2nd Annual Meeting of EUROfusion HPC ACHs

EPFL-ACH support to the JOREK code

C. Sommariva

EPFL – SPC, CH-1015, Lausanne Switzerland

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

⇒ Involvement & project status

⇒ Status of the task N# 1: implement native HDF5 parallel IO for JOREK particles

⇒ Status of the task N# 2: benchmark of the JOREK particles collision operators

⇒ Conclusions & Future Work

Involvement & Project Status

Involvement in the EUROfusion ACH-EPFL

⇒ Total involvement in the EUROfusion ACH-EPFL: **6PMs**

Projects:

- Support to the kinetic module of JOREK (JOREK-particles): **5.4PMs**
  - Task 1: Implement native HDF5 parallel IO for JOREK particles
  - Task 2: Benchmark of the JOREK particles collision operators
  - Task 3: Support the porting of JOREK particles on GPUs
- Visualization tools - Real-time dynamics and ParaView solutions: **0.6PMs**

⇒ Not reported here, please see F. Cabot presentation: “*Visualization tools - Real-time dynamics and ParaView solutions*” (@ ~12h40 on Wednesday 27th of November 2024)

Tasks agreed for the JOREK ACH-EPFL support 2024

Task	1 – Implement native HDF5 parallel IO for JOREK particles	2 – Benchmark of the JOREK particles collision operators	3 – Support the porting of JOREK particles on GPUs
Description	Implement native parallel HDF5 IO for removing one-node memory limit for JOREK-particles IO (MPI_Gatherv)	Benchmark JOREK-particles binary collision operator (non-relativistic) against the relativistic Langevin solver	Provide support for the porting of JOREK-particles to GPU via OpenMP and OpenACC pragmas
Status	PARTIALLY COMPLETED	IN PROGRESS	**PENDING**
Issues	Adjustments required before merging with JOREK production branch, impossible to access to LEONARDO for large performance testing	Delays with development of native HDF5 parallel IO, adaptation of the present implementation likely required	Insufficient number of PMs
Remaining actions to perform before completion	Performance testing of native HDF5 parallel IO with large datasets on LEONARDO and/or JED clusters	Adapt present code for performing meaningful benchmark, test & unit test adaptation, perform benchmarks	not planned for ACH projects anymore; covered by TSVV 8 itself with additional external support

Status of the task N# 1: implement native HDF5 parallel IO for JOREK particles

Task 1: premises and constraints for implementing native HDF5 parallel IO

JOREK-particle IO currently in production

- JOREK-particle data structure: list of derived “particle” types
- Hierarchical structure in HDF5 subdivided in particle groups & datasets
- Particle writing operations (∀ group):
  - Copy particle property in array
  - Gather property array from all MPI tasks to the master task
    - ⇒ ∀ column of multi-D array is gathered separately
  - Repeat for each particle property
  - Master task writes property array in HDF5 dataset
- Particle read operations (∀ group):
  - Parallel (native HDF5) read of property array by each MPI task
  - Copy property data in the respective field of the particle list element

```fortran
  ! Create group to write in
  write(group_name,"(A,i0.3,A)") "/groups/", i, "/"
  call h5gcreate_f(file, group_name, group_id, hdferr)
  call h5gclose_f(group_id, hdferr)
end if

! particle_base properties
! x
allocate(x(3,n_here), x_all(3,n_total))
do j=1,n_here
  x(:,j) = sim%groups(i)%particles(j)%x
end do
 call MPI_Gatherv(x(:,:), 3*n_here, MPI_REAL8, &
   x_all(:,:), particles_per_proc*3, [(sum(particles_per_proc(0:i-1),1)*3, i=0,n_cpu-1)], &
   MPI_REAL8, 0, MPI_COMM_WORLD, ierr)

call MPI_Gatherv(x(1,:), n_here, MPI_REAL8, &
                 x_all(1,:), particles_per_proc, [(sum(particles_per_proc(0:i-1),1), i=0,n_cpu-1)], &
                 MPI_REAL8, 0, MPI_COMM_WORLD, ierr)
call MPI_Gatherv(x(2,:), n_here, MPI_REAL8, &
                 x_all(2,:), particles_per_proc, [(sum(particles_per_proc(0:i-1),1), i=0,n_cpu-1)], &
                 MPI_REAL8, 0, MPI_COMM_WORLD, ierr)
call MPI_Gatherv(x(3,:), n_here, MPI_REAL8, &
                 x_all(3,:), particles_per_proc, [(sum(particles_per_proc(0:i-1),1), i=0,n_cpu-1)], &
                 MPI_REAL8, 0, MPI_COMM_WORLD, ierr)


! st
allocate(st(2,n_here), st_all(2,n_total))
do j=1,n_here
  st(:,j) = sim%groups(i)%particles(j)%st
end do
 call MPI_Gatherv(st(:,:), 2*n_here, MPI_REAL8, &
   st_all(:,:), particles_per_proc*2, [(sum(particles_per_proc(0:i-1),1)*2, i=0,n_cpu-1)], &
   MPI_REAL8, 0, MPI_COMM_WORLD, ierr)
call MPI_Gatherv(st(1,:), n_here, MPI_REAL8, &
                 st_all(1,:), particles_per_proc, [(sum(particles_per_proc(0:i-1),1), i=0,n_cpu-1)], &
                 MPI_REAL8, 0, MPI_COMM_WORLD, ierr)
call MPI_Gatherv(st(2,:), n_here, MPI_REAL8, &
                 st_all(2,:), particles_per_proc, [(sum(particles_per_proc(0:i-1),1), i=0,n_cpu-1)], &
```

Example of JOREK-particle write in production

```fortran
! Read base particle attributes
! x
allocate(real8_2D(3,n_here))
call HDF5_array2D_reading(file, real8_2D, group_name//"x",start=[0_HSIZE_T,i_here])
do j=1,n_here
  sim%groups(i)%particles(j)%x = real8_2D(:,j)
end do
deallocate(real8_2D)

! st
allocate(real8_2D(2,n_here))
call HDF5_array2D_reading(file, real8_2D, group_name//"st",start=[0_HSIZE_T,i_here])
do j=1,n_here
  sim%groups(i)%particles(j)%st = real8_2D(:,j)
end do
deallocate(real8_2D)

! weight
allocate(real8_1D(n_here))
call HDF5_array1D_reading(file, real8_1D, group_name//"weight",start=[i_here])
do j=1,n_here
  sim%groups(i)%particles(j)%weight = real8_1D(j)
end do
deallocate(real8_1D)
! i_elm
allocate(int4_1D(n_here))
call HDF5_array1D_reading_int(file, int4_1D, group_name//"i_elm", start=[i_here])
do j=1,n_here
  sim%groups(i)%particles(j)%i_elm = int4_1D(j)
end do
deallocate(int4_1D)
```

Example of JOREK-particle reading in production

Objective:

Maximum number of particles limited by the available memory in one node

⇒ **Improve scalability implementing native parallel HDF5 writing operations**

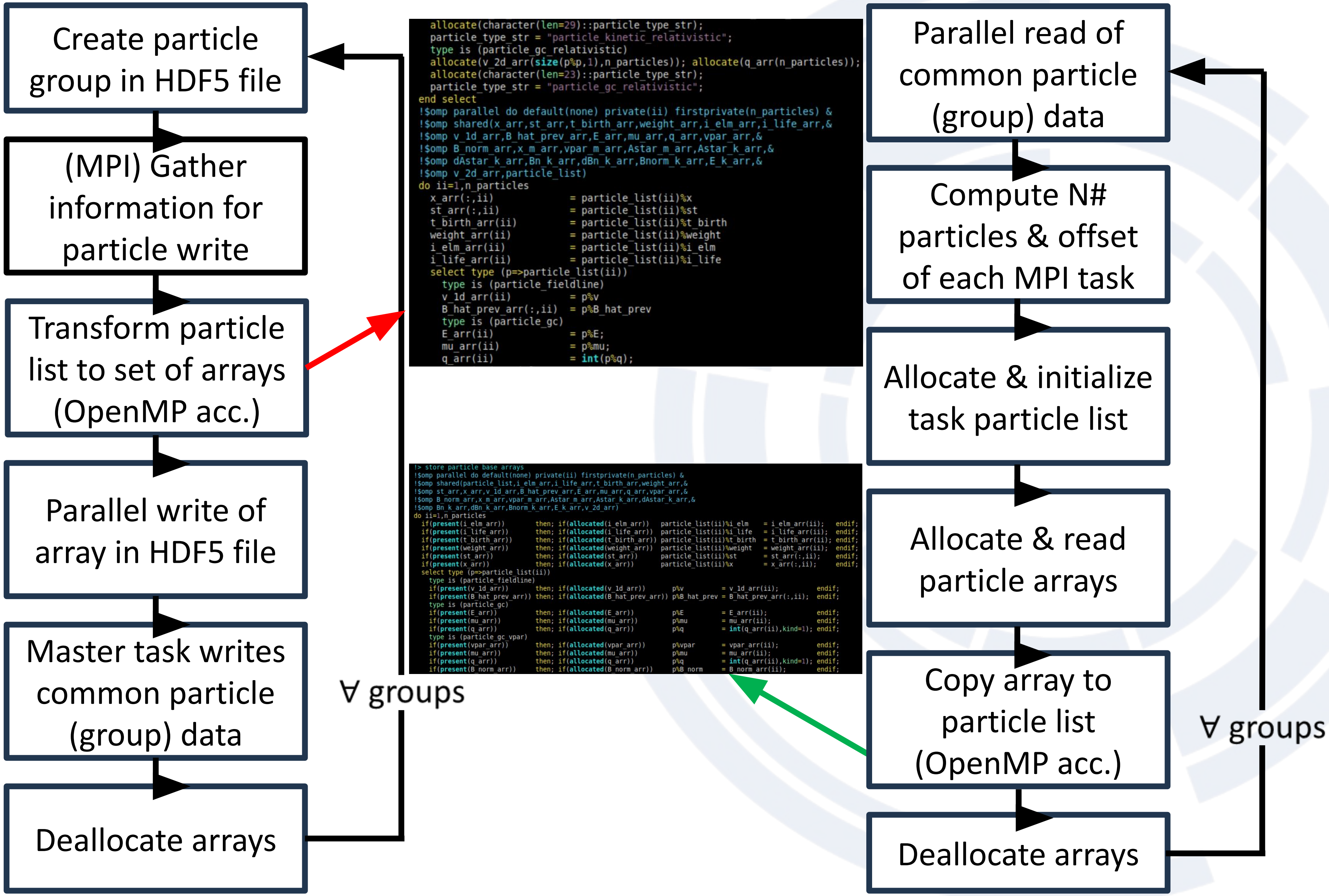
Constraints:

- Full (retro) compatibility with already existing JOREK-particles HDF5 files
- Preserve present implementation of JOREK-particles routines without duplications
- Do not degrade the performance of the JOREK-particles IO routines
- Provide unit tests and documentation

Task 1: status of the native HDF5 parallel IO implementation

<u>Particle IO fully restructured:</u>

- MPI (All)Gather only instanced once for retrieving particle info
- Data copy to/from particle list to/from arrays done only once
  - ⇒ Full OpenMP enabled
  - ⇒ May induce a slighter increase of memory usage
- MPI_Gatherv calls wrapped in common subroutine with native HDF5 parallel write
  - ⇒ MPI_Gatherv moved in common HDF5 module (available to all code)
- Group properties wrote only by the master task
  - ⇒ Assumed to be the same for each MPI task



Task 1: further details on the native HDF5 parallel IO implementation

combined MPI_Gatherv / native parallel HDF5

```
! check preset
mpio_collective=.true.; if(present(mpio_collective_in)) mpio_collective=mpio_collective_in;
! check whether gatherv can/should be used default false
use_gatherv_loc=use_gatherv.and.present(mpi_rank).and.present(n_cpu).and.&
present(mpi_comm_loc).and.present(dim1_all_tasks).and.present(displs)
use_hdf5_parallel=.false.
if(present(use_hdf5_parall  n)) use_hdf5_parallel=use_hdf5_parallel_in
if(use_gatherv_loc) use_hd  parallel=.not.use_gatherv_loc

! Gather all arrays in  e
if(use_gatherv_loc) th
  call MPI_Gatherv(ar  y1D,dim1_all_tasks(mpi_rank+1),MPI_INTEGER,array1D_tot,&
  dim1_all_tasks,di  ls,MPI_INTEGER,master_task,mpi_comm_loc,ierr)
  if(mpi_rank.eq.m  ster_task) then
    if(present(st  rt).and.present(compress_level)) then
                                   le_id,array1D_tot,dim1_tot,dse
                                   se_hdf5_parallel_in=use_hdf5_p
                                   ive)
                                   le_id,array1D_tot,dim1_tot,dse
      use_hdf5_parallel_in=use_hdf5_parallel,mpio_collective_in=mpi
    else if(present(compress_level)) then
      call HDF5_array1D_saving_int(file_id,array1D_tot,dim1_tot,dsetname,compress_level=compress_level,&
      use_hdf5_parallel_in=use_hdf5_parallel,mpio_collective_in=mpio_collective)
    else
      call HDF5_array1D_saving_int(file_id,array1D_tot,dim1_tot,dsetname,&
      use_hdf5_parallel_in=use_hdf5_parallel,mpio_collective_in=mpio_collective)
    endif
  endif
else
  if(present(start).and.present(compress_level)) then
    call HDF5_array1D_saving_int(file_id,array1D,dim1_tot,dsetname,&
    start=start,compress_level=compress_level,use_hdf5_parallel_in=use_hdf5_paral  el,&
    mpio_collective_in=mpio_collective)
  else if(present(start)) then
    call HDF5_array1D_saving_int(file_id,array1D,dim1_tot,dsetname,&
    start=start,use_hdf5_parallel_in=use_hdf5_parallel,&
    mpio_collective_in=mpio_collective)
  else if(present(compress_level)) then
    call HDF5_array1D_saving_int(file_id,array1D,dim1_tot,dsetname,&
    compress_level=compress_level,use_hdf5_parallel_in=use_hdf5_parallel,&
    mpio_collective_in=mpio_collective)
  else
    call HDF5_array1D_saving_int(file_id,array1D,dim1_tot,dsetname,&
    use_hdf5_parallel_in=use_hdf5_parallel,mpio_collective_in=mpio_collective)
  endif
endif
```

Use Gatherv?

Gather array from all MPI

Execute native parallel HDF5 otherwise

Execute serial HDF5 write if required

native parallel HDF5 write

```
!*** Check property for parallel IO
mpio_collective   = .true.
if(present(mpio_collective_in)) mpio_collective = mpio_collective_in
transfer_property = H5P_DEFAULT_F; use_hdf5_parallel = .false.
if(present(use_hdf5_parallel_in)) use_hdf5_parallel = use_hdf5_parallel_in
if(use_hdf5_parallel) call HDF5_set_parallel_io_properties(transfer_property,mpio_collective)

!*** Create and initialize dataspaces for datasets ***
dim(1) = dim1
rank   = 1
call H5Screate_simple_f(rank,dim,filespace,error

!*** Get compression property list ***
if (present(compress_level)) then
  property = get_HDF5_plist(rank, dim, .not. pr  sent(start), compress_level)
else
  property = get_HDF5_plist(rank, dim, .false.)
end if

!*** Create real dataset ***
call H5Dcreate_f(file_id,trim(dsetname),H5T_NATIVE_INTEGER,&
  filespace,dataset,error,property)

!*** Write the integer array data to the dataset using ***
!***  default transfer properties                      ***
if (present(start)) then
  call H5Screate_simple_f(1,shape(array1D,kind=HSIZE_T),dataspace,er  r)
  call H5Sselect_hyperslab_f(filespace, H5S_SELECT_SET_F, &
  start=start, count=shape(array1D,kind=HSIZE_T), hdferr=error)
  call H5Dwrite_f(dataset,H5T_NATIVE_INTEGER,array1D,shape(array1D,kin  HSIZ  ),error, &
    file_space_id=filespace, mem_space_id=dataspace,xfer_prp=transfer_pr  pert
  call H5Sclose_f(dataspace,error)
else
  call H5Dwrite_f(dataset,H5T_NATIVE_INTEGER,array1D,dim,error)
end if

!*** Closing ***
call H5Pclose_f(property,error)
call H5Sclose_f(filespace,error
call H5Dclose_f(dataset,error)
call H5Pclose_f(transfer_pro  rty,error)
end subroutine HDF5_array1D_sa  ing_int
```

Set property for IO

Create memory space & dataset

Write all array in HDF5 dataset

Write data hyperslab in HDF5

native parallel HDF5 read

```
!*** check if dataset exists otherwise return ***
call H5Lexists_f(file_id,trim(dsetname),exists,error)
if(.not.exists) then
  write(*,*) "WARNING: dataset ",dsetname," does not exist!"
  return
endif
!*** initialisation ***
dims = [int(0,kind=HSIZE_T)]; if(allocated(ar
                                  ,dataset_id,error)
                                  g the dataset ",dsetname,"!"
endif
call H5Dget_space_f(  ataset_id,dataspace_id,error)
!*** get the space ra  k ***
call H5Sget_simple_ex  ent_ndims_f(dataspace_id,rank,error)
if(rank.eq.1) then
  !*** get space size ***
  call H5Sget_simple_ext  nt_dims_f(dataspace_id,dims,maxdims,error)
  if(present(reqdims_in)  d.present(start)) then
    !*** load a data sla  f size reqdims and start start ***
    !*** using default tra  sfer properties ***
    allocate(array1D(reqdims_in(1))); array1D = 0;
    call H5Screate_simple_f(rank,reqdims_in,dataspace_req_id,error)
    call H5Sselect_hyperslab_f(dataspace_id, H5S_SELECT_SET_F, &
    start=start, count=shape(array1D,kind=HSIZE_T), hdferr=error)
      ll H5Dread_f(dataset_id,H5T_NATIVE_INTEGER,array1D,reqdims_in,error, &
      file_space_id=dataspace_id, mem_space_id=dataspace_req_id)
    call  5Sclose_f(dataspace_req_id,error)
  e se
    !*** allocate and read read the dataset ***
    !*** using default transfer properties ***
    allocate(array1D(dims(1))); array1D = 0;
    ca  l H5Dread_f(dataset_id,H5T_NATIVE_INTEGER,array1D,dims,error)
  endif
else
    ite(*,*) 'Read 1D allocatable integer arr  from HDF5, rank is not 1: skip!'
endif
!*** Closing ***
call H5Sclose_f(dataspace_id,error)
call H5Dclose_f(dataset_id,error)
end subroutine HDF5_allocatable_array1D_reading_int
```

Get array size

Allocate array

Select hyperslab and read array

Simple read of the dataset

Task 1: additional implementations and remaining subtasks

Additional subtasks performed:

- Implemented HDF5 write/read arrays for:
  - 1D-characters, 3D-integers, 3D-double, 4D-double, 5D-double
- Implemented reading routines for allocatable arrays
- Implemented helping routines for:
  - Get dataset rank, dimensions and shape, set parallel IO properties
- Implemented unit tests for:
  - HDF5 I/O routines, particle I/O routines, action wrappers for I/O
- Extensive documentation in the code
- Tested backward compatibility & parallel I/O operations

```
!-----------------------------------------
! HDF5 saving for a 1D array of characters
! For parallel applications the variable: mpi_comm_in and
! start must be defined.
!-----------------------------------------
! inputs:
!   file_id:              (HID_T) file identifier
!   array1D:              (character)(*)(:) array of characters to be written in file
!   dim1:                 (integer) size of the array
!   dsetname:             (character)(*) name of the dataset in which the data are written
!   start:                (integer)(1)(optional) starting index of the input data chunk
!                         in the global dataset
!   mpi_comm_in:          (integer)(optional) identifier of the MPI communicator
!   use_hdf5_parallel_in: (logical)(optional) if true, dataset transfer property is
!                         set to parallel MPIO, H5P_DEFAULT is used otherwise
!   mpio_collective_in:   (logical)(optional) toggle MPIO collective actions if true
!-----------------------------------------
subroutine HDF5_array1D_saving_char(file_id,array1D,dim1,dsetname,start,&
mpi_comm_in,use_hdf5_parallel_in,mpio_collective_in)
```

```
!-----------------------------------------
! create and set HDF5 parallel IO transfer properties
!-----------------------------------------
! Create a HDF5 - MPI transfer property list and set the behaviour of the
! paralle I/O. Presently, available parallel I/O behaviour are:
! COLLECTIVE: all MPI tasks must performed the same operation
! INDEPENDENT: not all MPI tasks must performed the same operation
! actions modifying the structure of a file or of the metadata must be done collectively
! inputs:
!   mpio_collective_in: (logical)(optional) if true (default) MPIO operations are collective
! outputs:
!   transfer_property: (HID_T) transfer property list defining the MPIO behaviour
!-----------------------------------------
subroutine HDF5_set_parallel_io_properties(transfer_property,mpio_collective_in)
```

Pending subtasks before completion:

- Performance test on large datasets on Leonardo HPC
  - ⇒ Waiting access to Leonardo HPC (CINECA authentication system down)
  - ⇒ Granted access to EPFL-JED cluster, installation of JOREK on JED underway
- Generate documentation for JOREK wikipage

Status of the task N# 2: benchmark of the JOREK particles collision operators

Task 2: binary collision vs relativistic Langevin solver

Binary collision solver [Homma 2012/2013/2016/2020]

- Takizuka-Abe collision operator for non-relativistic particles
  - Applicable only to uniformly weighted particles
  - Collision kernel based on Gaussian distribution of collision angles with $\langle\theta\rangle = 0$, $\langle\theta^2\rangle = \frac{n_b q_a^2 q_b^2 \ln\Lambda}{8\pi\varepsilon_0^2 m_r^2 u^3}\Delta t$
  - Different Coulomb logarithms available
    - Plasma physical properties recovered as a function of the heat flux definition
      - Available heat-flux: non-magnetized, strongly magnetized, generalized flux limiter
- Fully unit-tested (implemented within JOREK task 2), benchmarked against Homma 2012/2013 results

Relativistic Langevin solver [Särkimäki 2018]

- Langevin solver for Fokker-Planck (SDE) collision operator
  - Computes relativistic Coulomb collisions between particle or guiding center w background plasma (relativistic only)
  - Relativistic Fokker-Planck operator defined by advection, parallel & perpendicular diffusion coefficients
  - Fixed time step implementation only
- Background plasma modeled by Maxwell-Jüttner distribution based on MHD fields
- Benchmarked against DREAM code [Hoppe 2021] using the Bump-on-Tail test case [jorekwiki]

Binary & Langevin schemes solve different physics (relativistic/non-relativistic)
⇒ Unclear meaning of a direct benchmark

Task 2: extension of binary collision to relativistic collisions

Codes should solve similar physics before benchmarking:

⇒ **Probably, the easiest path is to extend the binary collision to relativistic particle collisions!**

Kinematic of relativistic binary collision for uniformly weighted particles developed in [Sentoku 1998, Sentoku 2008, Pérez 2012, Higginson 2020, Lavell 2024]:

- define the Center-of-Momentum (CM) of the collision particle pair: $\boldsymbol{v}^{CM} = \frac{\boldsymbol{p}_\alpha + \boldsymbol{p}_\beta}{m_\alpha \gamma_\alpha + m_\beta \gamma_\beta}$, $\gamma^{CM} = \left(1 - \frac{{v^{CM}}^2}{c^2}\right)^{-\frac{1}{2}}$
- transform the particle 4-momentum in the CM reference:

$$\boldsymbol{p}_{\alpha,\beta}^{CM} = \boldsymbol{p}_{\alpha,\beta} + \left[\frac{\gamma^{CM} - 1}{{v^{CM}}^2}\left(\boldsymbol{v}^{CM} \cdot \boldsymbol{p}_{\alpha,\beta}\right) - m_{\alpha,\beta}\gamma_{\alpha,\beta}\gamma^{CM}\right]\boldsymbol{v}^{CM}, \gamma_{\alpha,\beta}^{CM} = \gamma_\alpha \gamma^{CM}\left[1 - \frac{\boldsymbol{v}^{CM} \cdot \boldsymbol{v}_{\alpha,\beta}}{c^2}\right]$$

- compute the binary collision given the collisional polar (θ^{CM}) and azimuthal (ϕ^{CM}) angles

$$\boldsymbol{p}_\alpha^{\prime,CM} = -\boldsymbol{p}_\beta^{\prime,CM} = \begin{bmatrix} \frac{p_{\alpha x}^{CM} p_{\alpha z}^{CM}}{p_{\alpha\perp}^{CM}} & -\frac{p_{\alpha y}^{CM} p_{\alpha}^{CM}}{p_{\alpha\perp}^{CM}} & p_{\alpha x}^{CM} \\ \frac{p_{\alpha y}^{CM} p_{\alpha z}^{CM}}{p_{\alpha\perp}^{CM}} & -\frac{p_{\alpha x}^{CM} p_{\alpha}^{CM}}{p_{\alpha\perp}^{CM}} & p_{\alpha y}^{CM} \\ -p_{\alpha\perp}^{CM} & 0 & p_{\alpha z}^{CM} \end{bmatrix} \begin{bmatrix} \sin\theta^{CM} \cos\phi^{CM} \\ \sin\theta^{CM} \sin\phi^{CM} \\ \cos\theta^{CM} \end{bmatrix}, p_{\alpha\perp}^{CM} = \sqrt{{p_{\alpha x}^{CM}}^2 + {p_{\alpha y}^{CM}}^2}, p_\alpha^{CM} = \|\boldsymbol{p}_\alpha^{CM}\|$$

- transform the 4-momentum back from the CM to the laboratory frame

Task 2: binary collision kernels

Takizuka-Abe like collision kernel [Sentoku 2008]

- Compute velocity in one-particle-at-rest frame ($\boldsymbol{v}_\beta = 0$)

$$\boldsymbol{v}_\alpha^{OPR} = \frac{1}{1 - \frac{\boldsymbol{v}_\alpha \cdot \boldsymbol{v}_\beta}{c^2}} \left[\frac{\boldsymbol{v}_\alpha}{\gamma_\beta} - \left(1 - \frac{\gamma_\beta}{1+\gamma_\beta} \frac{\boldsymbol{v}_\alpha \cdot \boldsymbol{v}_\beta}{c^2} \right) \boldsymbol{v}_\beta \right]$$

- Collision frequency $\nu_{\alpha,\beta}^{coll}$ computed as in [Särkimäki 2018] already implemented in JOREK relativistic Langevin solver
- Compute the polar collision angle in OPR frame:
  - Gaussian distrib. $\langle \theta^{OPR} \rangle = 0$, $\left\langle \tan^2 \frac{\theta^{OPR}}{2} \right\rangle = \nu_{\alpha,\beta}^{coll} \Delta \mathrm{t}^{\mathrm{coll}}$
  - Same Box-Muller strategy as in JOREK binary collision
- Transform the polar angle in the CM frame

$$\tan \theta^{CM} = \frac{\sin \theta^{OPR}}{\gamma^{CM} \left(\cos \theta^{OPR} - \frac{v^{CM}}{v^{OPR}} \right)}$$

- Azimuthal angle: $\phi^{CM} = 2\pi u$ w $u \in [0,1)$ uniformly distributed random number

Nambu-like collision kernel [Pérez 2012, Higginson 2020, Lavell 2024]

- Collision frequency $\nu_{\alpha,\beta}^{coll}$ computed as in [Särkimäki 2018] already implemented in JOREK relativistic Langevin solver
- Compute the normalized path length
  - $s^{max} = \sqrt[3]{\frac{4\pi}{3}} \frac{n_{eff} \| v_\alpha - v_\beta \| (m_\alpha + m_\beta)}{\max\left(m_\alpha n_\alpha^{2/3}, m_\beta n_\beta^{2/3} \right)} \Delta t^{coll}$
  - $s = \min\left(\nu_{\alpha,\beta}^{coll} \Delta \mathrm{t}^{\mathrm{coll}}, s^{max} \right)$
- Compute A (fitting available): $\coth A - A^{-1} = e^{-s}$
- Compute the polar collision angle θ^{CM} ($\mathrm{u}_\theta \in [0,1)$ uniformly distributed random number):

$$\cos \theta^{CM} = A^{-1} \ln[e^{-A} + 2u_\theta \sinh A]$$

- Azimuthal angle: $\phi^{CM} = 2\pi u_\phi$ w $u_\phi \in [0,1)$ uniformly distributed random number

Task 2: sampling particle from background plasma & foreseen benchmarks

- Equilibrium distribution for relativistic plasmas is the Maxwell-Jüttner distribution: $f_{MJ}(\boldsymbol{p})d^3p = \frac{N_M}{4\pi m^2 c T_M K_2\left(\frac{mc^2}{T_M}\right)} e^{-\frac{\gamma m^2}{T_M}} d^3p$ w

K_n: Modified Bessel function of the second kind, T_M, N_M: plasma temperature and number density

$\Rightarrow$ The β −colliding particle must be sampled from the Maxwell-Jüttner distribution for consistency w [Särkimäki 2018]

Modified Canfield method [Zenitani 2022]:

$\Rightarrow$ Efficient method for sampling particles from a Maxwell-Jüttner distribution

- Modified Canfield method: accept-reject method based on:
  - Momentum intensity Γ −distributed variate
    - $\Rightarrow \Gamma$ − distributed RNG algorithm provided in [Zenitani 2022]
  - Momentum direction obtained from Box-Muller transform
  - Accept/reject parameters chosen for optimal convergence

Algorithm 2: the modified Canfield method

```
a ← 0.56,  b ← 0.35,  R_0 ← 0.95
compute π_3, π_4, π_5 for given t using Eqs. (22)–(24)
repeat
   generate X_1, X_2 ~ U(0, 1)
   if      X_1 < π_3              then i ← 3
   elseif  X_1 < π_3 + π_4        then i ← 4
   elseif  X_1 < π_3 + π_4 + π_5  then i ← 5
   else i ← 6
   endif
   generate x ~ Ga(i/2, t)
until X_2 < R_0 or X_2 < R(x; a, b)
generate X_3, X_4 ~ U(0, 1)
p ← sqrt(x(x + 2))
p_x ← p (2X_3 − 1)
p_y ← 2p sqrt(X_3(1 − X_3)) cos(2πX_4)
p_z ← 2p sqrt(X_3(1 − X_3)) sin(2πX_4)
```

Modified Canfield method as proposed in tab.II of [Zenitani 2022]

Foreseen collision operator benchmarks:

$\Rightarrow$ Relaxation of particle distribution towards Maxwell-Jüttner equilibrium

$\Rightarrow$ Bump-on-tail physics as reported in [jorekwiki]

Conclusions & Future work

Conclusions

- **Implement of native HDF5 parallel I/O for JOREK-particles:**
  - ❑ Restructured I/O operations w OpenMP accelerated particle list – array copy operations
  - ❑ Implemented hybrid MPI_Gatherv / parallel HDF5 writing routines
    - ⇒ Adapted HDF5 writing routines to native HDF5 parallelization
    - ⇒ Implemented hybrid MPI_Gatherv / parallel HDF5 writing routines
    - ⇒ Implemented writing operations for additional datatypes
  - ❑ Restructured/improved parallel HDF5 reading routines
    - ⇒ Implemented reading method for additional datatypes
    - ⇒ Implemented reading method for allocatable arrays
  - ❑ Unit tested and tested backward compatibility of HDF5 and particle I/O routines

- **Benchmark of the JOREK-particles collision operators:**
  - ⇒ Identified suitable strategy for sampling particles from Maxwell-Jüttner distribution
  - ⇒ Identified numerical scheme for performing relativistic binary particle collisions
  - ⇒ Identified suitable relativistic collision kernels: Takizuka-Abe & Nambu-like collision kernels

Future work

- **Actions for task 1 completion**:
  - ❑ Performance test of new JOREK-particles IO routines on large datasets
    - ⇛ Install JOREK on JED & perform tests
    - ⇛ Waiting for access to CINECA-Leonardo supercomputing
  - ❑ Write documentation in JOREK wiki server

- **Actions for task 2 completion**:
  - ❑ First implementation & unit testing of relativistic binary collision operator
    - ⇛ Introduce random sampling of Maxwell-Jüttner distribution
    - ⇛ Implement kinematic of relativistic binary collisions
    - ⇛ Implement relativistic Takizuka-Abe like collision kernel
  - ❑ First benchmark between Binary (Takizuka-Abe) and Langevin collision operator
    - ⇛ Relaxation towards Maxwell-Jüttner distribution
    - ⇛ Bump-on-tail distribution
  - ❑ Implementation of relativistic Nambu-like collision kernel and benchmarking

Thank you for the attention!

References

[Homma 2012] Y. Homma et A. Hatayama, J. Comp. Phys., vol. 231, p. 3211, 2012
[Homma 2013] Y. Homma et A. Hatayama, J. Comp. Phys., vol. 250, p. 206, 2013
[Homma 2016] Y. Homma et al., Nucl. Fusion, vol. 56, p. 036009, 2016
[Homma 2020] Y. Homma et al., Nucl. Fusion, vol. 60, p. 046031, 2020
[Särkimäki 2018] K. Särkimäki et al., Comp. Phys. Comm., vol. 222, p. 374, 2018
[Hesslow 2018] L. Hesslow et al., J. Plasma Phys., vol. 84, p. 905840605, 2018
[Hoppe 2018] M. Hoppe et al., Comp. Phys. Comm., vol. 268, p. 108098, 2021
[jorekwiki] JOREK wikipage https://www.jorek.eu/wiki/doku.php?id=particles_runaways
[Sentoku 1998] Y. Sentoku et al., J. Phys. Soc. Jap., vol. 67, p. 4084, 1998
[Sentoku 2008] Y. Sentoku et A.J. Kemp, J. Comp. Phys., vol. 227, p. 6846, 2008
[Pérez 2012] F. Pérez et al., Phys. of Plasmas, vol. 19, p. 083104, 2012
[Higginson 2020] D.P. Higginson et al., J. Comp. Phys., vol. 413, p. 109450, 2020
[Lavell 2024] M.J. Lavell et al., Phys. of Plasmas, vol. 31, p. 043902, 2024
[Zenitani 2022] S. Zenitani et S. Nakano, Phys. of Plasmas, vol. 29, p. 113904