TSVV3 activities

SOLEDGE3X

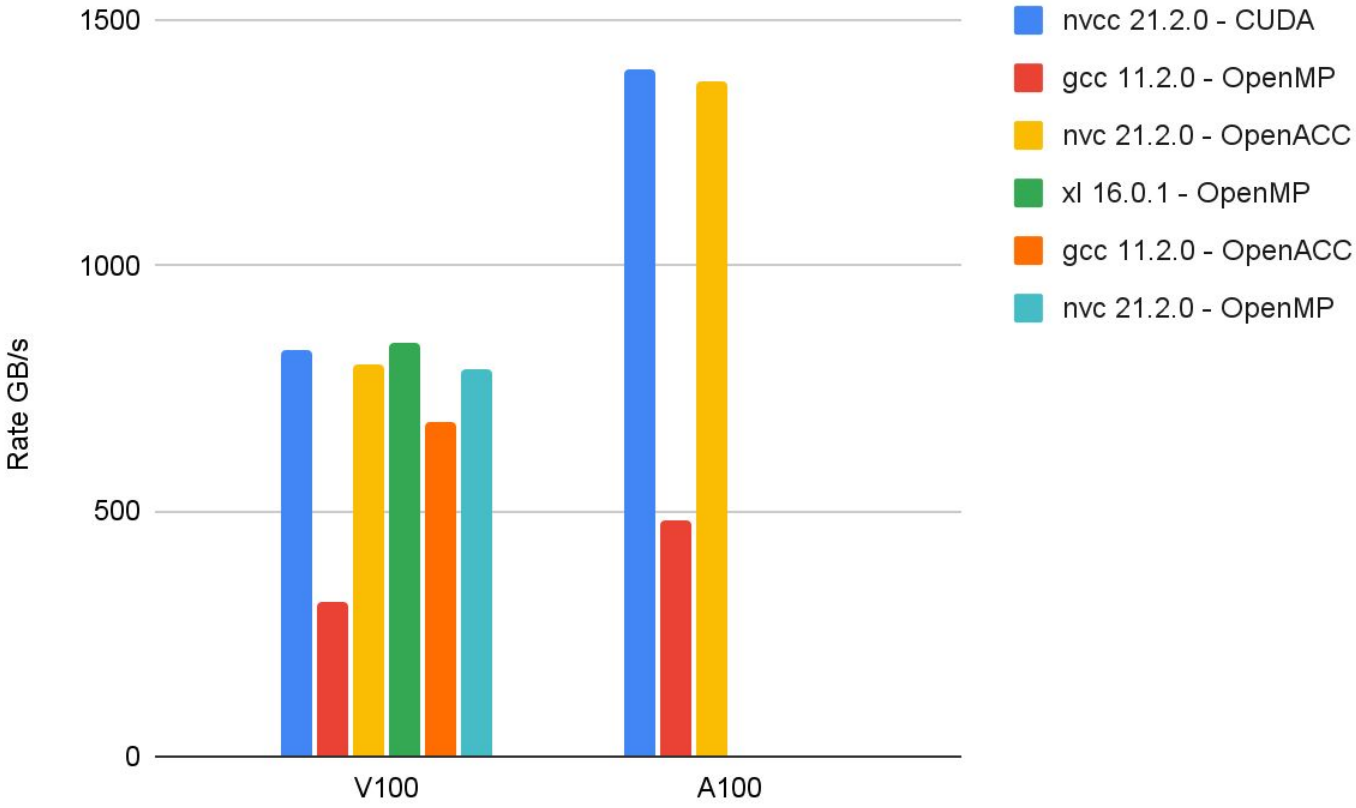
EUROfusion

EPFL

GPU Porting strategy

- For “explicit” solvers, porting by directives strategy with OpenMP offload and/or OpenACC has been chosen
    - This choice allows to keep only one version of code
    - And to perform tests on NVIDIA and AMD GPUs (and INTEL) for multiple compilers:
      xl (M100-Nvidia) ; gnu,gnu (Leonardo-Nvidia) ; cce (Adastra-AMD & LUMI-AMD)
    - On other fusion codes (CAS3D, Ascot5), this strategy shows good performance, in particular on Marconi100 with the IBM XL compiler with OpenMP offload
    - However, other compilers, like GCC, or NVHPC, present weak performances with OpenMP offload while OpenACC appears to be more efficient
    - This lack of performance portability led to introduce generic pragma to use OpenMP or OpenACC
- For implicit solvers use PETSC with new GPU features

Stream benchmark



GPU Porting strategy

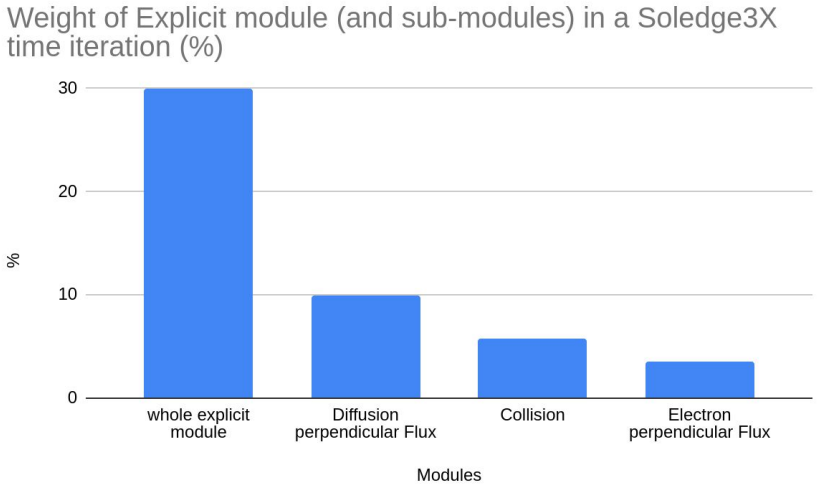
- Generic pragma for OpenMP/OpenACC

```
#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
```

Preliminary results

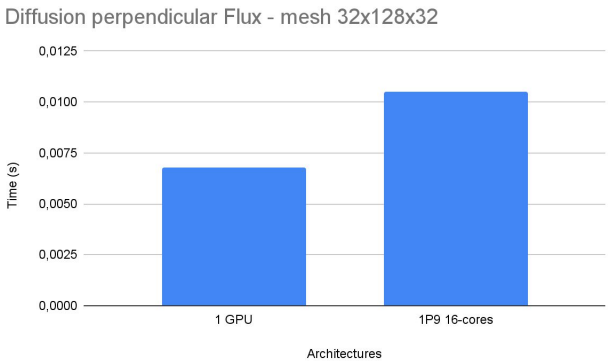
- Profiling with ScoreP allowed to point out the main kernels targeted for GPU porting in the explicit module

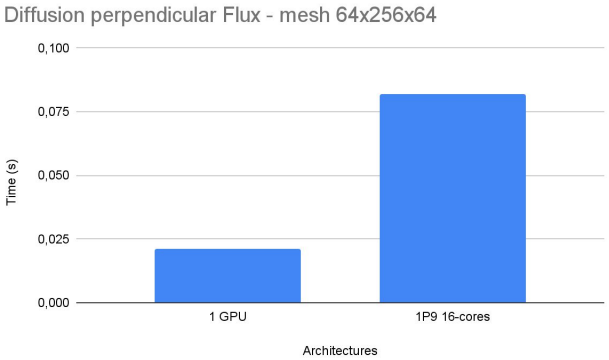


- First implementation in Explicit module with OpenMP and OpenACC:
  - Soledge3X has been installed on Marconi100 to exploit OpenMP with Xl compiler on V100
  - Soledge3X has been installed on EPFL cluster (equipped with Nvidia V100) to exploit OpenACC with NVHPC compiler to anticipate migration to Leonardo on Nvidia A100

Preliminary results

- Preliminary promising results on M100 on diffusion perpendicular fluxes computation (without cpu-gpu transfers):

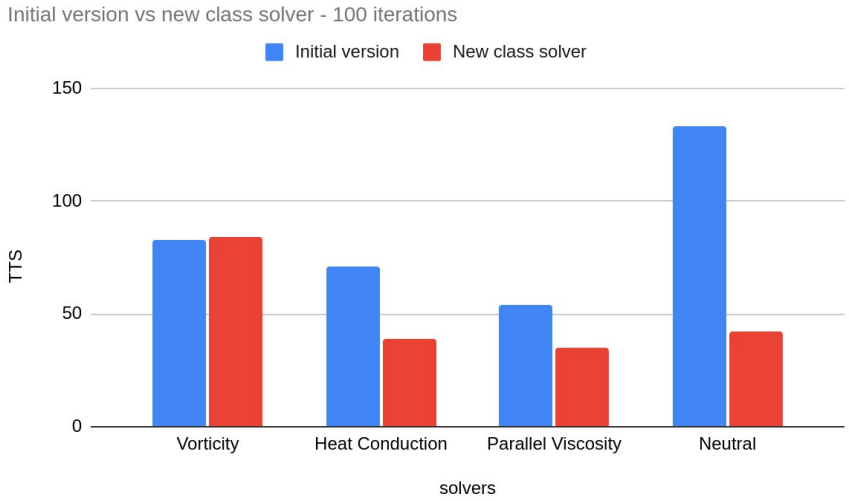




- Next steps : collision electron …
- Implicit solvers, see next slides

PETSC solver

- New factory pattern for implicit solvers
  - A new factory pattern was created by Soledge3X team to call every linear solver routine through generic procedures.
  - PETSc has been implemented in this new class solver
  - This new development allows also to set specific PETSc options for each 2D/3D solver and reuse the preconditioner depending on the solver and time step criteria
  - This implementation allows a gain up to 30%.



PETSC on GPU

- Last PETSC version (3.18) has been installed locally on M100 with new GPU features

- Usual and efficient preconditioners and solvers for matrices used in Soledge3X have been implemented in CUDA in PETSC:
  - GAMG for preconditioner
  - GMRES for solver

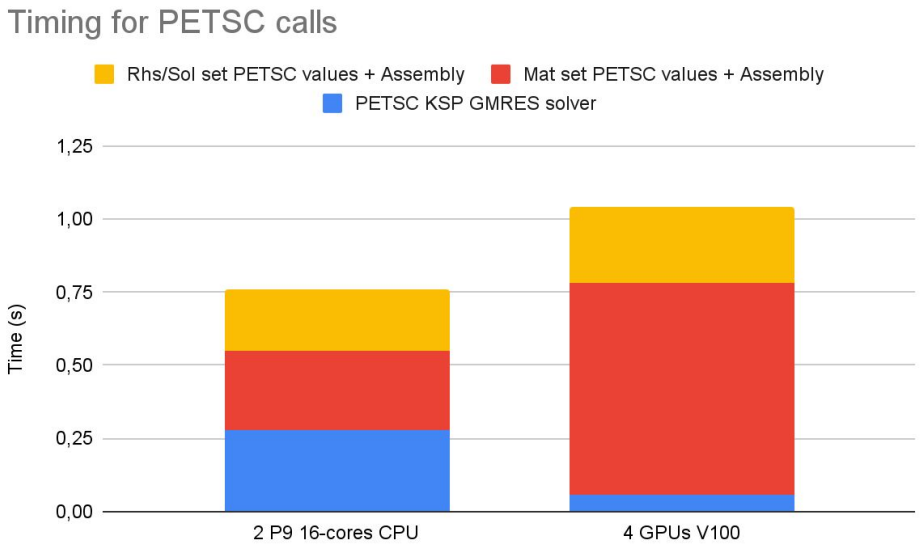
- Thanks to the new factory pattern, all 2D/3D implicit solvers can now run on GPU with PETSC using these command lines at run time:

-ksp_type gmres -pc_type gamg ***-mat_type mpiaijcusparse -vec_type mpicuda***

PETSC on GPU

- Preliminary promising tests focusing on PETSC KSP GMRES solver available on GPU show good performances (speed up x2.5 on 1 GPU V100 compared to 2 x P9 16-cores CPU)

- However CPU-GPU transfers still remain for PETSC values setting



PETSC 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
```

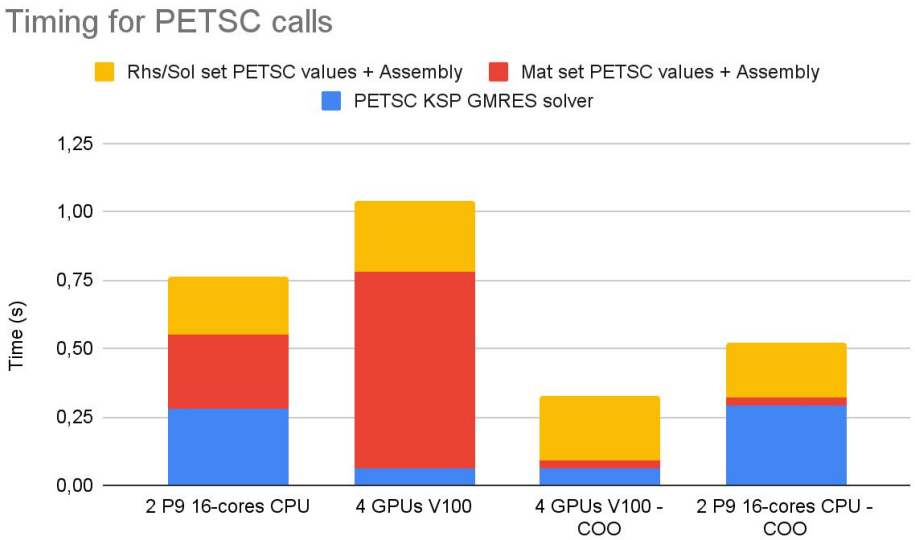
initial linSys_buildMat version

```
         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
                        !                            print*, "COPY to cuda pointers"
                        ! p_vorticity_stencVal     = stencVal
                        ! p_vorticity_irowGlobList = irowGlobList
                        ! p_vorticity_icolGlobList = icolGlobList
                        if (solver_vorticity_first_call) then
                           p_vorticity_oor(cnt:cnt+stencsize-1) = irowGlobList(1)
                           p_vorticity_ooc(cnt:cnt+stencsize-1) = icolGlobList(1:stencSize)
                        endif
                        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
      enddo ! ichunk
      cnt = cnt - 1
      petsc_cnt = cnt
      if (solver_vorticity_first_call) call
MatSetPreallocationCOO(mat_ptr%PETSCmat,petsc_cnt,p_vorticity_oor(1:cnt),p_vorticity_ooc(1:cnt),i
errPETSC)
      call
MatSetValuesCOO(mat_ptr%PETSCmat,p_vorticity_stencVal_coo(1:cnt),INSERT_VALUES,ierrPETSC)
```

new linSys_buildMat version

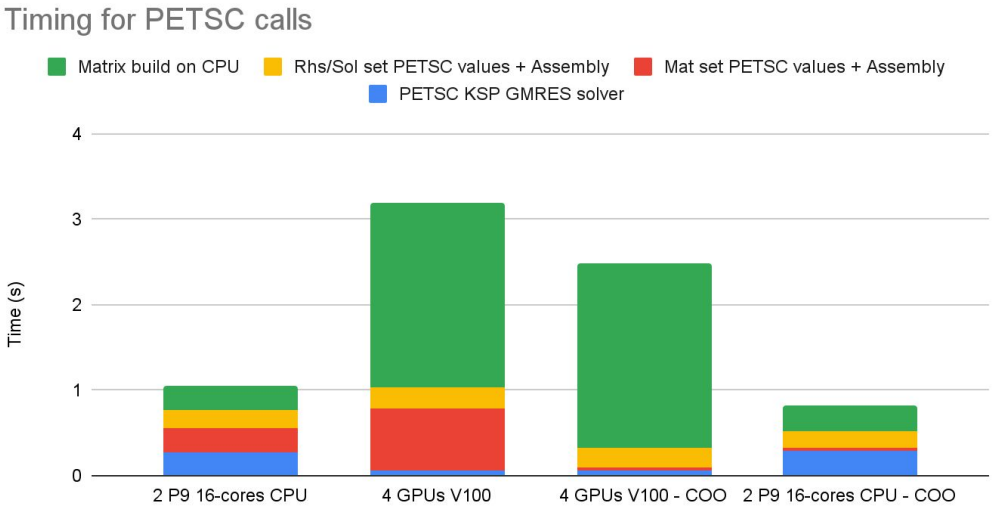
PETSC on GPU

- 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 …)



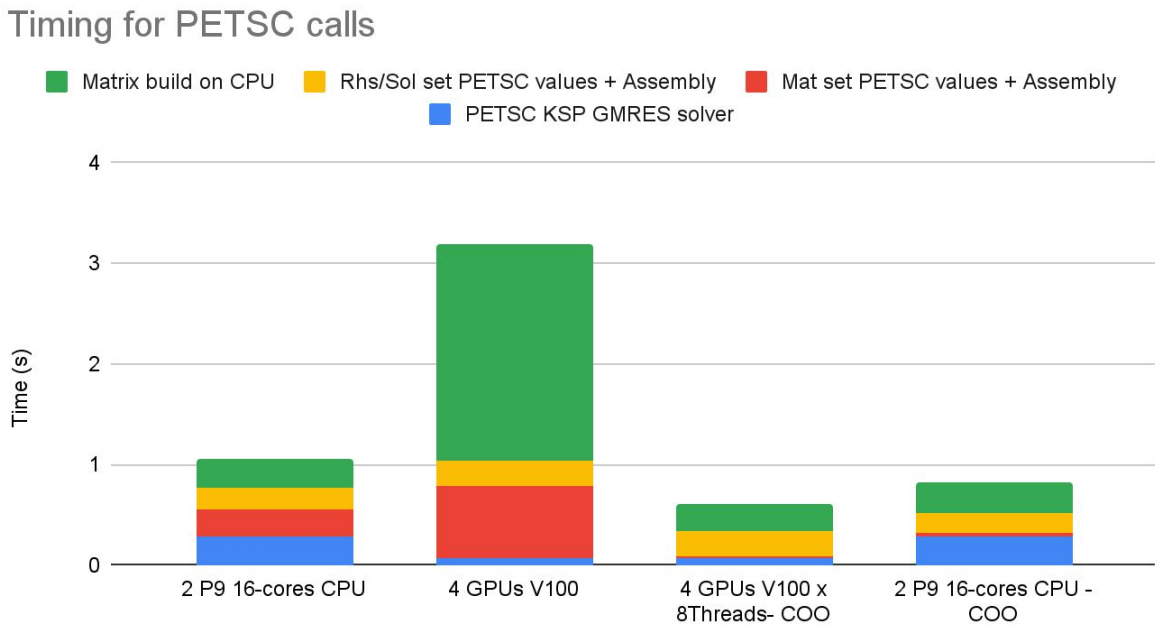
PETSC on GPU

- However matrices and vector are still built on CPU with many CPU-GPU transfers, which leads to low performance on GPU for global Implicit solver



PETSC on GPU

- Waiting for matrix building on GPU, this one can be computed with OpenMP



PETSC on GPU

On going work:

- Build matrix on GPU:
  - *MatSetValuesCOO* allows also to use cuda pointers
  - Use C binding to allocate arrays used by PETSC on GPU with cudallocmanaged
  - Use OpenMP *is_device_ptr* clause to use pointer both in cuda kernels (PETSC *MatSetValuesCOO*) and OpenMP offload regions (build Matrix)

PETSC on GPU

```
      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)
        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

                        if (solver_vorticity_first_call) then
                           p_vorticity_oor(cnt:cnt+stencsize-1) = irowGlobList(1)
                           p_vorticity_ooc(cnt:cnt+stencsize-1) = icolGlobList(1:stencSize)
                        endif
                        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
!$omp end target teams distribute parallel do simd
      enddo ! ichunk
      cnt = cnt - 1
      petsc_cnt = cnt
      if (solver_vorticity_first_call) call
MatSetPreallocationCOO(mat_ptr%PETSCmat,petsc_cnt,p_vorticity_oor(1:cnt),p_vorticity_ooc(1:cnt),i
errPETSC)
      call
MatSetValuesCOO(mat_ptr%PETSCmat,p_vorticity_stencVal_coo(1:cnt),INSERT_VALUES,ierrPETSC)
```

new linSys_buildMat version

PETSC on GPU

On going work:

- Use of Nvidia A100
  - Use of Miniapp for A100/V100 comparison (on 3D matrix from Implicit S3XE 3D solver)