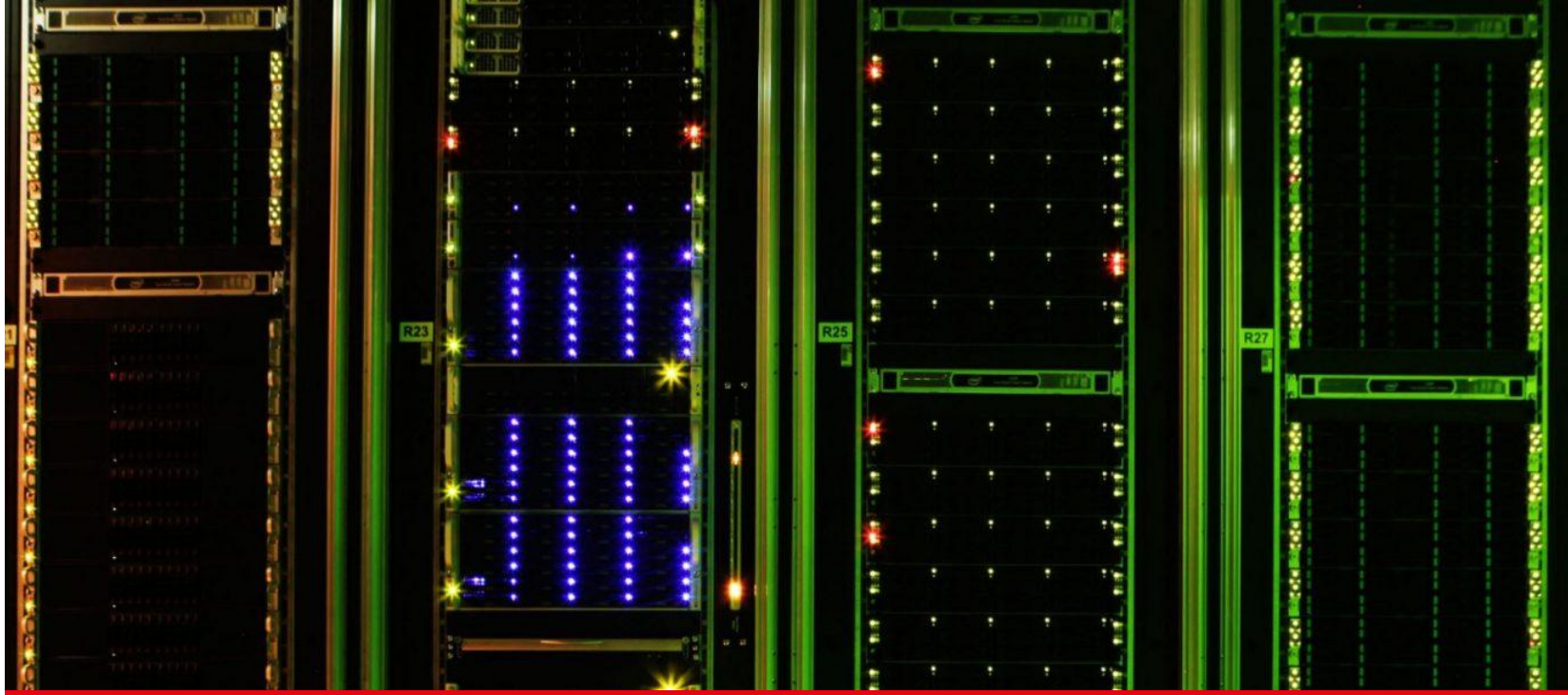


SOLEDGE3X GPU Porting

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

H. Bufferand, P. Tamain (Soledge group - CEA)

3rd Annual meeting of EUROfusion HPC ACHs – November 2025

SWISS PLASMA CENTER

EPFL SCITAS
Scientific IT and Application Support

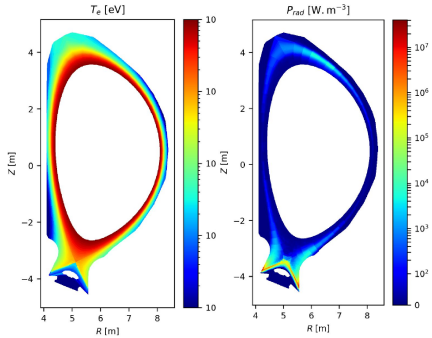
SOLEDGE3X: a versatile fluid code for the edge plasma

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

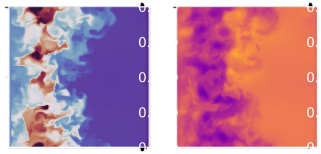
- Key features:
  - **Neutrals** either fluid (embedded) or kinetic (EIRENE)
  - Complete plasma **geometrical flexibility** (arbitrary number of X-points)
  - Usable in **2D or 3D**
  - Usable as **mean-field or self-consistent turbulence** code

- 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

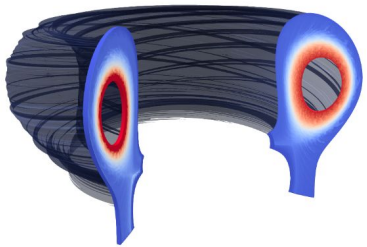
2D mean-field



2D turbulence



3D turbulence



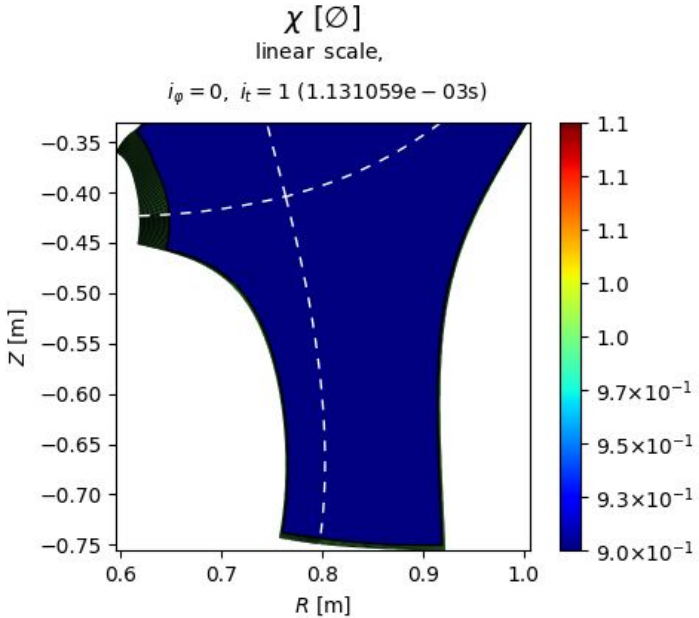
SOLEDGE3X domain decomposition

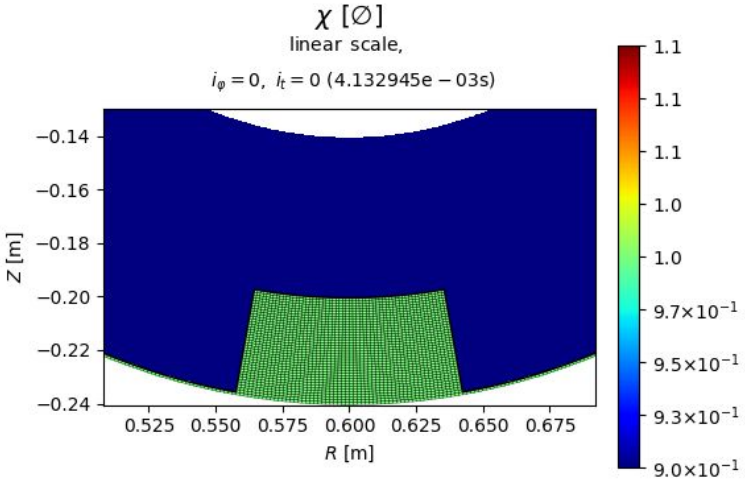
- 3 level domain decomposition:
  1. Structured zones for magnetic topology
  2. **MPI blocks: prioritized by flux surface** across zones
     - If $N_{MPI} \leq N_{FS}$ each MPI process in charge of a set of FS
     - If $N_{MPI} > N_{FS}$ largest flux surfaces will be shared by a team of MPI processes
  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
```

SOLEDGE3X boundary conditions

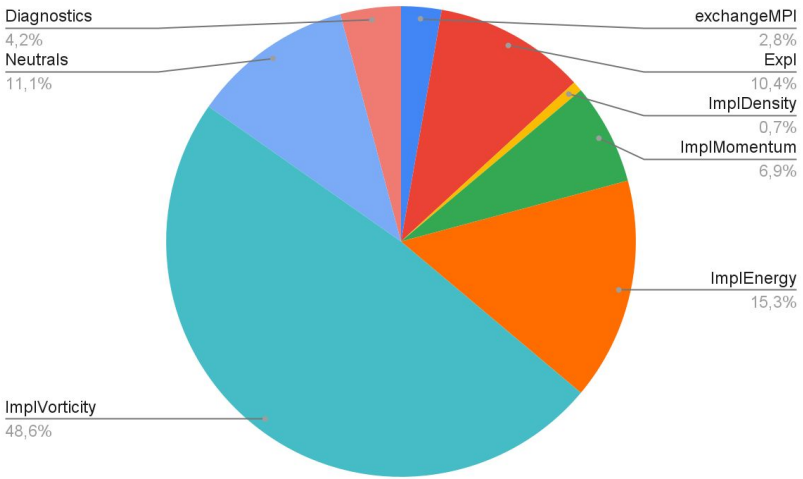
- Wall shape defined through mask function χ
- 2 types of boundary conditions:
  - Edges of wall mask => imposed through fluxes for most of them
  - Edges of simulation domain, typically one coes as wall should surround the plasma





CPU Version: Profiling

- CPU profiling



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
do ispec=1,Nspecies
   melt(specElt(ispec))=SpecMass(ispec)
end do
GPU_END_LOOP_ALL_LEVELS


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 :
  - **Parallel viscosity** terms
  - **Parallel heat conduction** terms

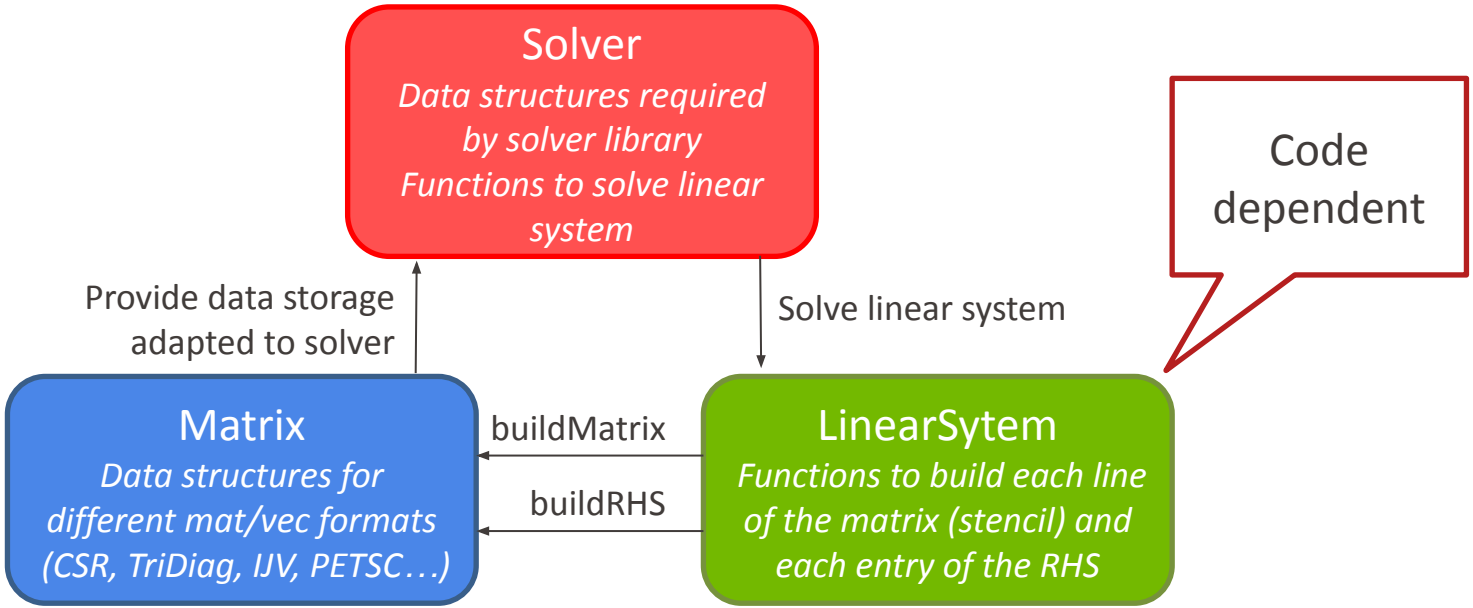
  } 2D

  - **Vorticity** equation
  - (optional) **fluid neutrals**
  - (optional) **potential filter**
  - **and more for EM …**

  } 3D

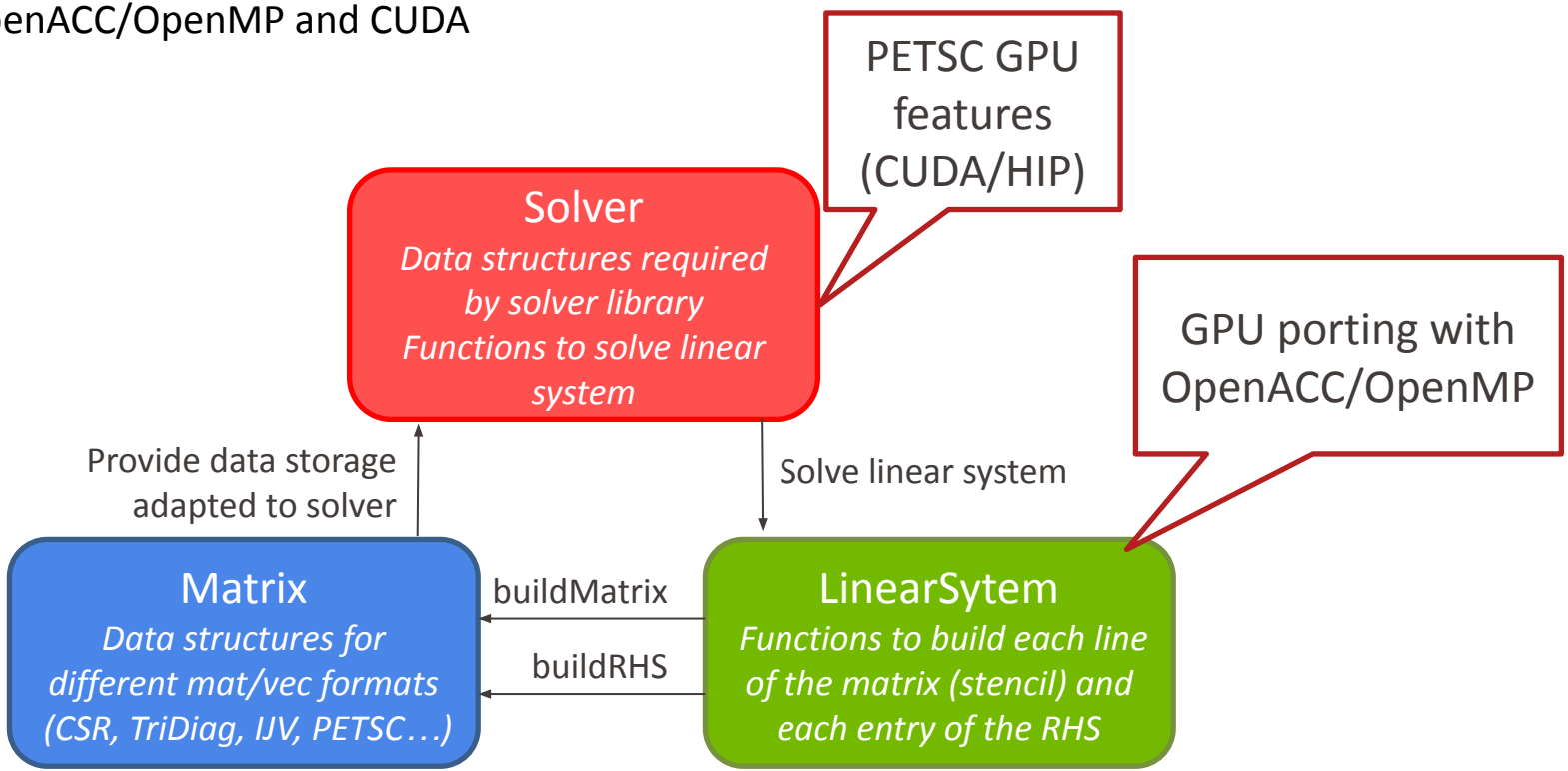
Implicit solvers

- Implicit solvers
  - solvers management based on 3 Fortran classes



Implicit solvers

- Mix OpenACC/OpenMP and CUDA



LinearSystem on GPU

- PETSC matrix filling is now done with PETSC ***MatSetValuesCOO*** instead of *MatSetValues* allowing to fill matrix with arrays instead of scalars, more efficient for GPU (and CPU too …)

```
      do ipsi = ipsimin, ipsimax
         do itheta = ithetamin, ithetamax
            do iphi = iphimin, iphimax
               do ifield = 1, self%NdofPerPoint
                  ! Get local row index and carry on only if it is non-zero
                  ! (otherwise means that this point is not part of the linear system, e.g. mask points)
                  irowLoc = self%getMatLocalIndex(ichunk,ipsi,itheta,iphi,ifield)
                  if (irowLoc.GE.1) then
                     ! Get stencil of local line of matrix
                     call  self%getStencil (ichunk, ipsi, itheta, iphi, ifield, &
                          stencSize, stencIpsi, stencItheta, stencIphi, stencIfield, stencVal)
                     irowGlobList(1) = self%getMatGlobalIndex(ichunk,ipsi,itheta,iphi,ifield) - 1 ! PETSC
indexing is from 0
                     do istencil = 1, stencSize
                        icolGlobList(istencil) = self%getMatGlobalIndex(ichunk, &
                             stencIpsi(istencil), stencItheta(istencil), stencIphi(istencil),
stencIfield(istencil)) &
                             - 1 ! PETSC indexing is from 0
                     enddo ! istencil
                     call MatSetValues(mat_ptr%PETSCmat, 1, irowGlobList, stencSize, icolGlobList, &
                          stencVal, INSERT_VALUES, ierrPETSC)
                  endif ! irowLoc >= 1
               enddo ! ifield
            enddo ! iphi
         enddo ! itheta
      enddo ! ipsi
```

initial linSys_buildMat version

```
!!$omp target teams distribute parallel do simd collapse(3) use_device_ptr(p_vorticity_stencVal_coo)
!$acc parallel loop                          collapse(3)     deviceptr (p_vorticity_stencVal_coo)
        do ipsi = ipsimin, ipsimax
           do itheta = ithetamin, ithetamax
              do iphi = iphimin, iphimax
                 do ifield = 1, self%NdofPerPoint
                    ! Get local row index and carry on only if it is non-zero
                    ! (otherwise means that this point is not part of the linear system, e.g. mask points)
                    irowLoc = self%getMatLocalIndex(ichunk,ipsi,itheta,iphi,ifield)
                    if (irowLoc.GE.1) then
                       ! Get stencil of local line of matrix
                       call  self%getStencil (ichunk, ipsi, itheta, iphi, ifield, &
                            stencSize, stencIpsi, stencItheta, stencIphi, stencIfield, stencVal)
                       irowGlobList(1) = self%getMatGlobalIndex(ichunk,ipsi,itheta,iphi,ifield) - 1 ! PETSC
indexing is from 0
                       do istencil = 1, stencSize
                          icolGlobList(istencil) = self%getMatGlobalIndex(ichunk, &
                               stencIpsi(istencil), stencItheta(istencil), stencIphi(istencil),
stencIfield(istencil)) &
                               - 1 ! PETSC indexing is from 0

                       enddo
                       p_vorticity_stencVal_coo(cnt:cnt+stencsize-1) = stencVal(1:stencSize)
                       cnt = cnt + stencSize
                    endif ! irowLoc >= 1
                 enddo ! ifield
              enddo ! iphi
           enddo ! itheta
        enddo ! ipsi
!$acc end parallel loop
!!$omp end target teams distribute parallel do simd
      cnt = cnt - 1
      petsc_cnt = cnt
!$acc host_data use_device ( mat_ptr%p_stencVal_coo )
!!$omp is_device_ptr        ( mat_ptr%p_stencVal_coo )
      call MatSetValuesCOO(mat_ptr%PETSCmat,mat_ptr%p_stencVal_coo(1:cnt),INSERT_VALUES,ierrPETSC)
```

new linSys_buildMat version

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

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)
```

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)
```

Explicit solvers

- The explicit solvers can often be ported reasonably easily but some functions have additional constraints.
- *NGammaT3D*, computing fluxes for the advection, illustrates some of these constraints.
- CPU/GPU Transfers are handled “by hand”

Explicit solvers

NGammaT3D - Original CPU version

```
if (any(b1_ptr(:,:,:).NE.0._dp).OR.any(bEM1_ptr(:,:,:).NE.0._dp).OR.any(vperp1(:,:,:).NE.0._dp)) then
   do itheta = ithetamin, ithetamax
      do iphi = iphimin, iphimax
         call NGammaT1D(mass, n_ptr(iphi,itheta,:), G_ptr(iphi,itheta,:), T_ptr(iphi,itheta,:), vperp1(iphi,itheta,:), &
              b1_ptr(iphi,itheta,:), bEM1_ptr(iphi,itheta,:), J_ptr(iphi,itheta,:), fluxN_psi(iphi,itheta,:), fluxG_psi(iphi,itheta,:), fluxE_psi(iphi,itheta,:), chi_ptr(iphi,itheta,:))
      enddo
   enddo
endif
```

Loop calling 1D function (limits parallelism)

Non-contiguous slices passed to function

NGammaT1D - Original CPU version

```
! Recovers the size of the arrays
Nz = size(n)

! Allocate memory for temporary arrays
allocate(v(1:Nz))
allocate(nl(1:Nz), vl(1:Nz), Tl(1:Nz))
allocate(nr(1:Nz), vr(1:Nz), Tr(1:Nz))

! Computes the parallel velocity
v = G / n

! Proceed to the WENO extrapolation of all the fields
call weno1D(n,nl,nr,2,chi)
call weno1D(v,vl,vr,2,chi)
call weno1D(T,Tl,Tr,2,chi)

! Applies thresholds on the extrapolated fields to avoid non
! physical values (negative densities, temperatures...)
nl = max(nl,NthreshMin)
nr = max(nr,NthreshMin)
Tl = max(Tl,TthreshMin)
Tr = max(Tr,TthreshMin)
```

Local allocations in loops

Vectorised operations

```
 ! Loop on cell faces and apply the Marquina's scheme
do iz = 2, Nz-2
   call NGammaTMarquina(mass,nl(iz),vl(iz),Tl(iz),vperpl(iz),bl(iz),Jl(iz), &
        nr(iz),vr(iz),Tr(iz),vperpr(iz),br(iz),Jr(iz),FluxM)
   fluxN(iz+1) = fluxN(iz+1) + FluxM(1)
   fluxG(iz+1) = fluxG(iz+1) + FluxM(2)
   fluxE(iz+1) = fluxE(iz+1) + FluxM(3)
enddo

! Clears memory
deallocate(v)
deallocate(nl, vl, Tl)
deallocate(nr, vr, Tr)
```

Loops in function called from a loop

Explicit solvers

NGammaT3D - New GPU version

```
if (any(metric(ichunk)%b1(:,:,:).NE.0._dp).OR.any(metric(ichunk)%bEM1(:,:,:).NE.0._dp).OR.any(vperp1(:,:,:).NE.0._dp)) then
   call transpose_array_3d(fieldsLoc(ichunk)%spec(ispec)%n, n_ptr, 3, 1, 2, n_buffer)
   call transpose_array_3d(fieldsLoc(ichunk)%spec(ispec)%G, G_ptr, 3, 1, 2, G_buffer)
   call transpose_array_3d(fieldsLoc(ichunk)%spec(ispec)%T, T_ptr, 3, 1, 2, T_buffer)
   call transpose_array_3d(metric(ichunk)%b1, b_ptr, 3, 1, 2, b_buffer)
   call transpose_array_3d(metric(ichunk)%bEM1, bEM_ptr, 3, 1, 2, bEM_buffer)
   call transpose_array_3d(metric(ichunk)%J, J_ptr, 3, 1, 2, J_buffer)
   call transpose_array_3d(chi(ichunk)%ival, chi_ptr, 3, 1, 2, chi_buffer)
   call transpose_array_3d(fluxN_psi, fluxN_ptr, 3, 1, 2, fluxN_buffer)
   call transpose_array_3d(fluxG_psi, fluxG_ptr, 3, 1, 2, fluxG_buffer)
   call transpose_array_3d(fluxE_psi, fluxE_ptr, 3, 1, 2, fluxE_buffer)
   call transpose_array_3d(vperp1, vperp_ptr, 3, 1, 2, vperp_buffer)
   call NGammaT1D(mass, n_ptr(:,lboundZ:,lboundY:), G_ptr(:,lboundZ:,lboundY:), T_ptr(:,lboundZ:,lboundY:), vperp_ptr, &
        b_ptr(:,lboundZ:,lboundY:), bEM_ptr(:,lboundZ:,lboundY:), J_ptr(:,lboundZ:,lboundY:), &
        fluxN_ptr(:,lboundZ:,lboundY:), fluxG_ptr(:,lboundZ:,lboundY:), fluxE_ptr(:,lboundZ:,lboundY:), chi_ptr(:,lboundZ:,lboundY:), &
        NxWG, Nz, Ny)
   call transpose_array_3d(fluxN_ptr, fluxN_psi, 2, 3, 1)
   call transpose_array_3d(fluxG_ptr, fluxG_psi, 2, 3, 1)
   call transpose_array_3d(fluxE_ptr, fluxE_psi, 2, 3, 1)
endif
```

Slices transposed to contiguous layout

Loop calling 3D function to keep possible parallelism

Explicit solvers

NGammaT1D - New GPU version

```
allocate(v(1:Nz,1:Ny,1:Nx))
allocate(nl(1:Nz,1:Ny,1:Nx), vl(1:Nz,1:Ny,1:Nx), Tl(1:Nz,1:Ny,1:Nx))
allocate(nr(1:Nz,1:Ny,1:Nx), vr(1:Nz,1:Ny,1:Nx), Tr(1:Nz,1:Ny,1:Nx))

GPU_ALLOC_ON_DEVICE v, nl, vl, Tl, nv, vr, Tr)

! Computes the parallel velocity
GPU_PARALLEL_LOOP ALL LEVELS collapse(3)
do ix = 1,Nx
   do iy = 1,Ny
      do iz = 1, Nz
         v(iz,iy,ix) = G(iz,iy,ix) / n(iz,iy,ix)
      end do
   enddo
enddo
GPU_END_PARALLEL_LOOP_ALL_LEVELS

! Proceed to the WENO extrapolation of all the fields
call weno1D_gpu(n,nl,nr,2,chi,Nx,Ny,Nz)
call weno1D_gpu(v,vl,vr,2,chi,Nx,Ny,Nz)
call weno1D_gpu(T,Tl,Tr,2,chi,Nx,Ny,Nz)
```

Large allocations

Pass 3D objects to functions to preserve maximum parallelism

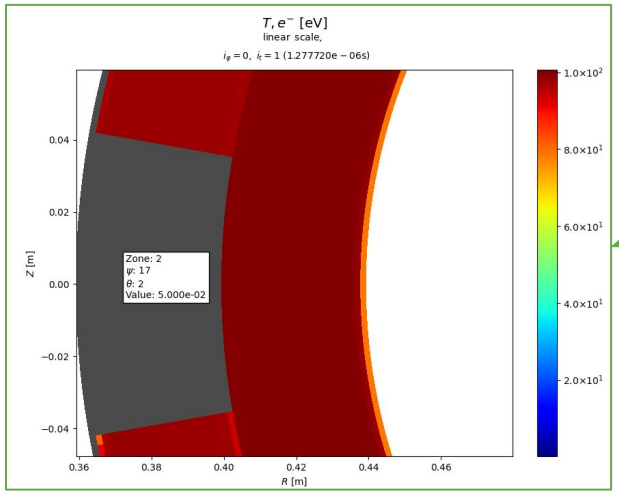
Unroll vectorised operations.
3D collapsed loops lead to maximum parallelism

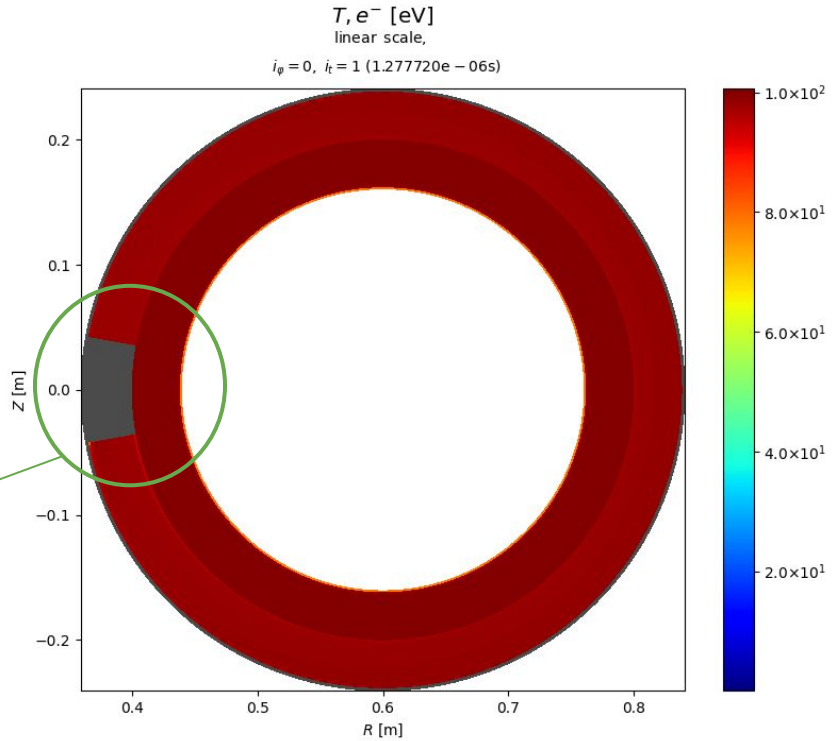
```
! Applies thresholds on the extrapolated fields to avoid non physical values (negative densities, temperatures...)
GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3)
do ix = 1,Nx
   do iy = 1,Ny
      do iz = 1, Nz
         nl(iz,iy,ix) = max(nl(iz,iy,ix),NthreshMin)
         nr(iz,iy,ix) = max(nr(iz,iy,ix),NthreshMin)
         Tl(iz,iy,ix) = max(Tl(iz,iy,ix),TthreshMin)
         Tr(iz,iy,ix) = max(Tr(iz,iy,ix),TthreshMin)
      end do
   enddo
enddo
GPU_END_PARALLEL_LOOP_ALL_LEVELS

! Loop on cell faces and apply the Marquina's scheme
GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3) private(FluxM)
do ix = 1,Nx
   do iy = 1,Ny
      do iz = 2, Nz-2
         call NGammaTMarquina1D(mass,nl(iz,iy,ix),vl(iz,iy,ix),Tl(iz,iy,ix),vperpl(iz,iy,ix),bl(iz,iy,ix),Jl(iz,iy,ix), &
              nr(iz,iy,ix),vr(iz,iy,ix),Tr(iz,iy,ix),vperpr(iz,iy,ix),br(iz,iy,ix),Jr(iz,iy,ix),FluxM)
         fluxN(iz+1,iy,ix) = fluxN(iz+1,iy,ix) + FluxM(1)
         fluxG(iz+1,iy,ix) = fluxG(iz+1,iy,ix) + FluxM(2)
         fluxE(iz+1,iy,ix) = fluxE(iz+1,iy,ix) + FluxM(3)
      enddo
   enddo
enddo
GPU_END_PARALLEL_LOOP_ALL_LEVELS

GPU_DELETE_FROM_DEVICE v, nl, vl, Tl, nv, vr, Tr)
! Clears memory
deallocate(v)
deallocate(nl, vl, Tl)
deallocate(nr, vr, Tr)
```

Function must be annotated to be called from GPU

Benchmark

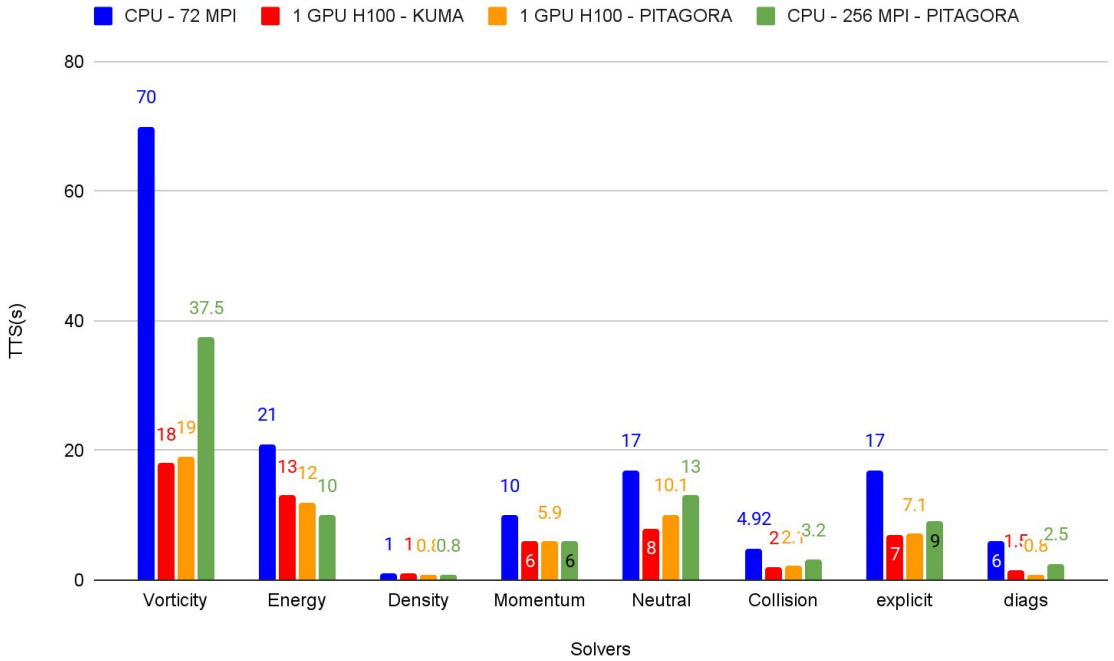
- Test case:
  Npsi = 64, Ntheta = 512, Nphi = 64
  - presence of a wall
  - Petsc for all implicit solvers
  - Neutrals
  - Vorticity filtering





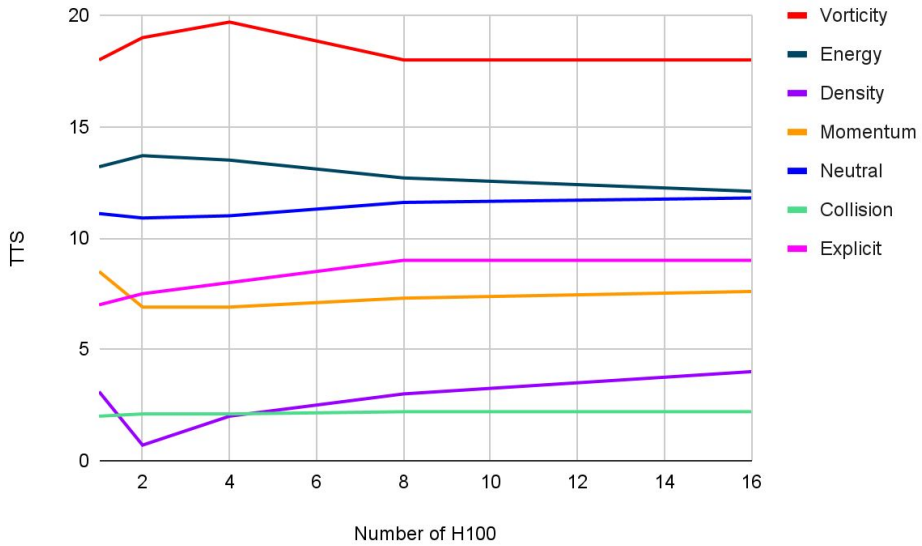
Preliminary results

- Benchmark:
  - **Jed@EPFL**: 2x Platinum 8360Y - 72 MPI
  - **Kuma@EPFL**: H100 - 1 GPU
  - **Pitagora@CINECA**: H100 - 1 GPU
  - **Pitagora@CINECA**: H100 - 256 MPI

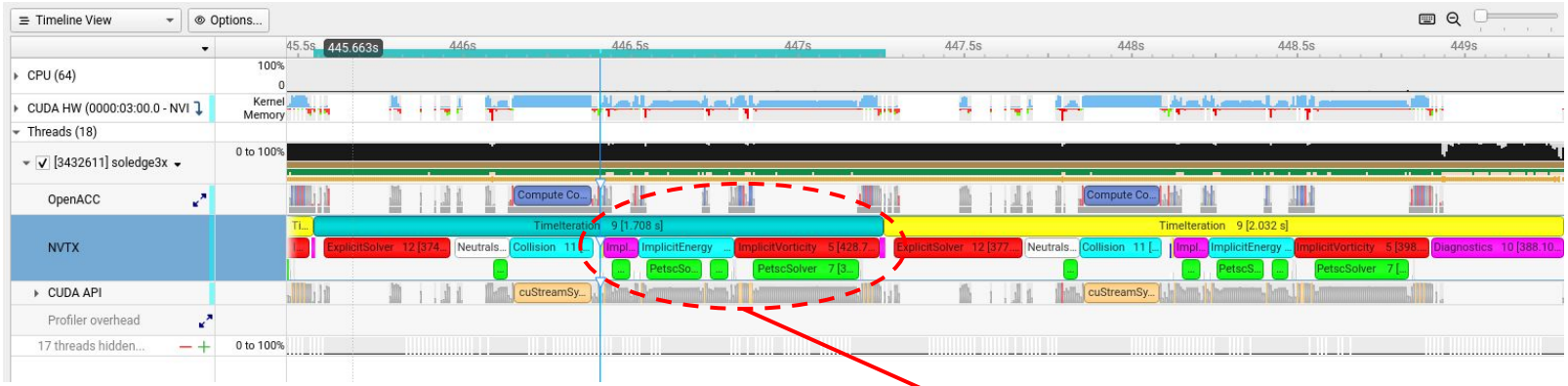


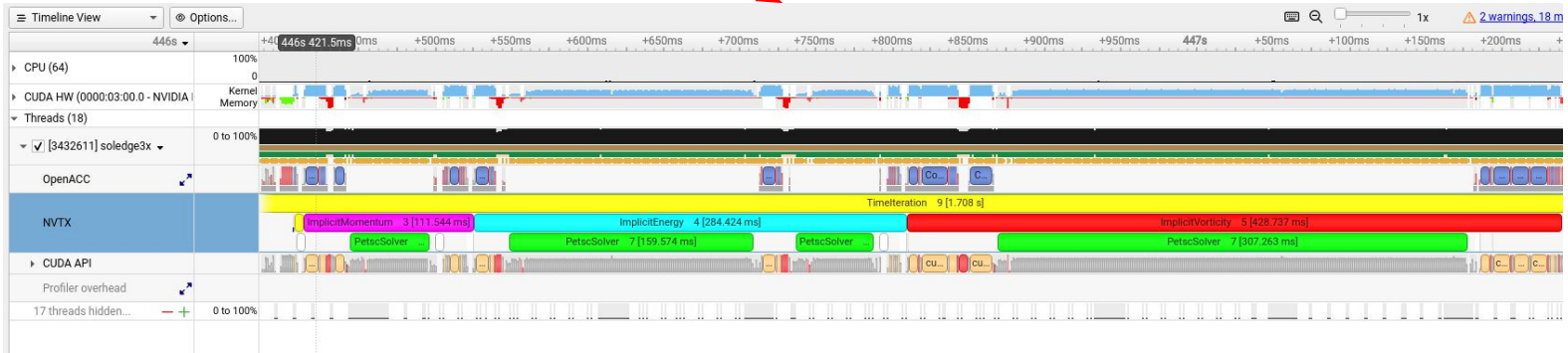
Weak scaling

- Benchmark:
  - Kuma@EPFL: H100, nvhpc/24.3
  - 2M cells per H100 or per cpu Nodes
  -



Profiling Nsys





work in progress

- Optimize Transfers
- GPU porting of Diags and Handle Diags with asynchronous tasks
- Test with OpenMP Offload