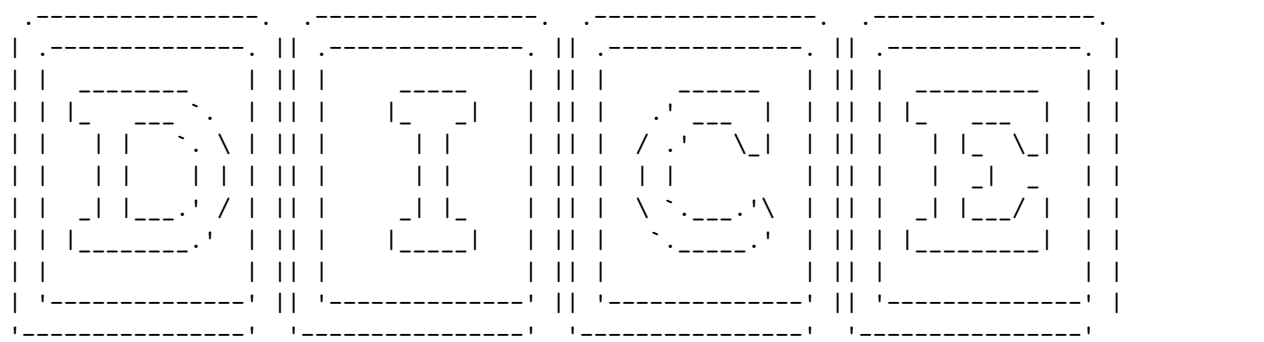
Contents



Version 0.9.45

Author: Thierry Sousbie - tsousbie[a]gmail[dot]com

DICE is a C++ template library designed to solve collisionless fluid dynamics in 6D phase space using massively parallel supercomputers via an hybrid OpenMP / MPI parallelization. This package implements a cosmological and physical VLASOV-POISSON solver for cold systems such as dark matter (CDM) dynamics called ColDICE and based on DICE.

In particular, the following features are implemented in DICE so far:

- An MPI distributed unstructured simplicial mesh that can be dynamically refined and coarsened and supporting many additional features (e.g. dynamic load balancing, serialization, multi-threaded iterators, Peano-Hilbert ordering, high order quadratures, ...)
- A structured AMR (Adaptive mesh refinement) grid.
- An exact AMR grid raytracer (i.e. the sequence of AMR cells intersected by rays is guaranteed to be exact)
- Exact 2D and 3D rasterisation via a fast method to project exactly (i.e. by computing exact integrals instead of resorting to random sampling) a first order function defined over the MPI-distributed mesh onto a structured (AMR) grid.
- A MPI / OpenMP fourier space FFT solver.
- MPI distributed regular grids with support for complex iterators, various interpolation kernels and high order quadratures.
- Many other things ...

Learn more about DICE and ColDICE by reading this scientific article or by visiting the VLASIX project website: http://www.vlasix.org/index.php?n=Main.ColDICE

(I) Installation

The DICE template library itself is located in the `include/dice/` directory and features 3 executables:

- `dice-config`:
  A utility to retrieve the compilation flags, defines and linker flags used when DICE was configured. This is used to make linking with DICE easy.
- `netconv`:
  A utility to convert unstructured mesh outputs to different file formats (e.g. VTK to use with paraview) and unperiodize them.
- `vtrSlicer`:
  A utility to extract slices or project fields from .vtr files (VTK format).

The Cold dark matter fluid dynamics solver ColDICE is composed of 6 main binaries:

- `coldice_2D` and `coldice_3D`:
  The 2D and 3D versions of the vlasov poisson solver
- `coldicePost_2D` and `coldicePost_3D`:
  The 2D and 3D versions of utilities used to load restart files produced by coldice_2D and coldice_3D and post process them.
- `coldiceStatic_2D` and `coldiceStatic_3D`:
  The 2D and 3D versions of the static potential vlasov poisson solver

Requirements

- A recent C++ compiler compatible with C++11 standard (e.g. g++ v4.8.2+ or intel's icpc v11). Clang v7.0.2 (OSX El Capitan) crashes on compilation when OpenMP is enabled !
  SEE NOTES BELOW
- The Eigen library version 3.2.90+ (in external/)
- The Boost library version 1.57+ (the headers suffice, not need to compile it)
- SparseHash libray, already included in the distribution, no need to install it (in external/)
- The FFTW library version 3.3.4+ with optional thread support (if openMP is used) (in external/)
- The GMP library for arbitrary precision arithmetic (you probably have it already) (in external/)
- The Quadruple Double precision library libqd (in external/)
  SEE NOTES BELOW for intel's compiler users

Additional requirements for MPI support:

- A recent version of the MPI library
  SEE NOTES BELOW for intel's MPI library users
- The MPI version of FFTW library version 3.3.4+ (in `external/`)
- The Metis 5.0+ and ParMETIS v4.0+ libraries (in `external/`)

IMPORTANT NOTES:

- OSX users of version up to (and including) El Capitan are strongly advised to use gcc instead of clang. All dependencies also need to be compiled with the same gcc version (i.e. you should not use versions downloaded from brew or fink). This is most conveniently achieved by setting the `CC` and `CXX` environment variable to the corresponding compiler and adding the path to the compiled library in front of the `DYLD_LIBRARY_PATH` variable.

- All necessary libraries can be found in the `external/` sub directory.
- Before installing metis and parmetis, set `IDXTYPEWIDTH 64` in `include/metis.h` and `metis/include/parmetis.h` respectively if required (this is already done for packages in the external directory).
- metis and parmetis can be compiled with openmp, but it is not the case by default (use `make config openmp=1`).
- When using intel compiler, Libqd **MUST** be compiled with `-fp-model precise` flag. Not doing so will result in exact projection algorithm failures !!!!
- When using intel MPI library, the code should be compiled with additional option `-mt_mpi` (use `-DCXX_FLAGS_EXTRA="-mt_mpi"` option with cmake)

Installing

Installing is pretty simple, go to the distribution directory `${DIST_PATH}` and type:

```
cd build/
cmake ../ <options>
```

A report will be displayed with default configuration and indications on how to change options (such as the installation path or path to libraries). Typical options are :

```
-DCMAKE_PREFIX_PATH=<PATH> indicates default path to all libraries
-DNO_MPI=true to compile without MPI support
```

Other options are indicated in the report. Problems occuring during configuration that may need fixing are reported as red lines in the report together with a hint on what should be done. If such a problem occurs, you may optionally rerun cmake with different options (it is often advised to clean the cache first):

```
cd ../; ./cleanup; cd build/
cmake ../ <new options>
```

Once everything is set, the code is built by running:

```
make -j N
```

where N is the number of cores to use in parallel. Upon completion, the resulting binaries are in `${DIST_PATH}/bin`. You can also optionally install the package by running:

```
make install
```

NOTE:
It may sometimes be necessary to clean-up all the files generated by cmake before changing cmake options (e.g. when a library was found but you want to use one from a different location, or when you want to change the compiler). This can be done by running the `cleanup` script from the distribution directory

```
cd ${DIST_PATH}
./cleanup
```

IMPORTANT NOTES:

- If you want to use a specific compiler (such as e.g. mpicc / mpicxx) that is not the default one on your system, you can either set the `CC` and `CXX` environement variables or specify the compiler with the `-DCMAKE_CXX_COMPILER=my_C++_compiler` and `-DCMAKE_C_COMPILER=my_C_compiler` options in `cmake`. Note that the build directory MUST be cleaned before changing the compiler (e.g. using the cleanup script in the root directory).
- When using intel compiler, Libqd *MUST* be compiled with `-fp-model precise` flag. Not doing so will result in exact projection algorithm failures !!!!
- When using intel MPI library, the code should be compiled with additional option `-mt_mpi` (use `-DCXX_FLAGS_EXTRA="-mt_mpi"` option with cmake) to enable multi-threading

Usage

Using DICE in a program is relatively easy as it suffice to include relevant header files in your code. Compiling is also made easy thanks to the `dice-config` utility which can be used to retrieve the compiler options necessary to the proper functioning of DICE. To compile a program `dummy.cpp` using DICE, just do:

```
`${DICE_DIR}/bin/dice-config -all` -I${DICE_DIR}/include dummy.cpp -o dummy
```

where `${DICE_DIR}` is the directory where dice was installed (or the directory where it was compiled if you did not install it). Note that `dice-config` can also list flags, defines and libraries individually for more complex projects.

(II) Documentation

The documentation is located in the `${DIST_PATH}/doc` folder and should contain the following:

- A Doxygen documentation of the library source code can be generated by going to the `doc` subfolder and running `doxygen` from there:

  ```
  cd ${DIST_PATH}/doc
  doxygen .
  ```

  The generated documentation can be viewed with any browser (e.g. by running `firefox html/index.html`).

- A scientific article describing the purpose of and main algorithms in DICE and ColdICE (`article.pdf`). Please reference the published version of this article if you make use of result obtained with this piece of software.

- A `VTK_file_formats.pdf` file containing a description of the output files format. See also http://www.vtk.org/Wiki/VTK_XML_Formats for complementary informations.

- This documentation in PDF format

(III) Vlasov-Poisson solvers: `coldice_xD`

Running:

Parameters for `coldice` programs can be specified directly on the command line preceded by a `-` marker or directly from a parameter file, in which case the parameter file name must be specified as the first argument, without `-` marker. They generaly take the form `<section>.<parameter>` and each parameter can accept one or more arguments, separated by spaces. If a given parameter is not specified, it is given a default value. Information on available parameters, their meaning and default values can be obtained by running:

```
coldice_nD -help
```

or by consulting the appropriate section of this manual. Note however that the list of options given by running the code may be more up-to-date, and it might be convenient to save this documentation to a file by running:

```
coldice_nD -help > parametersDoc.txt
```

A typical output will take the form of a list of alphabetically ordered parameters:

```
(...)
Category: 'mesh'
 ->  mesh.allocFactor = 1
   What fraction of the current number of element to reallocate when a pool is full.

 ->  mesh.delta[0] = 2
     mesh.delta[1] = 2
     mesh.delta[2] = 0
     mesh.delta[3] = 0
   Size of the initial bounding box

 ->  mesh.initPartitionTolerance = 1.01
   Tolerance on the load balance of the initial partition
(...)
```

In this particular example of the 2D version, a parameter `-mesh.delta` that sets the physical size of the initial bounding box is shown to take four arguments at most with default values 2, 2, 0, and 0. New values (e.g. a size of 1 instead of 2) can be specified on the command line as follows:

```
coldice_2D -mesh.delta 1 1
```

where the unspecified remaining two parameter values will take the default value 0.

Note that `-help` option lists only the options available with default parameters and specifying some parameters may unlock new available parameters. For instance, the option `-init.type` specifies the initial conditions type and has default value `sineWaves` with following options:

```
Category: 'init'
 ->  init.amplitude[0] = 0.2
     init.amplitude[1] = 0.2
   Amplitude of the sine waves
```

```
 ->  init.nWaves = 1
   Number of waves along each dimensions

 ->  init.shift[0] = 1
     init.shift[1] = 1
   Sine wave coordinates shift in [0,1]

 ->  init.velFactor[0] = 0
     init.velFactor[1] = 0
   Velocity to displacement ratio

 ->  init.velVector[0] = 0
     init.velVector[1] = 0
   Global velocity vector
```

Changing `-init.Type` to `cosmo` will change these options, which can be unveiled by running `coldice_2D -init.Type cosmo -help` :

```
Category: 'init'
 ->  init.fileName = ics_gadget.dat
   Gagdet file name (without .N extension for multiple files).

 ->  init.initTesselationType = ANY
   Topology of the initial grid simplicial tesselation (ALTERNATE,ANY,MINIMAL,REGULAR,USER_DEFINED)

 ->  init.maxMemPerNode = 1
   The maximum amount of memory to use on each MPI node, expressed in GigaBytes.

 ->  init.rowMajor = 1
   Use row major particle ordering if true (i.e. index of particle changes faster with X coordinate). D
```

Note that a summary of the parameter values that will be used for the current run is always given at the end of the output message. A complete list of available parameters is given at the end of this section.

Examples

A few sample parameters defining initial conditions used in the scientific article `doc/ColDICE.pdf` are given int he `ini/` sub directory:

- `chaos.ini`: Evolution of a small square patch in a 2D static chaotic potential (this must be ran with `coldiceStatic_2D`).
- `sineWaves.ini`: Cosmological collapse of two orthogonal sinusoidal waves.
- `cosmo.ini`: Simulation of structure formation in a Warm dark matter universe (20 h-1 Mpc scale). A sample script used to run this example over 768 cores is available in `scripts/run_cosmo_occigen`. The initial condition file can be generated using MUSIC with the parameter file `ini/cosmo.music`.
- `sineWaves_lowRes.ini`: A low resolution version of `sineWaves.ini` to run on a desktop computer, with regular outputs.

The last example (`sineWaves_lowRes.ini`) in particular can be ran on a modest workstation as follows:

```
coldice_2Dp sineWaves_lowRes.ini -solver.nThreads N -solver.outputDir Dir
```

With `N` the number of openMP threads to use and `Dir` the name of the ouput directory where files will be created (pay attention to the fact that this directory must exist prior to running the code). The program should run for up to a few minutes and produce a bunch of output files that can be visualized with Paraview (http://www.paraview.org).

Note that the phase-space sheet files need to be un-periodized and converted to the `vtu` format first, which can be done with the following command:

```
for i in *.NDnet; do netconv $i -unperiodize -to vtu; done
```

Output files

There are four types of output files :

- `dump` files:
  Outputs of the current state of the unstructured mesh or projected density / potential in a VTK format readable by paraview or convertible to such a format using `netconv`. The VTK file formats description is available in `${DIST_PATH}/doc/VTK_file_formats.pdf` (see also http://www.vtk.org/Wiki/VTK_XML_Formats). The `.NDnet` format specifications is available at the following URL: http://www2.iap.fr/users/sousbie/web/html/indexceff.html?post/NDnet-format
- `restart` files (`.rst`):
  Files used to restart runs, see the following section.
- `statistics.txt`:
  Contains statistics about the current run such as kinetic and potential energy, time step by time step (see header)
- `timings.txt`:
  Timings of several subparts of the computation during the current run. One file is created for each MPI process in the `timings` subdirectory. The meaning of each column is given in the header.

Restarting jobs

Restart files are produced at regular intervals specified by option `-solver.restartEvery <N>`. These restart files can be used to restart a job from were it was stopped with option `-restart <file name without .rst extension>`. When restarting a job, some parameters may be modified such as the resolution of the grid over which the potential is computed but other are forced to take values given in the initial run such as the mesh resolution. A summay of the "managed" parameters values that will be used for the current run is always given as a header in the output of `coldice` before starting computations.

Parameters description

An up-to-date list of parameters can be obtained using the `-help` option, by running `coldice_xD -help` (see list heareafter). Depending on the selected options, new options may be unlocked, these new options can also be listed by running a similar command line of the form `coldice_xD -mycategory.myoption -help`. A list of options given by the command `coldice_xD -help` at the moment this documentation was written follows:

```
Category: 'cosmology'
->  cosmology.h0 = 1
  Hubble parameter (h0 = H0 / 100 kms-1 Mpc-1)

->  cosmology.oB = 0
  Baryonic matter fraction (already included in oM)
```

```
->  cosmology.oL = 0
  Cosmological constant fraction

->  cosmology.oM = 1
  Matter fraction

->  cosmology.w = -1
  Equation of state for dark energy

Category: 'default'
->  debug = 0
  Set for debug mode (value 0 to 2)

->  debugHook = 0
  Starts an infinite loop on startup to allow hooking a debugger

->  debugWait = 0
  Wait for N seconds on startup to allow attaching a debugger

->  globalLogDir =
  The directory where the global log is created

->  globalLogFName = globalLog
  The name of the global log file

->  ignoreUnusedParams = 0
  Do not exit when encountering unknown params

->  initOnly = 0
  Just initialize, report, and exit

->  noGlobalLog = 0
  Set to prevent creation of a global log

->  restart =
  The name of a restart file
  (without the '.rst' or '_${RANK}.rst'(MPI) extension)

->  threads_per_node = -1
  The number of openMP threads to use per MPI node (auto set if negative)

->  verbose = 3
  Verbosity level (from 0=quiet to 4=debug)

Category: 'dump'
->  dump.amrAt =
  Sets AMR grid file dumps time interval (fmt: 'start:stop:every').

->  dump.amrEvery = 0
  Sets AMR grid file dumps time-steps interval
  (dump every N timesteps, never if N negative or 0).
```

```
->  dump.causticsAt =
  Sets caustics surface sub-mesh file dumps time interval.
  Format is 'start:stop:every'.

->  dump.causticsEvery = 0
  Sets caustics surface sub-mesh file dumps time-steps interval.
  Dump every N timesteps, never if N negative or 0.

->  dump.densityAt =
  Sets density grid file dumps time interval (fmt: 'start:stop:every').

->  dump.densityEvery = 0
  Sets density grid file dumps time-steps interval.
  Dump every N timesteps, never if N negative or 0.

->  dump.densityProfileCenter[0] = 0
    dump.densityProfileCenter[1] = 0
  The coodinates of the density profile center. Only used if
  specifyProfileCenter is set.

->  dump.linesAt =
  Sets lagrangian 1D lines sub-mesh file dumps time interval
  (fmt: 'start:stop:every').

->  dump.linesEvery = 0
  Sets lagrangian 1D lines sub-mesh file dumps time-steps interval
  (dump every N timesteps, never if N negative or 0).

->  dump.meshAt =
  Sets mesh file dumps time interval (fmt: 'start:stop:every').

->  dump.meshEvery = 0
  Sets mesh file dumps time-steps interval
  (dump every N timesteps, never if N negative or 0).

->  dump.potentialAt =
  Sets potential grid file dumps time interval
  (fmt: 'start:stop:every').

->  dump.potentialEvery = 0
  Sets potential grid file dumps time-steps interval
  (dump every N timesteps, never if N negative or 0).

->  dump.radialGridDensityAt =
  Sets radial density profile from grid file dumps time interval
  (fmt: 'start:stop:every').

->  dump.radialGridDensityEvery = 0
  Sets radial density profile from grid file dumps time-steps interval
  (dump every N timesteps, never if N negative or 0).

->  dump.radialMeshDensityAt =
  Sets radial density profile from mesh file dumps time interval
  (fmt: 'start:stop:every').
```

```
->  dump.radialMeshDensityEvery = 0
  Sets radial density profile from mesh file dumps time-steps interval
  (dump every N timesteps, never if N negative or 0).

->  dump.reset = 0
  Set this parameters to reset any default file dumps
  (i.e. reset dumps programmed from restart file)

->  dump.specifyProfileCenter = 0
  Profile is centered on the center of the bounding box if false, or use user specified
  coordinates if true (see densityProfileCenter)

->  dump.subsetsAt =
  Sets mesh subsets file dumps time interval (fmt: 'start:stop:every').

->  dump.subsetsEvery = 0
  Sets mesh subsets file dumps time-steps interval (dump every N timesteps, never if N negative or 0).

Category: 'glbDebug'
->  glbDebug.meshRefine = 0
  Set to debug mesh refinement.

Category: 'init'
->  init.type = sineWaves
  The type of initial condition: uniformGrid, sineWaves, phasedWave, cosmo, composite
  Each type of initial condition is associated to a different set of parameters in the 'init' section
  listed by running `coldice_xD -init.type TYPE -help`. The following parameters are for 'sineWaves'
  initial conditions type.

->  init.amplitude[0] = 2
    init.amplitude[1] = 2
  Amplitude of the sine waves

->  init.cosmo = 1
  Whether to use cosmology

->  init.delta[0] = 1
    init.delta[1] = 1
    init.delta[2] = 0
    init.delta[3] = 0
  Size of the initial mesh

->  init.initTesselationType = ANY
  Topology of the initial grid simplicial tesselation (ALTERNATE,ANY,MINIMAL,REGULAR,USER_DEFINED)

->  init.resolution[0] = 64
    init.resolution[1] = 64
  Resolution of the initial mesh (in pixels before tesselation)

->  init.x0[0] = -0.5
    init.x0[1] = -0.5
    init.x0[2] = 0
    init.x0[3] = 0
```

```
  Initial coordinates of the mesh lower left corner


Category: 'mesh'
->  mesh.allocFactor = 1
  What fraction of the current number of element to reallocate when a pool is full.

->  mesh.delta[0] = 1
    mesh.delta[1] = 1
    mesh.delta[2] = 0
    mesh.delta[3] = 0
  Size of the bounding box

->  mesh.initPartitionTolerance = 1.01
  Tolerance on the load balance of the initial partition

->  mesh.initPartitionType = KWAY
  The method to use for initial partitionning (KWAY,PH,PH_KWAY)

->  mesh.refinePartitionType = ADAPTIVE
  The method to use for repartitionning (ADAPTIVE,KWAY,PH,REFINE_KWAY)

->  mesh.repartThreshold = 1.15
  Inbalance factor that triggers repartitionning

->  mesh.repartTolerance = 1.01
  Tolerance on the load balance after repartitionning

->  mesh.x0[0] = -0.5
    mesh.x0[1] = -0.5
    mesh.x0[2] = 0
    mesh.x0[3] = 0
  Initial coordinates of the bounding box lower left corner

Category: 'projection'
->  projection.accuracyLevel = 0.001
  Required accuracy level for projection (DISABLED).
  Use D_ENABLE_ACCURACY_CHECKING compile time option to enable.

->  projection.anisotropyThreshold = 1e+06
  The simplex anisotropy threshold up to which exact projection is used
  (expressed as the maximum to minimum simplex extent ratio)

->  projection.enableHRModeThreshold = 0
  Fraction of the simplices that are allowed to be reprojected before definitively switching to high
  resolution projection mode (between 0 and 1, negative number=>always).

->  projection.resetHRMode = 0
  Ignore whether high resolution mode was set on a previous run and set it to false.
  This is only meaningfull when restarting a run.

->  projection.useVerticesThreshold = -1
  Minimum refinement level of a simplex above which its vertices are used to project it instead of
  exact integration. This should be equal to either '-1' or the value of 'solver.maxSimplexLevel'.
```

```
->  projection.widthThreshold = 1e-07
  The width of an element, expressed as a fraction of the bounding box size, down to which
  exact projection is used.

Category: 'solver'
->  solver.aEnd = 1
  Final value of the scale factor

->  solver.aStart = 0.01
  Initial value of the scale factor

->  solver.cflCondition = CFL_RHOMAX
  CFL condition type (CFL,CFL_RHOMAX,NONE,RHOMAX)

->  solver.cflGrid = 0.25
  Value of the CFL condition (fraction of a FFT grid pixel size)

->  solver.cflRhoMax = 0.01
  Value of the maximum density condition (fraction of an orbit size)

->  solver.checkProjectedDensity = 1
  Check the positivity of the projected density (also look for NaN values).

->  solver.coarsenEvery = 1
  Force coarsening every N timesteps

->  solver.coarsenHysteresis = 0.95
  NOT used

->  solver.cosmo = 1
  Use cosmological expansion ?

->  solver.dLogA = 0.1
  Maximum value for d(log(a))=da/a.

->  solver.dumpInitialMesh = 0
  Dump the mesh right after creating the initial conditions.

->  solver.dumpRestartSignalFileName = DUMP_RESTART
  The name of the file to create in the output directory in order to force the creatioàn of a restart
  file on next time step.

->  solver.exportFftWisdom = wisdom.fftw
  The name of a file to which the acquired FFTW wisdom will be exported. Set to 'SKIP' or 'skip' to
  prevent exportation.

->  solver.fastAmrBuild = 1
  If true, refine the AMR grid so that it only fits roughly the resolution of the mesh. This is
  probably always good enough and much faster ...

->  solver.fftGridLevel = 8
  Size of the FFT grid used for the Poisson solver (effective size is 2^fftGridLevel)
```

```
->  solver.fftWisdom = 1
  Which level of FFTWISDOM to use, in the range {0,1,2,3}={ESTIMATE,MEASURE,PATIENT,EXHAUSTIVE}.
  The higher the faster, but the slower the initialization ...

->  solver.importFftWisdom = wisdom.fftw
  The name of a file from which FFTW wisdom will be imported. Set to 'SKIP' or 'skip' to
  prevent importation.

->  solver.invariantThreshold = 1e-05
  Poincare invariant threshold to trigger refinement

->  solver.mass = 1
  Initial total mass

->  solver.maxAmrLevel = 8
  Maximum refinement level of the AMR grid (don't touch that, it's already set ;) )

->  solver.maxSimplexLevel = -1
  Maximum level of refinement a simplex is allowed to reach.
  (-1  for no maximum, unrefined simplices have level 0)
  See also 'projection.useVerticesThreshold'.

->  solver.nThreads = 4
  The number of threads to use per process. This parameter will override the default 'threads_per_node
  parameter and will be saved from one run to another.

->  solver.noCoarsen = 1
  Set to prevent mesh coarsening

->  solver.noRefine = 0
  Set to prevent mesh refinement

->  solver.noRepartWeight = 1
  If true, MPI regions weight is directly proportional to the number of simplices.

->  solver.noRestart = 0
  Set to prevent from dumping any restart file.

->  solver.outputDir =
  The directory where all files will be put (default is './', directory must exist)

->  solver.phSortThreshold = 0.05
  Maximum allowed ratio of randomly distributed to ordered cells before triggering a Peano-Hilbert
  sort of the local meshes.

->  solver.projectionOrder = 1
  Order of the exact mass projection (only 0 and 1 are implemented so far)

->  solver.rebuildAmrEvery = 1
  How many timestep to wait before entirely rebuilding the AMR grids.
  (IGNORED : not implemented yet!)

->  solver.refineThreshold = 3
  NOT used
```

```
->  solver.resimulate = 0
  Set to start a resimulation after the simulation completes

->  solver.resimulateEvery = 0
  Not implemented

->  solver.restartEvery = 50
  Write a restart file every N timestep (no restart is written if N negative or 0)

->  solver.skipInitialPoisson = 1
  Skips the pre-solving of poisson equation for when the CFL conditon is set to CFL_RHOMAX.
  Instead, measure initial RHOMAX directly on the sheet.

->  solver.splitLongestEdge = 0
  Set to split the longest longest edge when refining instead of trying to minimize the invariant

->  solver.squeezeMesh = 0
  You don't want to know ;)

->  solver.statisticsFileName = statistics.txt
  The name of the file where statistics will be stored

->  solver.stopSignalFileName = STOP
  The name of the file to create in the output directory in order to cleanly stop the run.

->  solver.symmetry = NONE
  Type of symmetry to enforce.
  (AXIAL0,AXIAL01,AXIAL012,AXIAL02,AXIAL1,AXIAL12,AXIAL2,CENTRAL,CONSTANT0,CONSTANT1,CONSTANT2,NONE,
  PLANAR0,PLANAR01,PLANAR012,PLANAR02,PLANAR1,PLANAR12,PLANAR2)
  THIS OPTION DOES NOT WORK PROPERLY WITH MPI YET !

->  solver.timeLimit = -1
  CPU time limit (in hours, will stop before spending it). This is an exact limit (negative for no lim
  See also timeLimitSafety

->  solver.timeLimitSafety = 0.95
  Fraction of the timeLimit while it is safe to keep computing. Leaves (1-timeLimitSafety)*timeLimit
  for writing the restart file.

->  solver.timingsFileName = timings.txt
  The name of the file where timings will be stored

->  solver.volumeThreshold = 1e+10
  Volume threshold to trigger refinement (NOT used)

Category: 'units'
->  units.H = 1
  Hubble parameter (default value in s-1).

->  units.length = 1
  Length unit expressed in meters (default: -).

->  units.mass = 1
```

```
   Mass unit expressed in meters (default: -).

 ->  units.velocity = 1
   Velocity unit expressed in meters (default: -).
```

Initial conditions

There are currently 5 types of initial conditions implemented in the code. Additionally to these initial conditions, it should be relatively easy to implement your own. If you want to do so, check the `IC_dummy.hxx` file provided in the `${DICE_DIR}/solvers/ColDICE/init/` directory.

Currently implemented initial conditions are:

- `uniformGrid`: A flat uniform grid that can be advected by a static potential.
- `sineWaves`: Two orthogonal sinusoidal overdensities. Can be used with physical or cosmological conditions.
- `phasedWaves`: A distorted plane wave. Can be used with physical or cosmological conditions
- `cosmo`: cosmological initial conditions read from a `gadget` type file as generated e.g. with MUSIC (https://people.phys.ethz.ch/~hahn/MUSIC/)
- `composite`: the composition of several grids for simulating e.g. interacting objects. This is quite rudimentary at the moment !

Specific parameters are associated to each type of initial conditions. The following list was obtained by running `coldice_xD -init.type <TYPE> -help`:

`TYPE=cosmo`:

```
 ->  init.fileName = ics_gadget.dat
   Gagdet file name (without .N extension for multiple files).

 ->  init.initTesselationType = ANY
   Topology of the initial grid simplicial tesselation
   (ALTERNATE,ANY,MINIMAL,REGULAR,USER_DEFINED)

 ->  init.maxMemPerNode = 1
   The maximum amount of memory to use on each MPI node, expressed in GigaBytes.
   Default is 1GB per node.

 ->  init.rowMajor = 1
   Use row major particle ordering if true (i.e. index changes faster with X coordinate).
   Default is rowMajor.

 ->  init.x0[0] = 0
     init.x0[1] = 0
   Position of the lower left corner of the box
```

`TYPE=sineWaves`:

```
 ->  init.amplitude[0] = 2
     init.amplitude[1] = 2
   Amplitude of the sine waves

 ->  init.cosmo = 1
```

```
   Whether to use cosmology

 ->  init.delta[0] = 1
     init.delta[1] = 1
     init.delta[2] = 0
     init.delta[3] = 0
   Size of the initial mesh

 ->  init.initTesselationType = ANY
   Topology of the initial grid simplicial tesselation
   (ALTERNATE,ANY,MINIMAL,REGULAR,USER_DEFINED)

 ->  init.resolution[0] = 64
     init.resolution[1] = 64
   Resolution of the initial mesh (in pixels before tesselation)

 ->  init.x0[0] = -0.5
     init.x0[1] = -0.5
     init.x0[2] = 0
     init.x0[3] = 0
   Initial coordinates of the mesh lower left corner

TYPE=phasedWaves

 ->  init.aCross = 0.15
   Time of first shell crossing

 ->  init.delta[0] = 100
     init.delta[1] = 100
     init.delta[2] = 0
     init.delta[3] = 0
   Size of the initial mesh

 ->  init.epsilon = 0.5

 ->  init.initTesselationType = ANY
   Topology of the initial grid simplicial tesselation
   (ALTERNATE,ANY,MINIMAL,REGULAR,USER_DEFINED)

 ->  init.k = 1
   Perturbation parameter

 ->  init.kp = 1
   Plane wave parameter

 ->  init.resolution[0] = 64
     init.resolution[1] = 64
   Resolution of the initial mesh (in pixels before tesselation)

 ->  init.x0[0] = 0
     init.x0[1] = 0
     init.x0[2] = 0
     init.x0[3] = 0
   Initial coordinates of the mesh lower left corner
```

```
TYPE=composite:

 ->  init.add = none
   The type of component to add (only 'box' is available now) . Can be called several times ...

 ->  init.bBoxDelta[0] = 2
     init.bBoxDelta[1] = 2
     init.bBoxDelta[2] = 0
     init.bBoxDelta[3] = 0
   Size of the bounding box

 ->  init.bBoxX0[0] = -1
     init.bBoxX0[1] = -1
     init.bBoxX0[2] = 0
     init.bBoxX0[3] = 0
   Initial coordinates of the bounding box lower left corner
```

(IV) Solvers post-reatment from restart files: `coldicePost_xD`

`coldicePost_xD` programs are similar to the `coldice_xD` ones but are adapted to the post-treatment of restart files produced by `coldice_xD`. They work exactly the same as coldice programs and take the same arguments as `coldice_xD` plus a few additional ones in the "post" section (i.e. of the form `post.<option>`). This program is typically run as:

```
coldicePost_xD -restart <FILENAME without `_XXXX.rst` extension> -post.option...
```

and all options available in the `post` section can be listed by running `coldicePost_xD -help`:

```
Category: 'post'
->  post.dumpAmr = 0
  Whether to dump AMR files.

->  post.dumpDensity = 0
  Whether to dump density grid files

->  post.dumpMesh = 0
  Whether to dump the unstructured mesh files

->  post.dumpRadialGridDensity = 0
  Whether to dump the radial grid density (from projected grid)

->  post.dumpRadialMeshDensity = 0
  Whether to dump the radial mesh density (from the mesh)

->  post.dumpSubmesh = 0
  Whether to dump the unstructured mesh subsets files

->  post.gridLevel = 8
  Sets the resolution of the density grid

->  post.projectionOrder = 1
  Order of the exact mass projection (0 or 1)

->  post.radialMeshDensitySamples = 10
  The number of samples to use per overlap for radial mesh density computation

->  post.refineMesh = 0
  Refine the mesh using tracers.
```

(V) Static potential Vlasov-poisson solvers: `coldiceStatic_xD`

`coldiceStatic_xD` programs are similar to `coldice_xD` ones but work with a static potential for which an analytic solution is known. They work exactly the same as coldice programs and take the same arguments plus a few additional ones in the `staticSolver` section (i.e. of the form `staticSolver.<option>`). These parameters can also be listed with the "-help" option :

```
Category: 'staticSolver'
  ->  staticSolver.gravitationalMass = 1
    Total mass generating the static potential

  ->  staticSolver.potential = PLUMMER
    Static potential type (CHAOTIC,PLUMMER,UDF)
```

and a few parameters are associated to each type of potential, listed in the `potential` section. For instance, the parameters associated to the `CHAOTIC` potential are obtained by runnning `coldiceStatic_xD -staticSolver.potential CHAOTIC -help`:

```
Category: 'potential'
  ->  potential.Rc = 0.2
    Central radius

  ->  potential.epsilon = 0.5
    Perturbation amplitude (epsilon=0 -> regular orbits)

  ->  potential.q = 0.9
    Excentricity
```

(VI) Unstructured mesh output files manipulation : `netconv`

netconv is used to converted unstructured mesh outputs in `.NDnet` format to other formats and/or unperiodizing the mesh for visualization. A short documentation can be obtained by running `netconv -help`, and it is typically used to convert files to the `vtu` format that can be visualized with `paraview` :

```
netconv <NDnet filename> [-unperiodize] -to vtu
```

where the `unperiodize` option can be specified to explicitly break periodic boundary conditions that would otherwise produce visual artefacts.

(VII) Troubleshooting

- An MPI run crashes at "Gathering potential . . . "
  This may be a MPI configuration problem, if you are using openMPI, try setting `export OMPI_MCA_mpi_leave_pinned=0`, or try finding an equivalent option for your MPi implementation.
- You get the "Negative or NaN values found for the projected mass in xxx voxel(s)." ERROR (not the WARNING), followed by a message saying there may be a bug, and the code stops.
  If you are using an Intel compiler, you must compile libQD with the `-fp-model precise` flag enabled (DICE solvers should automatically have it, check the compile flags to make sure). If you get this message only once or twice in a very long MPI run, and the code does not stop, don't worry, the code is running fine.
- You get an error message at startup saying "This MPI implementation does not support any type of multithreading".
  If you are using the Intel compiler, you must add the `-mt_mpi` compile flag to enable a minimal level of multithreading capability (use `-DCXX_FLAGS_EXTRA="-mt_mpi"` option with cmake)