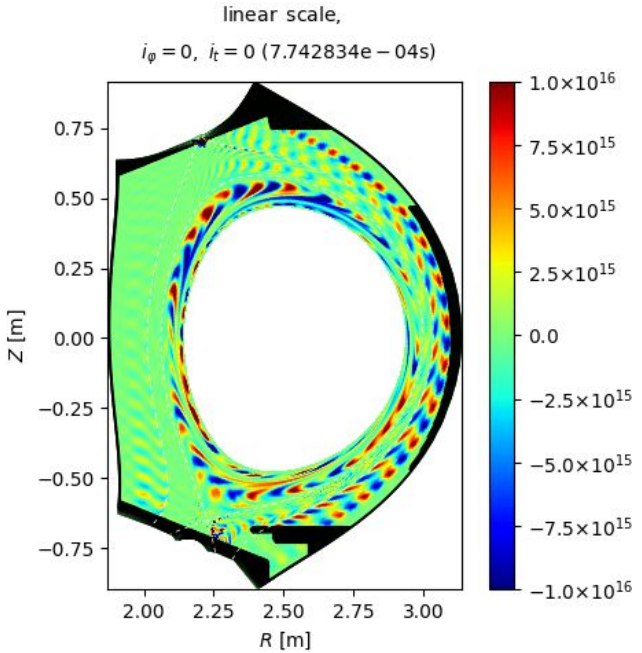


Some profiles

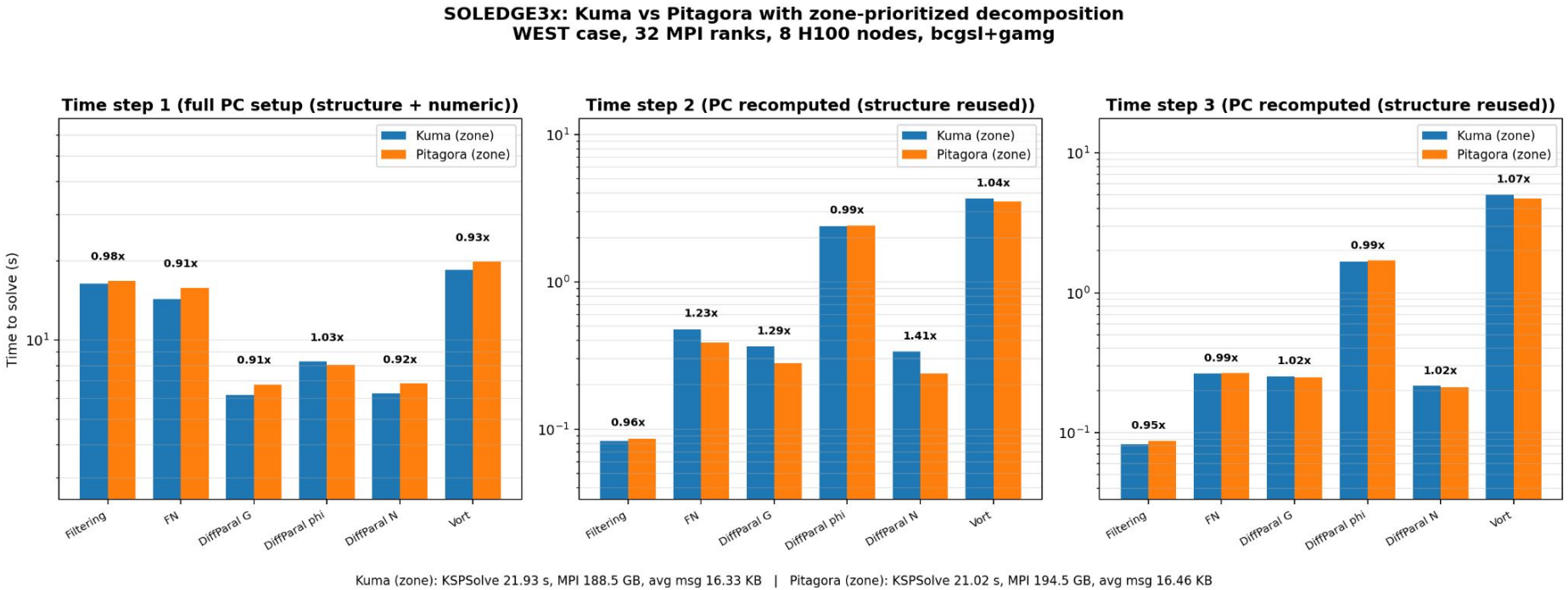
- The simulation reached 1 ms of physical time and took 120 hours on 8 nodes (32 GPUs).

  → Relative fluctuations of density [m^{-3}]

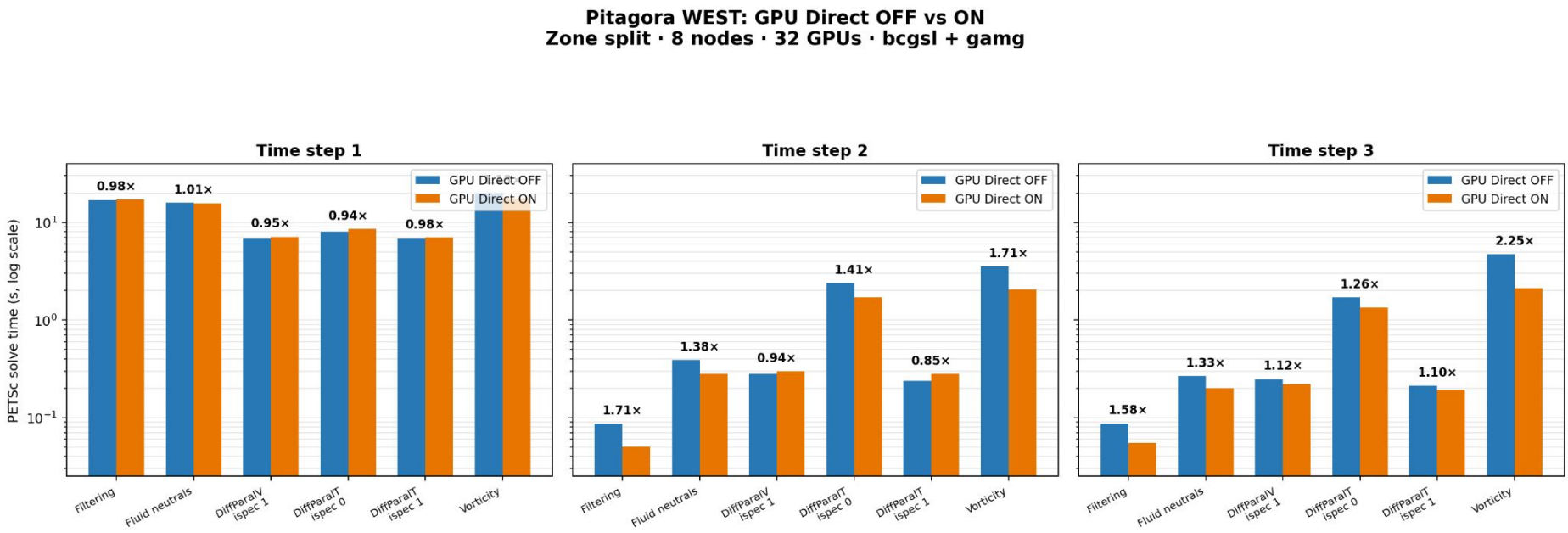




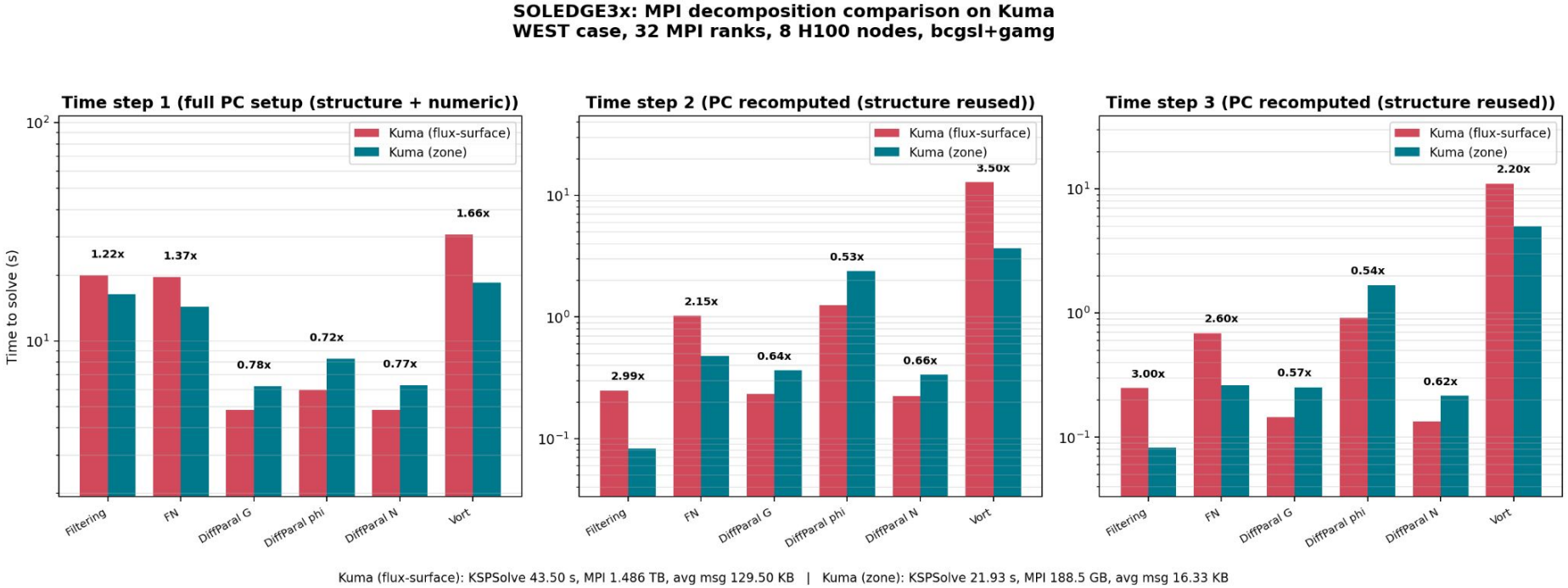
WEST Production test



WEST Production test

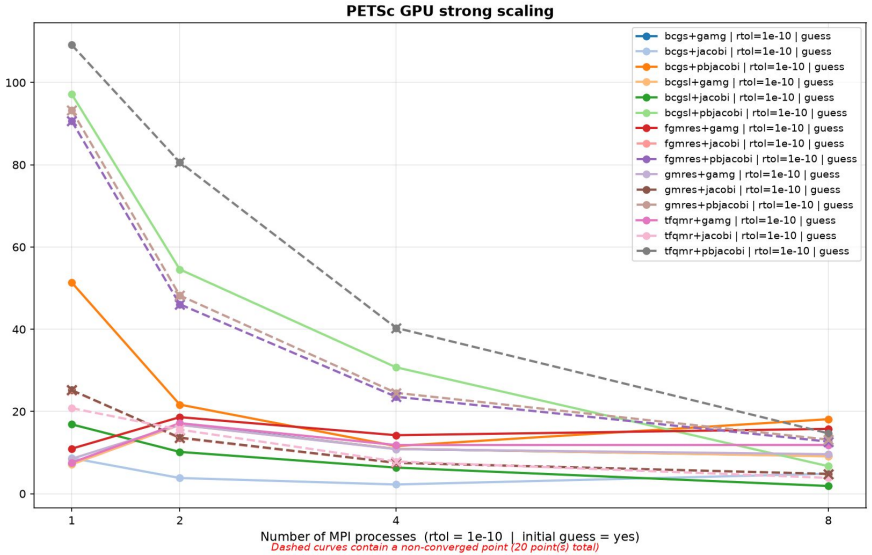


WEST Production test



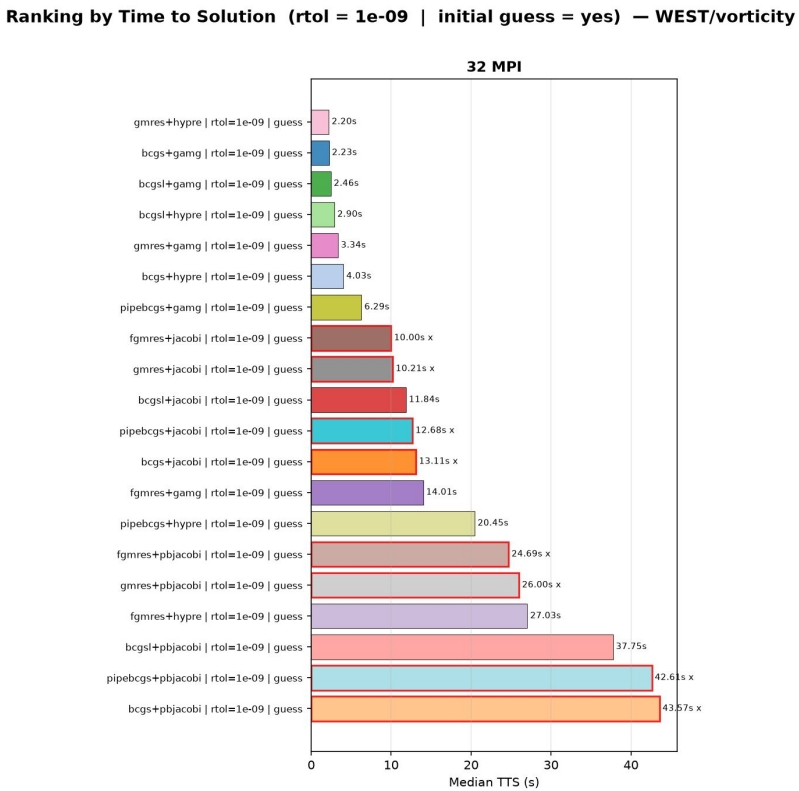
Miniapp for Linear Solver

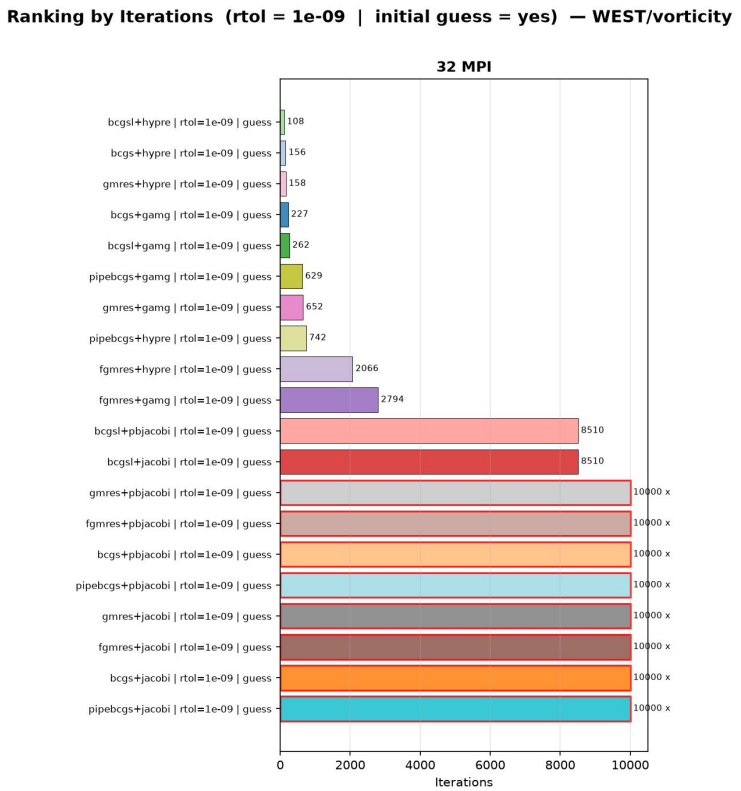
- Build a Miniapp to optimize Implicit Solvers:
  - Lightweight Python/petsc4py benchmark for evaluating PETSc linear solvers on HPC systems
  - Supports both CPU and NVIDIA GPU execution, using CUDA and PETSc AIJcuSPARSE matrices
  - Tests combinations of KSP solvers and preconditioners from a simple JSON configuration
  - Supports MPI scaling across multiple processes/GPUs and nodes
  - Uses realistic PETSc inputs: matrix, RHS, initial guess, and reference solution
  - https://github.com/peyberne/petsc-python-miniapp
  - Measures time-to-solution, including separate setup and preconditioner-reuse timings
  - Generates scaling results for comparing solver configurations and MPI/GPU counts
  - Designed to reproduce SOLEDGE3x-like steady-state solver behavior and identify suitable PETSc configurations



Miniapp for Linear Solver

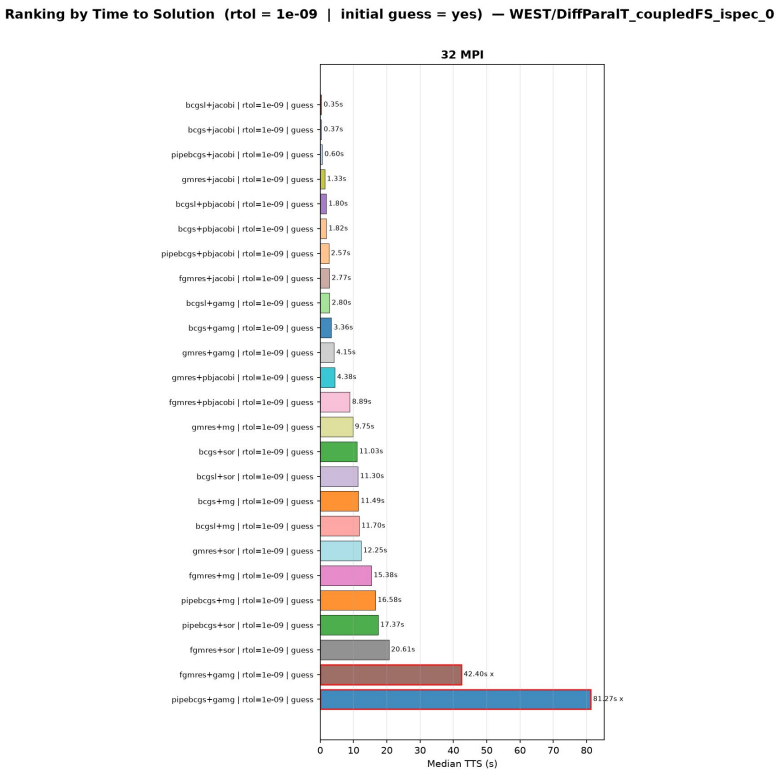
- Vorticity Implicit Solver:

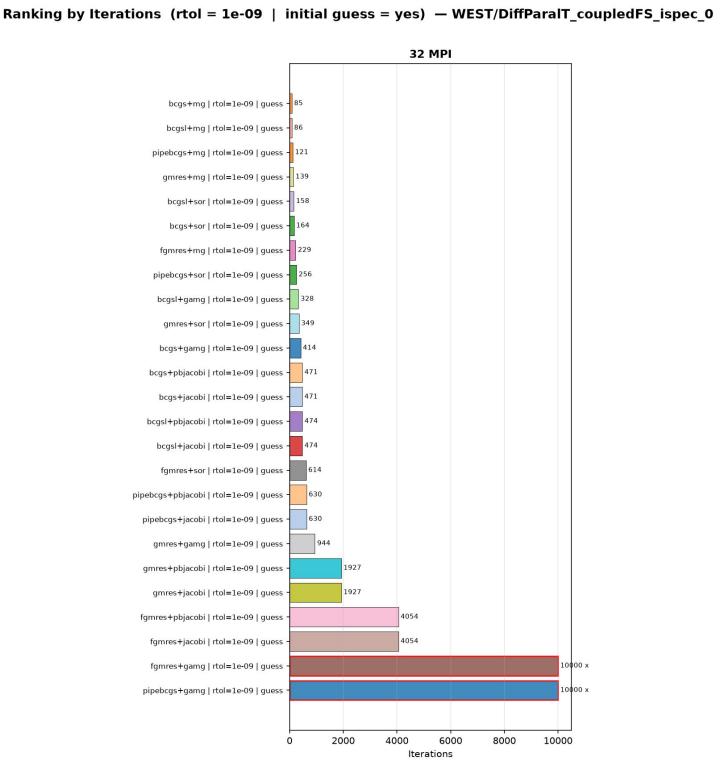




Miniapp for Linear Solver

- DiffParalT Implicit Solver:





NVTX Instrumentation in Fortran

- Fortran bindings
- Integration strategy
- Benefits for analysis

```
module nvtx_mod

  use iso_c_binding
  implicit none

  ! NVTX color is ARGB: 0xAARRGGBB (alpha must be FF for opaque colors)
  integer, private :: col(7) = [ int(Z'FF00FF00',8), int(Z'FF0000FF',8), int(Z'FFFFFF00',8), &
                                 int(Z'FFFF00FF',8), int(Z'FF00FFFF',8), int(Z'FFFF0000',8), int(Z'FFFFFFFF',8) ]

  ! C char buffer for the message (null-terminated). Must have TARGET for c_loc().
  character(kind=C_CHAR), private, target :: tempName(256)

  integer, parameter :: invtx_ImplicitDensities             = 1
  integer, parameter :: invtx_ImplicitMomentum              = 2
….
  integer, parameter :: invtx_weno                          = 13

  ! NVTX event attributes struct (NVTX v2-ish layout). Size must match the C struct.
  type, bind(C) :: nvtxEventAttributes
               integer(C_INT16_T) :: version = 1
               integer(C_INT16_T) :: size    = 48
….             = C_NULL_PTR
  end type nvtxEventAttributes

  interface
               integer(C_INT) function nvtxRangePushA(name) bind(C, name='nvtxRangePushA')
               use iso_c_binding
               character(kind=C_CHAR), dimension(*) :: name
               end function nvtxRangePushA

               integer(C_INT) function nvtxRangePushEx(event) bind(C, name='nvtxRangePushEx')
               use iso_c_binding
               import :: nvtxEventAttributes
               type(nvtxEventAttributes), intent(in) :: event
               end function nvtxRangePushEx

               integer(C_INT) function nvtxRangePop() bind(C, name='nvtxRangePop')
               use iso_c_binding
               end function nvtxRangePop
  end interface

contains

  subroutine nvtxStartRange(name, id)
               use iso_c_binding
               character(kind=C_CHAR, len=*), intent(in) :: name
               integer, optional, intent(in) :: id

               type(nvtxEventAttributes) :: event
…..
  end subroutine nvtxStartRange


  subroutine nvtxEndRange()
               use iso_c_binding
               integer(C_INT) :: rid
               rid = nvtxRangePop()
  end subroutine nvtxEndRange

end module nvtx_mod
```

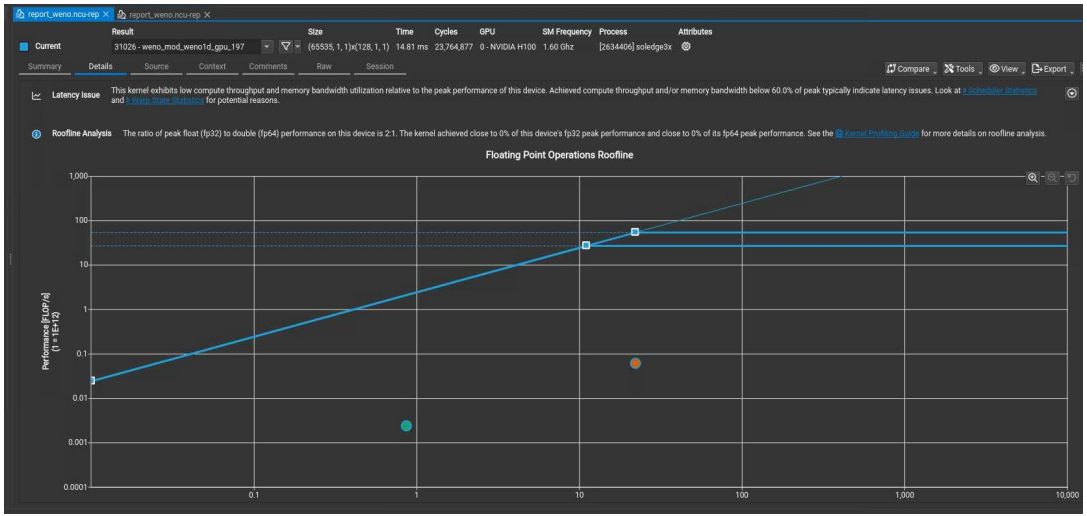
Nsys stats

- srun nsys profile --stats=true -t cuda,openacc
- *build_mat_loop* called by every implicit solver
- *weno* weirdly time consuming

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
27.9	1,165,205,839	23	50,661,123.4	35,985,738.0	34,912,745	69,515,157	16,309,527.4	linsys_buildmat_loop_petsc_coo_1753_gpu
16.4	684,103,244	57	12,001,811.3	12,101,649.0	11,784,336	12,118,192	148,570.5	weno_mod_weno1d_gpu_197_gpu
12.0	501,134,901	648	773,356.3	100,368.0	5,984	4,350,149	1,277,420.6	void cub::CUB_200101_500_520_600_610_700_750_800_860_890_900_NS::DeviceRadixSortOnesweepKernel<cub:…
3.8	159,825,657	3,973	40,228.0	5,024.0	2,464	257,344	73,904.1	void cusparse::csrmv_v3_kernel<std::integral_constant<bool, (bool)0>, int, int, double, double, dou…
3.6	152,485,705	4	38,121,426.3	38,198,739.0	37,712,208	38,376,019	293,380.0	vorticity_getglobalchunkstencil_vorticity_getglobalchunkstencil_petsc_1782_gpu
3.4	140,923,351	8	17,615,418.9	17,628,310.5	17,358,549	17,923,190	156,265.6	diffparalt_getglobalchunkstencil_diffparalt_getglobalchunkstencil_petsc_446_gpu
2.3	97,915,691	2	48,957,845.5	48,957,845.5	48,794,885	49,120,806	230,460.9	vorticity_mod_computejpoladt_138_gpu
2.3	97,314,920	2	48,657,460.0	48,657,460.0	48,631,940	48,682,980	36,090.7	vorticity_mod_computejpoladt_90_gpu
2.3	94,718,885	2	47,359,442.5	47,359,442.5	47,084,771	47,634,114	388,444.2	vorticity_mod_computejpoladt_187_gpu
1.8	74,201,926	4	18,550,481.5	18,535,817.5	18,453,113	18,677,178	99,756.9	fn_getglobalchunkstencil_fn_getglobalchunkstencil_petsc_379_gpu
1.5	61,343,598	4	15,335,899.5	15,358,994.0	15,173,685	15,451,925	132,822.5	diffparalv_getglobalchunkstencil_diffparalv_getglobalchunkstencil_petsc_418_gpu
1.4	57,717,767	3	19,239,255.7	19,249,209.0	19,132,279	19,336,279	102,363.6	filteringphi_getglobalchunkstencil_filteringphi_getglobalchunkstencil_petsc_343_gpu
1.2	49,428,995	12	4,119,082.9	4,141,398.0	3,811,877	4,320,485	174,007.4	linsys_buildrhs_loop_1_gpu_1963_gpu
1.2	49,205,093	12	4,100,424.4	3,999,029.5	3,746,885	5,565,032	480,344.1	solver_class_mod_solveriterative_storesol_701_gpu
1.0	42,469,947	2	21,234,973.5	21,234,973.5	21,007,070	21,462,877	322,304.2	vorticity_getglobalchunkrhs_vorticity_getglobalchunkrhs_petsc_1402_gpu
0.9	37,420,666	178	210,228.5	9,136.0	2,720	4,804,007	658,464.0	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)8192, int, …
0.9	36,980,339	96	385,211.9	15,872.0	2,463	6,260,552	1,127,467.0	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)0, int, int…
0.7	30,395,497	2	15,197,748.5	15,197,748.5	15,193,941	15,201,556	5,384.6	plasmaevolve_mod_evolveimplicitp_72_gpu

NCU-Roofline

- *weno* weirdly time consuming




```
! Loop on the cells and performs interpolation
GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3) private(reconstm, reconstp, beta, alpham, alphap)
do ix = 1,Nx
do iy = 1,Ny
do iz=1+order-1,Nz-order+1
! Polynomial reconstructions at the faces of the cell
do r = 0, order-1
reconstm(r) = 0._dp
do j = 0, order-1
                reconstm(r) = reconstm(r) + crj(r,j)*var(iz-r+j, iy, ix)
enddo
enddo
do r = 0, order-1
reconstp(r) = 0._dp
do j = 0, order-1
                reconstp(r) = reconstp(r) + crj(r-1,j)*var(iz-r+j, iy, ix)
enddo
enddo
! Calculation of beta (smoothness) factors
SELECT CASE (order)
CASE (2)
beta(0) = (var(iz+1, iy, ix)-var(iz, iy, ix))**2
beta(1) = (var(iz, iy, ix)-var(iz-1, iy, ix))**2
CASE (3)
beta(0) = 13./12.*(var(iz, iy, ix)-2.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2 &
                +1./4.*(3.*var(iz, iy, ix)-4.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2
beta(1) = 13./12.*(var(iz-1, iy, ix)-2.*var(iz, iy, ix)+var(iz+1, iy, ix))**2 &
                +1./4.*(var(iz-1, iy, ix)-var(iz+1, iy, ix))**2
beta(2) = 13./12.*(var(iz-2, iy, ix)-2.*var(iz-1, iy, ix)+var(iz, iy, ix))**2 &
                +1./4.*(var(iz-2, iy, ix)-4.*var(iz-1, iy, ix)+3.*var(iz, iy, ix))**2
END SELECT
! Calculation of WENO coefficients
alphapsum = 0.0
alphamsum = 0.0
do r = 0, order-1
alphap(r) = dplus(r)/(epsilon+beta(r))**2
alpham(r) = dminus(r)/(epsilon+beta(r))**2
alphapsum = alphapsum + alphap(r)
alphamsum = alphamsum + alpham(r)
end do
ur(iz-1, iy, ix) = 0.0
ul(iz, iy, ix) = 0.0
do r = 0, order-1
ur(iz-1, iy, ix) = ur(iz-1, iy, ix) + (alphap(r)/alphapsum)*reconstp(r)
ul(iz, iy, ix) = ul(iz, iy, ix) + (alpham(r)/alphamsum)*reconstm(r)
end do
! WENO reconstruction
enddo
end do
end do
GPU_END_PARALLEL_LOOP_ALL_LEVELS
```

Verbose compilation flag

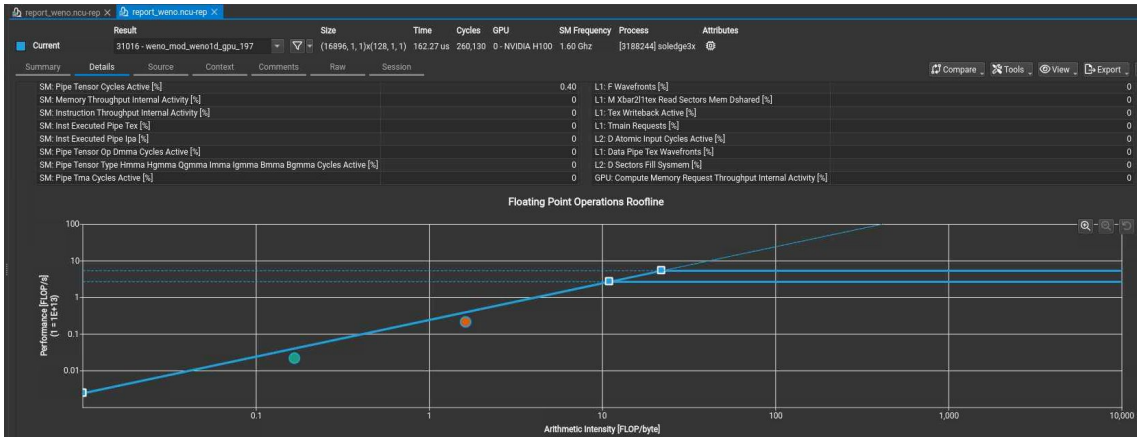
- *-Minfo=all*
- inner loops are parallelized by the compiler

```
197, Generating implicit firstprivate(order,nz,ny,nx)
     Generating NVIDIA GPU code
    198, !$acc loop gang collapse(3) ! blockidx%x
    199,   ! blockidx%x collapsed
    200,   ! blockidx%x collapsed
    202, !$acc loop vector(128) ! threadidx%x
    204, !$acc loop seq
    208, !$acc loop vector(128) ! threadidx%x
    210, !$acc loop seq
    230, !$acc loop vector(128) ! threadidx%x
         Generating implicit reduction(+:alphapsum,alphamsum)
    238, !$acc loop seq
197, Generating implicit copyout(ur(order-1:nz-order,:ny,:nx)) [if not already present]
     Generating implicit copyin(var(:,:ny,:nx),dminus(:order-1),crj(:order-1,:order-1))
     Generating implicit copyout(ul(order:nz-order+1,:ny,:nx)) [if not already present]
     Generating implicit copyin(dplus(:order-1)) [if not already present]
198, Loop not fused: no successor loop
     Loop not vectorized/parallelized: too deeply nested
199, Loop not vectorized/parallelized: too deeply nested
200, Generating implicit firstprivate(alphapsum,alphamsum)
     Invariant if transformation
     Loop not fused: no successor loop
     FMA (fused multiply-add) instruction(s) generated
202, Loop is parallelizable
     Generating implicit firstprivate(j)
     Loop not fused: enclosed in different number of nests
     2 loops fused
     Loop not fused: enclosed in different number of nests
     2 loops fused
     Loop not fused: enclosed in different number of nests
     2 loops fused
204, Complex loop carried dependence of reconstm prevents parallelization
     Loop carried reuse of reconstm prevents parallelization
     Loop not fused: possible intervening definition
     Generated vector simd code for the loop containing reductions
208, Loop is parallelizable
210, Complex loop carried dependence of reconstp prevents parallelization
     Loop carried reuse of reconstp prevents parallelization
     Generated vector simd code for the loop containing reductions
230, Loop is parallelizable
     Loop not fused: possible intervening definition
     Generated vector simd code for the loop containing reductions
238, Complex loop carried dependence of ur prevents parallelization
     Loop carried reuse of ur prevents parallelization
     Complex loop carried dependence of ul prevents parallelization
     Loop carried reuse of ul prevents parallelization
     Generated vector simd code for the loop containing reductions
```

```
196       ! Loop on the cells and performs interpolation
197       GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3) private(reconstm, reconstp, beta, alpham, alphap)
198       do ix = 1,Nx
199          do iy = 1,Ny
200             do iz=1+order-1,Nz-order+1
201                ! Polynomial reconstructions at the faces of the cell
202                do r = 0, order-1
203                   reconstm(r) = 0._dp
204                   do j = 0, order-1
205                      reconstm(r) = reconstm(r) + crj(r,j)*var(iz-r+j, iy, ix)
206                   enddo
207                enddo
208                do r = 0, order-1
209                   reconstp(r) = 0._dp
210                   do j = 0, order-1
211                      reconstp(r) = reconstp(r) + crj(r-1,j)*var(iz-r+j, iy, ix)
212                   enddo
213                enddo
214                ! Calculation of beta (smoothness) factors
215                SELECT CASE (order)
216                CASE (2)
217                   beta(0) = (var(iz+1, iy, ix)-var(iz, iy, ix))**2
218                   beta(1) = (var(iz, iy, ix)-var(iz-1, iy, ix))**2
219                CASE (3)
220                   beta(0) = 13./12.*(var(iz, iy, ix)-2.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2 &
221                        +1./4.*(3.*var(iz, iy, ix)-4.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2
222                   beta(1) = 13./12.*(var(iz-1, iy, ix)-2.*var(iz, iy, ix)+var(iz+1, iy, ix))**2 &
223                        +1./4.*(var(iz-1, iy, ix)-var(iz+1, iy, ix))**2
224                   beta(2) = 13./12.*(var(iz-2, iy, ix)-2.*var(iz-1, iy, ix)+var(iz, iy, ix))**2 &
225                        +1./4.*(var(iz-2, iy, ix)-4.*var(iz-1, iy, ix)+3.*var(iz, iy, ix))**2
226                END SELECT
227                ! Calculation of WENO coefficients
228                alphapsum = 0.0
229                alphamsum = 0.0
230                do r = 0, order-1
231                   alphap(r) = dplus(r)/(epsilon+beta(r))**2
232                   alpham(r) = dminus(r)/(epsilon+beta(r))**2
233                   alphapsum = alphapsum + alphap(r)
234                   alphamsum = alphamsum + alpham(r)
235                end do
236                ur(iz-1, iy, ix) = 0.0
237                ul(iz, iy, ix) = 0.0
238                do r = 0, order-1
239                   ur(iz-1, iy, ix) = ur(iz-1, iy, ix) + (alphap(r)/alphapsum)*reconstp(r)
240                   ul(iz, iy, ix) = ul(iz, iy, ix) + (alpham(r)/alphamsum)*reconstm(r)
241                end do
242             enddo
243          end do
244       end do
245       GPU_END_PARALLEL_LOOP_ALL_LEVELS
```

NCU-Roofline

- *weno* weirdly time consuming:
  - inner loops are parallelized by the compiler
  - use of:
    #define GPU_LOOP_SEQ !$acc loop seq




```
! Loop on the cells and performs interpolation
GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3) private(reconstm, reconstp, beta, alpham, alphap)
do ix = 1,Nx
do iy = 1,Ny
do iz=1+order-1,Nz-order+1
! Polynomial reconstructions at the faces of the cell
GPU_LOOP_SEQ
do r = 0, order-1
reconstm(r) = 0._dp
GPU_LOOP_SEQ
do j = 0, order-1
                reconstm(r) = reconstm(r) + crj(r,j)*var(iz-r+j, iy, ix)
enddo
enddo
GPU_LOOP_SEQ
do r = 0, order-1
reconstp(r) = 0._dp
GPU_LOOP_SEQ
do j = 0, order-1
                reconstp(r) = reconstp(r) + crj(r-1,j)*var(iz-r+j, iy, ix)
enddo
enddo
! Calculation of beta (smoothness) factors
SELECT CASE (order)
CASE (2)
beta(0) = (var(iz+1, iy, ix)-var(iz, iy, ix))**2
beta(1) = (var(iz, iy, ix)-var(iz-1, iy, ix))**2
CASE (3)
beta(0) = 13./12.*(var(iz, iy, ix)-2.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2 &
              +1./4.*(3.*var(iz, iy, ix)-4.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2
beta(1) = 13./12.*(var(iz-1, iy, ix)-2.*var(iz, iy, ix)+var(iz+1, iy, ix))**2 &
              +1./4.*(var(iz-1, iy, ix)-var(iz+1, iy, ix))**2
beta(2) = 13./12.*(var(iz-2, iy, ix)-2.*var(iz-1, iy, ix)+var(iz, iy, ix))**2 &
              +1./4.*(var(iz-2, iy, ix)-4.*var(iz-1, iy, ix)+3.*var(iz, iy, ix))**2
END SELECT
! Calculation of WENO coefficients
alphapsum = 0.0
alphamsum = 0.0
GPU_LOOP_SEQ
do r = 0, order-1
alphap(r) = dplus(r)/(epsilon+beta(r))**2
alpham(r) = dminus(r)/(epsilon+beta(r))**2
alphapsum = alphapsum + alphap(r)
alphamsum = alphamsum + alpham(r)
end do
ur(iz-1, iy, ix) = 0.0
ul(iz, iy, ix) = 0.0
GPU_LOOP_SEQ
do r = 0, order-1
ur(iz-1, iy, ix) = ur(iz-1, iy, ix) + (alphap(r)/alphapsum)*reconstp(r)
ul(iz, iy, ix) = ul(iz, iy, ix) + (alpham(r)/alphamsum)*reconstm(r)
end do
! WENO reconstruction
enddo
end do
end do
GPU_END_PARALLEL_LOOP_ALL_LEVELS
```

Nsys stats

- srun nsys profile --stats=true -t cuda,openacc

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
33.2	1,162,820,005	23	50,557,391.5	35,931,171.0	34,766,213	69,225,484	16,211,812.5	linsys_buildmat_loop_petsc_coo_1753_gpu
14.3	501,125,429	648	773,341.7	100,704.0	5,951	4,350,913	1,277,685.8	void cub::CUB_200101_500_520_600_610_700_750_800_860_890_900_NS::DeviceRadixSortOnesweepKernel<cub:…
4.6	159,909,078	3,973	40,248.9	5,024.0	2,464	257,024	73,944.0	void cusparse::csrmv_v3_kernel<std::integral_constant<bool, (bool)0>, int, int, double, double, dou…
4.4	153,640,586	4	38,410,146.5	38,366,970.5	38,241,607	38,665,038	180,797.6	vorticity_getglobalchunkstencil_vorticity_getglobalchunkstencil_petsc_1782_gpu
4.0	141,136,354	8	17,642,044.3	17,574,292.5	17,438,150	17,950,946	181,514.3	diffparalt_getglobalchunkstencil_diffparalt_getglobalchunkstencil_petsc_446_gpu
2.8	97,649,667	2	48,824,833.5	48,824,833.5	48,742,577	48,907,090	116,328.3	vorticity_mod_computejpoladt_138_gpu
2.7	95,569,954	2	47,784,977.0	47,784,977.0	46,614,129	48,955,825	1,655,829.1	vorticity_mod_computejpoladt_90_gpu
2.7	94,845,251	2	47,422,625.5	47,422,625.5	47,372,593	47,472,658	70,756.6	vorticity_mod_computejpoladt_187_gpu
2.1	75,004,630	4	18,751,157.5	18,779,172.5	18,412,551	19,033,734	258,109.4	fn_getglobalchunkstencil_fn_getglobalchunkstencil_petsc_379_gpu
1.8	61,759,660	4	15,439,915.0	15,425,715.0	15,324,449	15,583,781	124,036.8	diffparalv_getglobalchunkstencil_diffparalv_getglobalchunkstencil_petsc_418_gpu
1.6	56,761,195	3	18,920,398.3	19,149,507.0	18,391,683	19,220,005	459,235.7	filteringphi_getglobalchunkstencil_filteringphi_getglobalchunkstencil_petsc_343_gpu
1.4	49,188,431	12	4,099,035.9	4,117,217.0	3,738,913	4,261,026	138,815.9	solver_class_mod_solveriterative_storesol_701_gpu
1.4	49,051,475	12	4,087,622.9	4,103,377.0	3,759,618	4,339,874	160,733.4	linsys_buildrhs_loop_1_gpu_1963_gpu
1.2	42,034,190	2	21,017,095.0	21,017,095.0	20,932,264	21,101,926	119,969.2	vorticity_getglobalchunkrhs_vorticity_getglobalchunkrhs_petsc_1402_gpu
1.1	37,426,758	178	210,262.7	9,344.0	2,751	4,805,282	658,524.7	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)8192, int, …
1.1	36,979,024	96	385,198.2	15,760.0	2,432	6,260,642	1,127,506.7	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)0, int, int…

Conclusion and work in progress

- Compare CPU and GPU results at the same time step, using 8 nodes of the Pitagora GPU partition and 8 nodes of its CPU partition
  - **GPU execution** is approximately **5x faster** than CPU execution
- Optimize implicit solvers for GPU execution
  → PETSc GAMG: tune aggregation and smoothing to better handle strong anisotropic couplings
- Port the collision operator to GPU
- Explore hypre for multithreading and GPU acceleration
- Use NVIDIA TorchFort to investigate **surrogate-model-based** matrix preconditioning

Collision module

- Collision - GPU version was not GPU-friendly
  - Implemented as a large monolithic kernel
  - Call of Lapack for each GPU thread
  - Matrix dimension depending on the #species
  - Many local scratch arrays
  - Register spilling:
    - GPU stack lives in registers;
    - when not enough registers are available,

    their content is said to spill into a special scratch memory –
    a perthread addressable region of main memory
    ⇒ must be allocated and freed at the kernel execution boundary
    ⇒ thus increasing even more the kernel duration

```
!$acc parallel loop gang worker collapse(3)
do isi= ipsimin, ipsimax
   do itheta = ithetamin, ithetamax
      do iphi = iphimin, iphimax
         !$acc parallel loop vector collapse(2)
         do ispec2 = 1, Nspecies
            do ispec = 1, Nspecies
               … some work on specirs …
         … some work building matrix M …
         IM = LAPACKInvert(M)
         … some work on IM …
      end do
   end do
end do
```

Collision module

- Collision - new version more GPU-friendly (and CPU-friendly!)
  - Decomposed into multiple smaller and independent kernels
  - Uses batched data structures for improved throughput
  - Employs LAPACK batched routines for matrix inversion

```
do ipsi_start = ipsiminWG, ipsimaxWG, blockSizePsi
   do itheta_start = ithetaminWG, ithetamaxWG, blockSizeTheta
      do iphi_start = iphiminWG, iphimaxWG, blockSizePhi
         ! End of block index
            ipsi_end   = min(ipsi_start  +blockSizePsi  -1, ipsimaxWG  )
            itheta_end = min(itheta_start+blockSizeTheta-1, ithetamaxWG)
            iphi_end   = min(iphi_start  +blockSizePhi  -1, iphimaxWG  )
            Npts = (ipsi_end-ipsi_start+1)*(itheta_end-itheta_start+1)*(iphi_end-iphi_start+1)
            GPU_PARALLEL_LOOP_ALL_LEVELS COLLAPSE(4)
            do ispec = 0, Nspecies
               do ipsi = ipsi_start, ipsi_end
                  do itheta = itheta_start, itheta_end
                     do iphi = iphi_start, iphi_end
                        ipt = ((ipsi - ipsi_start) * ntheta + (itheta - itheta_start)) * nphi + (iphi - iphi_start) + 1
            GPU_PARALLEL_LOOP_ALL_LEVELS COLLAPSE(3)
            do ispec2 = 1, nspec
              do ispec = 1, nspec
               do ipt = 1, Npts
                  conductivities_out(ipt,ispec,ispec2) = ...

            IM_batched = LAPACKInvert_batched(M_batched,Npts)
```

Boundary Conditions and MPI

- Generic structure `TFieldsSet` for physical fields
  - list of fields in a `TFieldsSet` structure
- Get flattened structure
- Make a generic communication routine based on the generic fields structure
- use of GPU direct

```
    do ispec = 0, Nspec
       fieldsLoc%spec(ispec)%n => fieldsSet%sets(ispec)%fields(ifield)%data
       ifield = ifield + 1
       fieldsLoc%spec(ispec)%G => fieldsSet%sets(ispec)%fields(ifield)%data
       ifield = ifield + 1
       fieldsLoc%spec(ispec)%T => fieldsSet%sets(ispec)%fields(ifield)%data
       GPU_ATTACH_PTR fieldsLoc%spec(ispec)%n)
       GPU_ATTACH_PTR fieldsLoc%spec(ispec)%G)
       GPU_ATTACH_PTR fieldsLoc%spec(ispec)%T)
```

```
allocate(sndBuff(Nbuff))
allocate(rcvBuff(Nbuff))

GPU_ALLOC_ON_DEVICE sndBuff )
GPU_ALLOC_ON_DEVICE rcvBuff )
do ifield= 1, flatFilelds%Nfields
     GPU_ATTACH_PTR flatFields%fields(ifield)%data)
end do

GPU_PARALLEL_LOOP_ALL_LEVELS
do ibuff = 1, Nsnd_loc
   do ifield = 1, NfieldsCnt
                sndBuff((ibuff-1)*NfieldsCnt+ifield) = flatFields%fields(ifieldsCnt(ifield))%data(iphi,itheta,ipsi)
   enddo
enddo ! ibuff
GPU_END_PARALLEL_LOOP_ALL_LEVELS

GPU_USE_DEVICE_POINTER sndBuff, rcvBuff )
call MPI_ALLTOALLV(sndBuff, sndcounts, snddispls, MPI_DOUBLE_PRECISION,  &
    rcvBuff, rcvcounts, rcvdispls, MPI_DOUBLE_PRECISION, MPIcomm, IERR)
GPU_END_USE_DEVICE_POINTER
```

Diagnostics

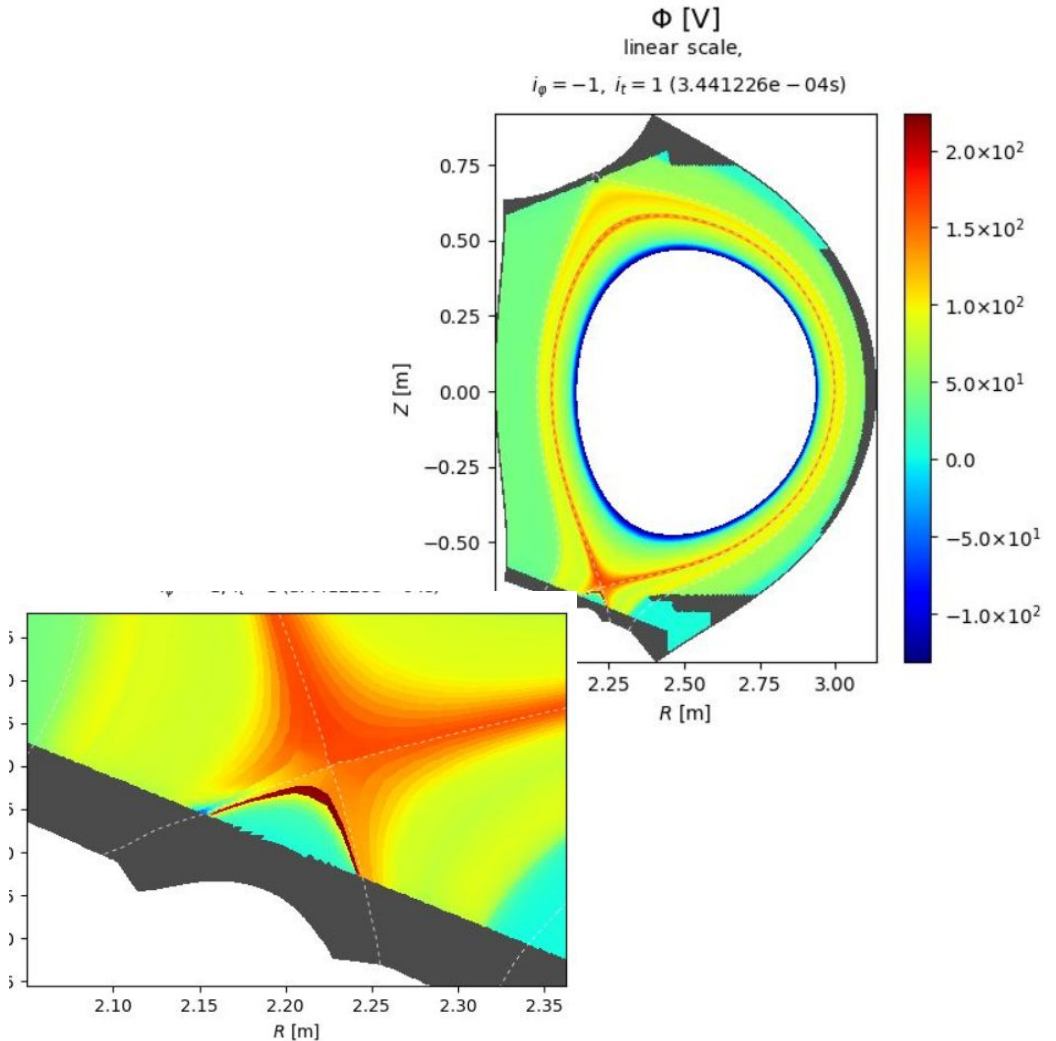
- generic structure TFieldsSet for fields in the frame of a domain
  - list of fields in a TFieldsSet structure
- get flatten structure the fields
- make a generic communication routine based on the generic fields structure.

```
    do ispec = 0, Nspec
       fieldsLoc%spec(ispec)% n => fieldsSet%sets(ispec)%fields(ifield)%data
       ifield = ifield + 1
       fieldsLoc%spec(ispec)% G => fieldsSet%sets(ispec)%fields(ifield)%data
       ifield = ifield + 1
       fieldsLoc%spec(ispec)% T => fieldsSet%sets(ispec)%fields(ifield)%data
       GPU_ATTACH_PTR  fieldsLoc%spec(ispec)%n)
       GPU_ATTACH_PTR  fieldsLoc%spec(ispec)%G)
       GPU_ATTACH_PTR  fieldsLoc%spec(ispec)%T)
```

```
call diagFields_computeStats(self%splitPsiPhiPlane, self%fieldsPsiPhiPlaneFlat,
self%fieldsStatsPsiPhiPlaneFlat, self%timeStatsAgregatePhi, NPP)
```

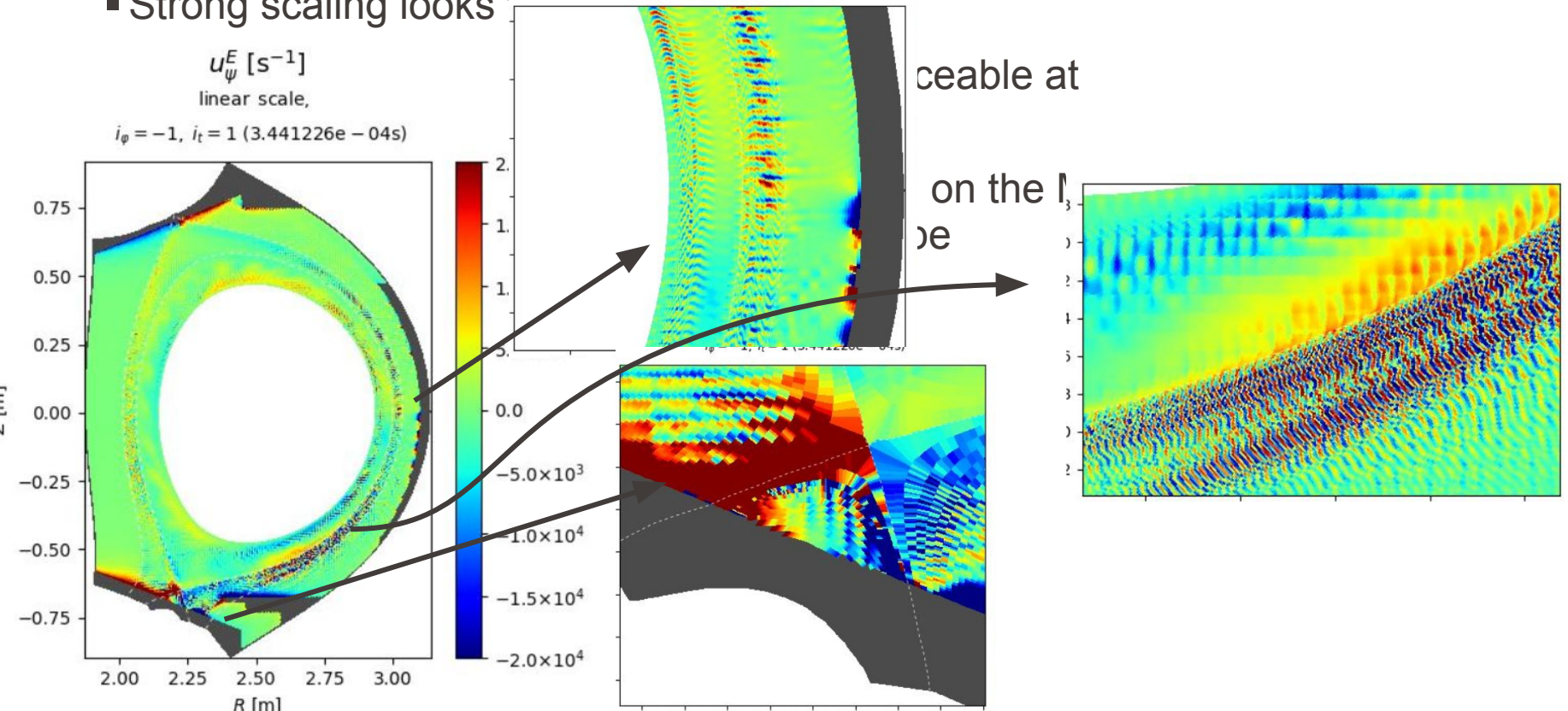
Some profiles

- The plasma potential in the box also doesnt look bad, but an weirdly high values is appearing in the PFR
- I saw the same when running the 3D simulations with ripples
- No RE used, but toroidal average Ti used – switching off



Some indication of resolution problems

- Strong scaling looks



ceable at

on the M

pe