EPFL

TSVV3 activities

SOLEDGE3X Team

EUROfusion

EPFL

GPU porting strategy

- GPU porting
  - Open ACC/MP for advection and matrix construction (use of generic pragmas)
  - PETSC (with CUDA/HIP) for linear solvers

```
#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
```

```
!GPU_PARALLEL
  melt(0)=SpecMass(0) !e-
  !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
!GPU_END_PARALLEL
```

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.

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

Explicit solvers

- The cost of transposition is not significant

- For 64x256x64 mesh on GPU A100
  - 1ms for transposition
  - 80 ms for fluxes computation

Implicit solvers

- 5 implicit solvers in SOLEDGE3X :
  - **Parallel viscosity** terms
  - **Parallel heat conduction** terms

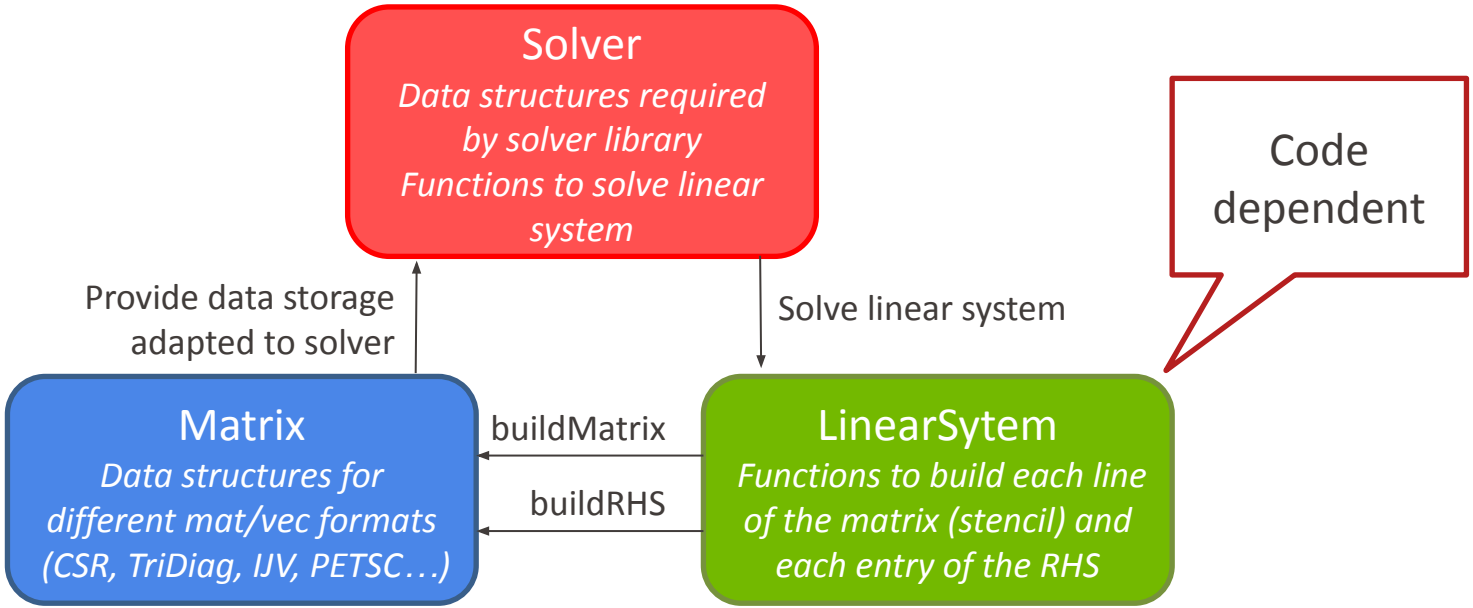
  } 2D

  - **Vorticity** equation
  - (optional) **fluid neutrals**
  - (optional) **potential filter**

  } 3D

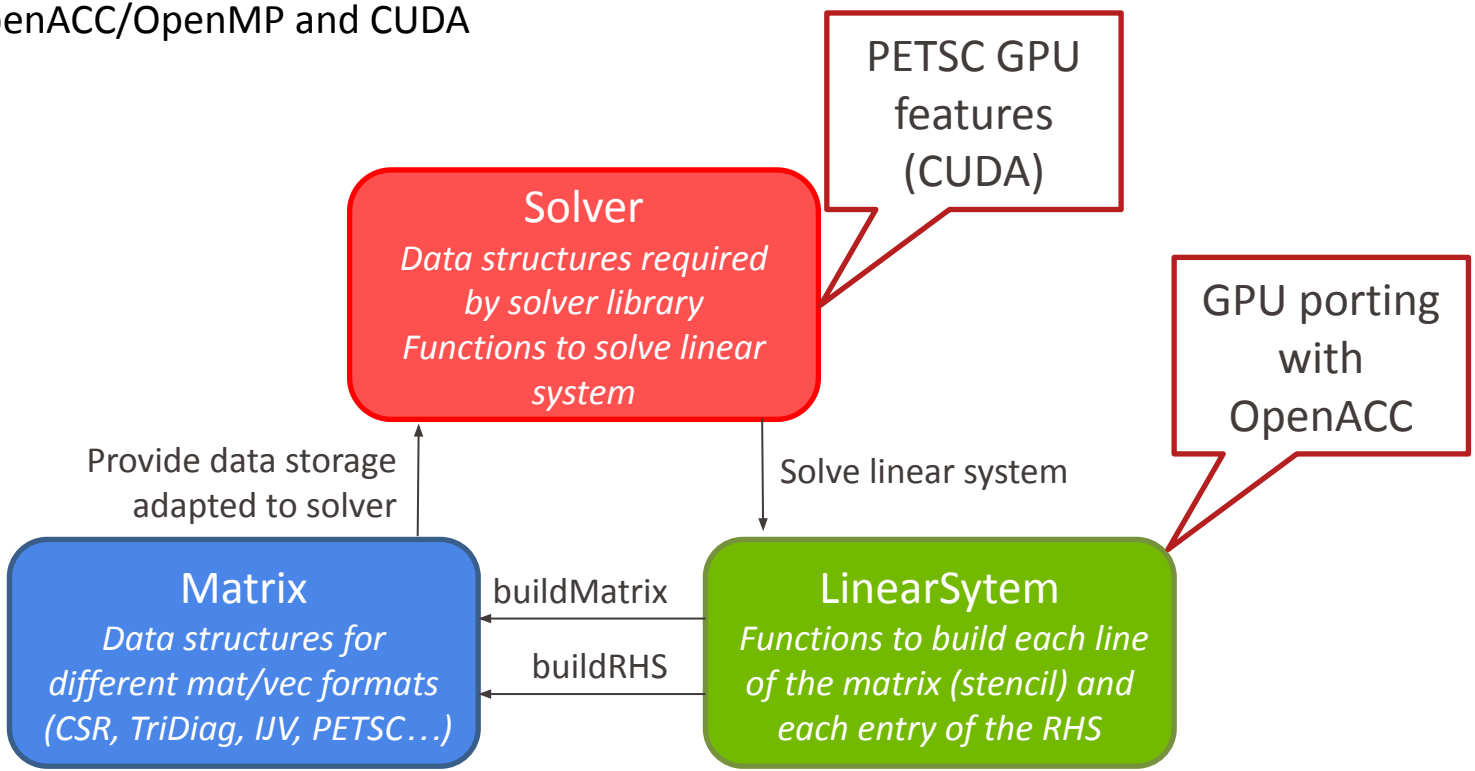
Implicit solvers

- 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

work in progress

- Finish to port explicit terms (in particular collision module)
- Propagate gpu porting of Vorticity solver to the other implicit solvers
  - facilitated by generic solver implementation
- 2D systems
  - Initially for viscosity and heat conduction terms: one 2D system per flux surface
  - Force single matrix for these 2D systems
- Tests on Grace-Hopper
- Handle MPI communications

Preliminary results