MEDUSA

—

Guide to Coupling

(for SVN revision 346*ff* of MEDUSA)

Guy Munhoven
Université de Liège, Belgium
http://www.astro.ulg.ac.be/~munhoven

21st September 2017

Contents

1 General Overview

1.1 Naming conventions

In the following, the biogeochemical model that MEDUSA is being coupled to will be called the *host model*, denoted `HOST` or `host` in the code parts.

1.2 Your attention, please...

Some sections of text are marked by boxed notices:

- the boxed exclamation mark in the margin points out requirements that are mandatory; **!**
- the boxed "MPI" in the margin marks special requirements that must be taken into consideration when developing a coupled model to be used with an MPI parallel processing infrastructure; MPI
- a boxed number in the margin marks a rare section describing modifications in MEDUSA that require code updates in host models that are coupled to a MEDUSA version with an SVN revision number prior to the indicated one. 301

1.3 Sea-floor grid types and interfacing

Internally, MEDUSA uses a sequential numbering of the sediment columns, whatever their distribution or ordering on the globe, if any. The communication between MEDUSA and the host model therefore generally requires information about the mapping of the host model grids (most often two-dimensional) onto the linear MEDUSA grid and vice-versa

MEDUSA currently includes interfacing subprograms for three different types of ocean grid-point distributions, to be selected by adequate pre-processor directives. The actual geographical ordering of the grid points does not matter here, but only the shape of the grid arrays as declared and used in the ocean model.

1D one-dimensional ordering (default, no pre-processor directive required)

2D two-dimensional grid (use `-DMEDUSA_BASE2D` as a compiler option in the `Makefile` to select this type);

2DT2D two-dimensional array of two-dimensional sub-grids ("tiles") (use `-DMEDUSA_BASE2DT2D` in the `Makefile` to select this type).

Under MPI, one selects the type of grid that a single process will have to process, irrespective of how the sub-grids (tile sets) are globally distributed and organised. MPI

1.4 Time-stepping in MEDUSA

MEDUSA uses an *fully implicit* time-stepping procedure. As a consequence, to perform a time step from t_i to t_{i+1}, it requires the following information for each sediment column (i.e., seafloor grid-point)

- $X(t_i)$ – the model state at time t_i (initial state);
- $P(t_{i+1})$, $S(t_{i+1})$, $T(t_{i+1})$ – depth, salinity, and temperature at time t_{i+1};
- $A(t_{i+1})$ – surface area at time t_{i+1} (optional);
- $C^{\mathrm{f}}(t_{i+1})$ – the boundary conditions for the solutes at the sediment-water interface at time t_{i+1};
- $F^{\mathrm{s}}(t_{i+1})$ – the top solid fluxes (deposition fluxes) at the sediment-water interface at time t_{i+1}.

From these informations MEDUSA then calculates

- $X(t_{i+1})$ – the model state at time t_{i+1} (final state);
- $F^{\mathrm{f}}(t_{i+1})$ – the top solute fluxes (remineralization fluxes) at the sediment-water interface at time t_{i+1}.
- $B^{\mathrm{s}}(t_{i+1})$ – the burial fluxes across the bottom of the sediment-water interface at time t_{i+1}.

The time-stepping scheme of the host model will generally be different from that in MEDUSA: most often explicit, shorter time steps, possibly with variable lengths. In order to collect the forcing data that MEDUSA requires to carry out a time step from time t_i to t_{i+1}, one will thus have to run the biogeochemical host model first from time t_i to t_{i+1} and to prepare the data required by MEDUSA at time t_{i+1}:

- $P(t_{i+1})$, $S(t_{i+1})$, $T(t_{i+1})$ (and $A(t_{i+1})$, if required) – either by averaging over the interval $]t_i, t_{i+1}[$, or by directly using the values at time t_{i+1};
- $C^{\mathrm{f}}(t_{i+1})$ – either by averaging over the interval $]t_i, t_{i+1}[$, or by directly using the values at time t_{i+1};

- $F^{\mathrm{s}}(t_{i+1})$ – for mass conservation reasons, only by averaging over the interval $]t_i, t_{i+1}[$.

It should be noted that the host model's mass balance equations would actually require the remineralization fluxes $F^{\mathrm{f}}(t_{i+1})$ (which are valid for the interval $]t_i, t_{i+1}[$) beforehand. However, these are not available at this stage. Possible solutions to this "chicken and egg" problem are

asynchronous coupling: instead of $F^{\mathrm{f}}(t_{i+1})$, the host model uses $F^{\mathrm{f}}(t_i)$ derived at the previous step, or for the first step, sets it to zero or uses some other estimation procedure (or reads it in from a results file previously generated, as would be typically the case for restart runs);

iterative coupling: using an estimate F_0^{f} of the unknown $F^{\mathrm{f}}(t_{i+1})$, the host model makes a first trial integration from time t_i to t_{i+1}. From the derived boundary conditions at time t_{i+1}, MEDUSA calculates a revised F_1^{f} that the host model then uses to repeat the integration from time t_i to t_{i+1}. The procedure is repeated until the results reach satisfactory converge. $F^{\mathrm{f}}(t_{i+1})$ is then set to the last F_n^{f} calculated.

The examples provided below assume that *asynchronous coupling* has been chosen.

2 Coupling procedure: for the impatient

2.1 Build the MEDUSA code

Dress the list of components that need to be included in MEDUSA, together with their physical and chemical characteristics and generate the codes for MEDUSA.

2.2 Host communication modules

Prepare appropriate versions of `MOD_HOST_O2S` and `MOD_HOST_S2O`, as outlined in the templates

`mod_host`*`TT`*`_o2s_template.F`
`mod_host`*`TT`*`_s2o_template.F`

in the directory `src-med/gen/templates` after the code generation is complete. *`TT`* stands for either `1D`, `2D` or `2DT2D`, as adequate for the application being developed. Do not forget to rename the modules and their source files by replacing any reference to "`HOST`" or "`host`" by an acronym or name related to your application.

2.3 Develop a MEDUSA setup module

Prepare the subroutine `SETUP_MEDUSA_FOR_HOST` to perform the MEDUSA setup (i.e., prepare and format the information required for the adequate version of `SETUP_CENTRAL`). For convenience, we recommend to have it contained in a module that we will name `MOD_HOST_MEDUSA_SETUP`. Such a module can be prepared by following the instructions given in the templates `mod_host`*TT*`_medusa_setup.F_template`, which can be found in the directory `src-med/gen/templates` after the code generation is complete (*TT* stands again for either `1D`, `2D` or `2DT2D` – please select the version appropriate for your application).

2.4 Extend the host model's biogeochemical module

Different modifications are required in the biogeochemical module of the host model:

1. insert `USE` clauses to give access to the arrays in the two communication modules

   ```
   USE MOD_HOST_O2S
   USE MOD_HOST_S2O
   ```

2. include code to calculate the averages required in `MOD_HOST_O2S`;

3. amend the conservation equations in the host model to include source terms in the bottom grid-boxes for the return fluxes from the sediment to the ocean, returned by the arrays in `MOD_HOST_SO2` — beware that fluxes in MEDUSA are positive downwards (into the sediment);

4. amend the mass balance equations to include source terms in surface grid-boxes (most realistically along the coastlines) for the "riverine input", upon which the global sediment burial rate is going to adjust – for the calibration stage, they can be set such that they globally match the sediment burial rate of the sediment, i.e., the solid accumulation rate at the bottom of the sedimentary mixed layer, given by the `seafloor_bfflx_*` arrays in `MOD_HOST_S2O`, which must be converted to the equivalent solute fluxes;

5. disable any remineralization closure that may be present for the seafloor solid fluxes

2.5 Extend the main program or develop a MEDUSA wrapper

In the main program, or in a dedicated wrapper, please insert code for the following tasks, in that order.

- Use clauses for the required modules

```
USE MOD_DEFINES_MEDUSA
USE MOD_EXECONTROL_MEDUSA
USE MOD_PROCESSCONTROL
USE MOD_EQUILIBCONTROL
USE MOD_SEDCORE
USE MOD_SEAFLOOR_INIT

USE MOD_FILES_MEDUSA
USE MOD_HOST_MEDUSA_SETUP
USE MOD_HOST_O2S
USE MOD_HOST_S2O
```

  The first block refers to modules from `libmedusa.a`, the MEDUSA library; the four modules from the second block have to be developed on purpose for your application. Templates for all of them can be found in `src-med/templates` or under `src-med/gen/templates`.

- Initialize the MEDUSA execution environment. This is currently **only required** if the model is being compiled for use **under MPI**. If the coupled model is not compiled for MPI, proceed to the next step; for multi-purpose use, the code can be encapsulated by `#ifdef ALLOW_MPI` and `#endif` pre-processor directives. MPI

  One of two options must be chosen:

  1. if MPI has not yet been initialized, one must use

     ```
     CALL MEDEXE_MPI_INIT()
     ```

  2. if MPI has already been initialized in the host program, the MPI communicator that controls the operation of MEDUSA must be registered in MEDUSA's execution environment, using the optional argument `k_mpi_comm_host`, which must hold the ID of the communicator (`MPI_COMM_WORLD` or the one reserved for MEDUSA)

     ```
     CALL MEDEXE_MPI_INIT(k_mpi_comm_host)
     ```

- Retrieve the list for the files to be read in and written out

```
CALL INIT_FILELIST_MEDUSA()
```

- Set up the MEDUSA central system, by calling the setup subroutine from the previously developed module `MOD_HOST_MEDUSA_SETUP`:

  ```
  CALL SETUP_MEDUSA_FOR_HOST(..., n_columns)
  ```

  The argument list of `SETUP_MEDUSA_FOR_HOST` depends on the actual application, and on which information can be derived directly inside the subroutine, and which information has to be transmitted by argument. However, such a subroutine must always return the number of sediment columns actually detected in the argument `n_columns` (`INTEGER`). This information is required at a later stage.

- Initialize the equilibrium and rate law parameterizations

  ```
  CALL InitEquilibParameters
  CALL InitProcessParameters
  ```

- Set up and initialize the MEDUSA core system

  ```
  CALL SETUP_SEDCORE_SYSTEM(
 &     cfn_ncin_sedcore, cfn_ncout_sedcore)
  ```

  Note that the two filename variables involved in the previous command are declared in `MOD_FILES_MEDUSA`.

- Set up `MOD_HOST_O2S` and `MOD_HOST_S2O`, and initialize the arrays for the sediment-to-ocean fluxes in `MOD_HOST_S2O` to zero:

  ```
  CALL SETUP_MOD_HOST_O2S()
  CALL SETUP_MOD_HOST_S2O()
  CALL CLEAR_S2O_DATASET()
  ```

- Initialize the MEDUSA central system:

  ```
  IF (cfn_ncin_init /= "/dev/null") THEN
    CALL InitSeafloorFromNetCDFFiles(cfn_ncin_init
 &                                   cfn_ncin_flx)
    IF (cfn_ncin_flx /= "/dev/null") THEN
      CALL SEDIMENT_TO_OCEAN(...)
    ENDIF
  ELSEIF (cfn_nmlin_init /= "/dev/null") THEN
    CALL InitSeafloorFromNamelistFile(cfn_nmlin_init)
  ELSE
  ```

```
  ! No valid initialization file found.
  ... ! Print out adequate error messages.
  CALL ABORT_MEDUSA()
ENDIF
```

- Open the results files:

  ```
  CALL OPEN_NCFILES_MEDUSA(atime)
  ```

  where `atime` is a `DOUBLE PRECISION` argument that should hold the initial time of the simulation experiment.

After these operations, MEDUSA is ready to start its calculations.

For each sediment time step running, say, from t_i to t_{i+1}, a series of operations must be carried out, one block before the host biogeochemical module carries out that time step and one block thereafter. Please insert the following code lines inside the time loop for the sediment steps in the main program (or wrapper), in that order:

- Set the time and time-step length for MEDUSA

  ```
  atime  = ... ! t_i [yr]
  datime = ... ! t_{i+1} - t_i [yr]
  ```

- Reset the arrays in `MOD_HOST_O2S` so that the host biogeochemical model can use them to accumulate the data that it must provide to MEDUSA:

  ```
  CALL CLEAR_O2S_DATASET()
  ```

- Integrate the biogeochemical module of the host model over the time interval t_i to t_{i+1}: the biogeochemical module prepares the data required for MEDUSA to cover the same time step afterwards and stores them in `MOD_HOST_O2S`.

- Transfer the contents of `MOD_HOST_O2S` into `MOD_SEAFLOOR_CENTRAL`

  ```
  CALL OCEAN_TO_SEDIMENT(...)
  ```

- Call MEDUSA's solver `SOLVSED_ONESTEP` to perform the sediment time step from time `atime` to `atime + datime`

```
      CALL SOLVSED_ONESTEP(atime, datime, n_columns,
     &                     iflag, n_trouble)

      atime = atime + datime
```

Notice: the time variable `atime` must only be updated after the call to `SOLVSED_ONESTEP` completes.

- Clean up and purge the MEDUSA core system

```
      CALL REACLAY_X_CORELAY(atime)
```

- If required, write out the sediment state to file

```
      CALL WRITERES_NCFILES_MEDUSA(atime)
```

- Prepare the next time step: transcribe the flux values just calculated by MEDUSA from `MOD_SEAFLOOR_CENTRAL` to `MOD_HOST_S2O`, where the host biogeochemical module can retrieve them.

```
      CALL SEDIMENT_TO_OCEAN(...)
```

After the last time step completes (i.e., behind the time loop):

- Close and finalize the results files:

```
      CALL CLOSE_RESFILES_MEDUSA
```

- Finalize the MEDUSA execution environment. This is currently only required under MPI: MPI

```
      CALL MEDEXE_MPI_FINALIZE()
```

3 Coupling procedure: step by step

The typical sequence of operations to carry out in the main program of an ocean biogeochemistry model that is coupled to MEDUSA is outlined in the template `host-codeblocks.F_template`, which is located in the directory `src-med/templates`. That file contains sample blocks of code to insert into that main program or to use for the development a dedicated wrapper subroutine.

3.1 Generate the code for MEDUSA

Dress the list of components that need to be included in MEDUSA, together with their physical and chemical characteristics and generate the codes for MEDUSA.

3.2 Initial set-up: execution environment and file I/O

3.2.1 Initializing the execution environment (MPI only)

The execution environment is controlled by `MOD_EXECONTROL_MEDUSA`. For MEDUSA versions that are not compiled for use with MPI, there are no explicit steps to carry out – the execution environment is completely set by static parameters. You may proceed immediately to section 3.2.2 if you do not plan to use MEDUSA under MPI.

If MEDUSA is going to be used under MPI, it is necessary to initialize its execution environment, otherwise, it will not work correctly. This initialization must be done by calling the subroutine `MEDEXE_MPI_INIT` from `MOD_EXECONTROL_MEDUSA`. Two cases must be distinguished: MPI

1. if MPI has not yet been initialized, one must use

   ```
   CALL MEDEXE_MPI_INIT()
   ```

   without any argument;

2. if MPI has already been initialized at an earlier stage in the main program, the MPI communicator that controls MEDUSA must be registered in `MOD_EXECONTROL_MEDUSA` and communicated via the optional argument `k_mpi_comm_host`:

   ```
   CALL MEDEXE_MPI_INIT(k_mpi_comm_host)
   ```

   `k_mpi_comm_host` may be the default communicator `MPI_COMM_WORLD` or some other special communicator dedicated to MEDUSA, but not `MPI_COMM_NULL`.

When `MEDEXE_MPI_INIT` is called without the optional `k_mpi_comm_host`, MEDUSA's execution control environment also takes complete control of MPI: `MEDEXE_MPI_INIT` calls `MPI_INIT` to launch the initialization of the MPI environment and a later call of `MEDEXE_MPI_FINALIZE` will also call `MPI_FINALIZE` to deactivate MPI. With the argument, `MEDEXE_MPI_INIT` assumes that the MPI environment has already been initialized and the communicator is only

registered; a later call to MEDEXE_MPI_FINALIZE will not finalize the MPI environment (i.e., it will not call MPI_FINALIZE), but only reset the MPI-related information in MOD_EXECONTROL_MEDUSA.

3.2.2 MEDUSA file I/O

MEDUSA I/O involves a series of files that can be used either to initialize the state of the model or to store the evolution of both the sedimentary mixed-layer (REACLAY domain) state, of the underlying transition buffer layer (TRANLAY domain) and the deeper core layers (CORELAY domain).

Although the administration of these files can be implemented in an *ad hoc* fashion, it is recommended to use the interface defined in the module MOD_FILES_MEDUSA from mod_files_medusa.F_template (in the directory src-med/templates). For the rest of this guide, it will be assumed that this MOD_FILES_MEDUSA is used.

That template can be used "as is" or extended to more special needs (please do not delete anything from that module, as other templates depend on it). MOD_FILES_MEDUSA provides a public list of all the files that are considered in MEDUSA. The files in the list are organized in two namelists

- nml_cfg, which collects the files that are common to all versions and configurations of MEDUSA (plus a title to be included in all the NETCDF files); nml_cfg **must not** be modified. !
- nml_extra, which collects the files are specific for a given application; any additions or modifications required should be brought in here.

MOD_FILES_MEDUSA also provides the following subroutines to perform recurrent operations

- SUBROUTINE INIT_FILELIST_MEDUSA() to initialize the list from the file given by the parameter cpfn_medusa_files in MOD_FILES_MEDUSA (pre-set to "medusa_files.cfg" — can be changed) and to print out a summary initial report of the files used (with their names) or not used. A template for such a medusa_files.cfg file can be found in medusa_files.cfg_template (in the directory src-med/templates).

  Under MPI, the file given by cpfn_medusa_files is only read in by the master (root) process, which broadcasts the resulting list contents to all other processes. Accordingly, only the master process requires access to it. MPI

- SUBROUTINE OPEN_NCFILES_MEDUSA(atime) to open the NETCDF results files that have names different from "/dev/null" and write out the initial state at time atime (DOUBLE PRECISION, INTENT(IN))

- `SUBROUTINE WRITERES_NCFILES_MEDUSA(atime)` to write out the relevant records for time `atime` (`DOUBLE PRECISION, INTENT(IN)`) to the open NETCDF files (except for the SEDCORE file)

- `SUBROUTINE CLOSE_NCFILES_MEDUSA()` to close all the open NETCDF files, including the SEDCORE file.

Please notice that, under MPI, I/O operations should only be performed after MPI has been initialized. This is a generally valid recommendation for MPI – I/O operations on files in an MPI program may lead to unpredictable behaviour before `MPI_INIT` has been called. Accordingly, the subroutines from `MOD_FILES_MEDUSA` should only be called after MPI is set up

MPI

A consistent list with the required file names can then be retrieved by

```
CALL INIT_FILELIST_MEDUSA()   ! From mod_files_medusa.F
```

Whatever the chosen method, the list of files to be used by MEDUSA must be initialized before the next setup stages, since the setup partially depends on the combination of I/O files that are requested.

!

3.3 Setting up MEDUSA's central and core systems

There are two key modules in MEDUSA that need to be set up now that the list of input and output files is known:

`MOD_SEAFLOOR_CENTRAL` hosts the *central system*. MEDUSA's numerical solver `SOLVSED_ONESTEP` fetches the data that are required to perform one time step for all the sediment cores considered (initial state of the sediment, boundary conditions, forcings, sediment grid-point distributions, etc.) from `MOD_SEAFLOOR_CENTRAL` and also stores the state of the sediment and the resulting fluxes upon completion of each time step there. `MOD_SEAFLOOR_CENTRAL` also holds the remapping information for the communication between the host model and MEDUSA

`MOD_SEDCORE` hosts the *core system*. Here a stack or synthetic core of buried sediment layers is kept for each sediment column considered, so that chemical erosion can be taken into account.

The variables that reflect the global configuration of MEDUSA now need to be initialized, and the storage arrays allocated and shaped before the rest of the setup can be done.

3.3.1 Setting up MOD_SEAFLOOR_CENTRAL

The configuration of MOD_SEAFLOOR_CENTRAL can only be done by calling the SEAFLOOR_SETUP subroutine provided in MOD_SEAFLOOR_CENTRAL itself. The argument list of SEAFLOOR_SETUP depends on the selected grid type.

Type 1D is the simplest case, since MEDUSA uses itself internally a one-dimensional ordering of the sediment columns. The argument list of the subroutine SEAFLOOR_SETUP is accordingly simple and straightforward. The order and names (for keyword-based calling) of the dummy arguments are as follows:

```
 SUBROUTINE SEAFLOOR_SETUP(
&     n_columns_host, darea_gridelts_host,
&     dcnpoh_c_host, dcnpoh_n_host, dcnpoh_p_host,
&     dcnpoh_o_host, dcnpoh_h_host, dcnpoh_remin_o2_host)

 INTEGER, INTENT(IN)
&  :: n_columns_host
 DOUBLE PRECISION, DIMENSION(:), INTENT(IN)
&  :: darea_gridelts_host
 DOUBLE PRECISION, DIMENSION(:, :), OPTIONAL, INTENT(IN)
&  :: dcnpoh_c_host, dcnpoh_n_host, dcnpoh_p_host,
&     dcnpoh_o_host, dcnpoh_h_host, dcnpoh_remin_o2_host
```

where

- n_columns_host sets the number of grid elements that MEDUSA should consider; all of the grid points are supposed to be sea-floor grid-elements;
- darea_gridelts_host is a 1D-array with n_columns_host elements giving the surface areas [m^2] of the grid-elements;
- dcnpoh_c_host, dcnpoh_n_host, dcnpoh_p_host, dcnpoh_o_host, dcnpoh_h_host, dcnpoh_remin_o2_host are 2D arrays giving the elemental molar C, N, P, O, H and O_2 remineralization ratios for organic matter (sometimes called Redfield ratios). As declared above, all of these six are optional; more precisely, all of them must be present in the call, or none of them: if none of them is present in the call, the default ratios from MOD_MATERIALCHARAS are used instead. The length of the first dimension of all of these 2D arrays must be equal to the number of organic matter components included in MEDUSA; the length of the second dimension must

- either be equal to 1, in which case the values are used for all grid elements,
- or equal to `n_columns_host`, in which case each grid element may have its own combination.

Type 2D is more complex. The order, types, names and ranks of the dummy arguments are as follows:

```
 SUBROUTINE SEAFLOOR_SETUP(imask_ocean_host,
&     dixref_gridelts_host, djyref_gridelts_host,
&     darea_gridelts_host,
&     dcnpoh_c_host, dcnpoh_n_host, dcnpoh_p_host,
&     dcnpoh_o_host, dcnpoh_h_host, dcnpoh_remin_o2_host,
&     n_columns)

 INTEGER, DIMENSION(:,:), INTENT(IN)
&  :: imask_ocean_host
 DOUBLE PRECISION, DIMENSION(:,:), INTENT(IN)
&  :: dixref_gridelts_host, djyref_gridelts_host
 DOUBLE PRECISION, DIMENSION(:,:), INTENT(IN)
&  :: darea_gridelts_host
 DOUBLE PRECISION, DIMENSION(:, :,:), OPTIONAL, INTENT(IN)
&  :: dcnpoh_c_host, dcnpoh_n_host, dcnpoh_p_host,
&     dcnpoh_o_host, dcnpoh_h_host, dcnpoh_remin_o2_host

 INTEGER, INTENT(OUT)
&  :: n_columns
```

where

- `imask_ocean_host` is used as a mask to define the ocean grid-points: elements of `imask_ocean_host` that are strictly positive are considered to be ocean points, elements of `imask_ocean_host` that are zero or negative are filtered out. Different positive values can be used to subdivide the ocean domain into sub-domains for further processing or diagnostics.
- `dixref_gridelts_host` and `djyref_gridelts_host` must have the same shape as `imask_ocean_host` and are used to specify some geographical coordinate information (e.g., x-y, longitude-latitude) for the grid-elements; both arrays are only used for information purposes (included, e.g., in the file produced by `STORE_NC_AUX`).

- `darea_gridelts_host` should give the surface areas [m^2] of the grid elements and must have the same shape as `imask_ocean_host`. These are generally only used for diagnostic purposes and such as global mass balances, but could possibly be required for the data transfer between the host model and MEDUSA. The operation of MEDUSA itself does not depend on this argument.

- `dcnpoh_c_host`, `dcnpoh_n_host`, `dcnpoh_p_host`, `dcnpoh_o_host`, `dcnpoh_h_host`, `dcnpoh_remin_o2_host` have the same signification as for the 1D type above. They must fulfil the same conditions regarding the first dimension and the "all or none" presence at call. Similar to the 1D case, the lengths of the last two dimensions must either be equal to 1 for all of the six arrays, in which case all of the grid elements use the same elemental ratios, or they must be equal to the shape of `imask_ocean_host`, in which case the organic matter reaching the sea-floor can have a different elemental composition at each sea-floor grid element.

- `n_columns` provides, upon return from the call, the actual number of sea-floor grid-elements deduced from `imask_ocean_host`.

The ocean grid points are ordered in a one-dimensional way for internal usage, and correspondence tables are dimensioned (allocated) and initialized.

Type 2DT2D is analogous to type 2D. By order and by name, the arguments are exactly the same as for type 2D; they differ, however, by their shapes:

```
 SUBROUTINE SEAFLOOR_SETUP(imask_ocean_host,
&     dixref_gridelts_host, djyref_gridelts_host,
&     darea_gridelts_host,
&     dcnpoh_c_host, dcnpoh_n_host, dcnpoh_p_host,
&     dcnpoh_o_host, dcnpoh_h_host, dcnpoh_remin_o2_host,
&     n_columns)

 INTEGER, DIMENSION(:,:, :,:), INTENT(IN)
&  :: imask_ocean_host
 DOUBLE PRECISION, DIMENSION(:,:, :,:), INTENT(IN)
&  :: dixref_gridelts_host, djyref_gridelts_host
 DOUBLE PRECISION, DIMENSION(:,:, :,:), INTENT(IN)
&  :: darea_gridelts_host
 DOUBLE PRECISION, DIMENSION(:, :,:, :,:),
```

```
&  OPTIONAL, INTENT(IN)
&  :: dcnpoh_c_host, dcnpoh_n_host, dcnpoh_p_host,
&     dcnpoh_o_host, dcnpoh_h_host, dcnpoh_remin_o2_host

 INTEGER, INTENT(OUT)
&  :: n_columns
```

where

- `imask_ocean_host` is used as a mask to define the ocean grid-points as for type 2D. With type 2DT2D it has, however, a rank of 4 and the shape `(/nix, njy, nsx, nsy /)`, where `nix` and `njy` define the extension of each single 2D tile along the x and the y direction, respectively, while `nsx` and `nsy` define how many tiles are bundled together along the x and y directions, respectively.

- `darea_gridelts_host`, `dixref_gridelts_host`, `djyref_gridelts_host` have the same signification and must fulfil the same constraints as their 2D counterparts;

- `dcnpoh_c_host`, `dcnpoh_n_host`, `dcnpoh_p_host`, `dcnpoh_o_host`, `dcnpoh_h_host`, `dcnpoh_remin_o2_host` have the same signification as their 2D counterparts above, and must fulfil the same conditions regarding the first dimension as well as the "all or none" optional characteristic. Similar to the 2D case, the six arrays must all have the same shape. The lengths of the last four dimensions must either all be equal to 1, in which case all of the grid elements use the same elemental ratios, or they must match the shape of `imask_ocean_host`, in which case the organic matter reaching the sea-floor can have different elemental compositions at each sea-floor grid element.

- `n_columns` provides again the actual number of sea-floor grid-elements deduced from `imask_ocean_host`, upon return from the call.

The assembly and formatting of the data required by `SEAFLOOR_SETUP` can be tedious and unnecessarily clutter up the main program or the wrapper written for the purpose of coupling. It is therefore recommended to have a separate subroutine perform this task. Upon completion of the MEDUSA code generation procedure, `src-med/gen/templates` contains templates in the files `mod_host`*TT*`_medusa_host.F_template` (where *TT* stands for `1D`, `2D` or `2DT2D`) for a module called `MOD_HOST_MEDUSA_SETUP` that contains such a subroutine called `SETUP_MEDUSA_FOR_HOST`.

In the following, it is assumed that this module-subroutine based approach is adopted.

Under MPI, the border of each process' sub-grid may possibly contain one or several lines and columns of overlap, so that each process has access to the variable values of all of the neighbours of the grid-points that it processes, even for those grid-points processed by other process, to be able to complete the adopted discretization stencils. In MEDUSA, all sea-floor grid-points are isolated from each other. Such overlaps are not required, and may even lead to confusion. They should therefore be marked as "non-ocean" points in `imask_ocean_host` with the 2D and 2DT2D types (by negative or zero values) and filtered out beforehand with the 1D type.

MPI

3.3.2 Initializing MEDUSA's parameter values

After `MOD_SEAFLOOR_CENTRAL` has been set up, the equilibrium and rate law parameters can be initialized:

```
CALL InitEquilibParameters    ! From mod_equilibcontrol.F
CALL InitProcessParameters    ! From mod_processcontrol.F
```

These two subroutines are provided by stock MEDUSA modules and it is mandatory to use them.

!

3.3.3 Setting up `MOD_SEDCORE`

Starting from rev. 301, the setup procedure of `MOD_SEDCORE` has completely changed. The subroutines `SEDFILE_NOFILE` and `SEDFIL_OPEN` are not available any more. The new `SETUP_SEDCORE_SYSTEM` from `MOD_SEDCORE` now completely controls the way the sedimentary layer buffer and files are configured and initialized:

301

```
 SUBROUTINE SETUP_SEDCORE_SYSTEM(
&     cfn_ncin_sedcore, cfn_ncout_sedcore)

 CHARACTER(LEN=*), INTENT(IN), OPTIONAL
&  :: cfn_ncin_sedcore, cfn_ncout_sedcore
```

where

- `cfn_ncin_sedcore` can be used to specify the name of a NETCDF file for reading in a set of historical core layers (for restart, or buffer initialization);

- `cfn_ncout_sedcore` can be used to specify the name of the NETCDF file to store the historical layers;
- both arguments are optional; default values are `"/dev/null"`.

Setting a file name explicitly to default or implicitly accepting the default value means that the corresponding file must not be used.

The way `MOD_SEDCORE` will be using the memory layer stack (the stack of sedimentary layers underneath the mixed layer) depends on the chosen combination of and values of `cfn_ncin_sedcore` and `cfn_ncout_sedcore`:

- If `cfn_ncout_sedcore` is set to default, the model uses a purely internal core layer stack and its contents are lost upon completion of the simulation experiment; if `cfn_ncin_sedcore` is also set to default, the model simulation starts with an empty core layer stack, otherwise the contents of `cfn_ncin_sedcore)` are used to initialize the core layer stack.
- If `cfn_ncout_sedcore` is not set to default, the internal core layer stack is used as a buffer and its contents are written to a new file whose name is given by `cfn_ncout_sedcore`; any existing file with the same name will be overwritten.
  If at the same time `cfn_ncin_sedcore` is set to default, the model simulation starts with an empty core layer stack and an empty output file, else two options are offered:
  1. if `cfn_ncin_sedcore` and `cfn_ncout_sedcore` point to different files, `cfn_ncin_sedcore` is scanned and the characteristics of its top layers are transcribed to the new file `cfn_ncout_sedcore` to start the model simulation experiment;
  2. if `cfn_ncin_sedcore` and `cfn_ncout_sedcore` point to the same file, then the results of the forthcoming simulation experiment will be appended to that file.

Please notice that a file whose name is given in `cfn_ncin_sedcore` must already exist, otherwise the model aborts.

3.4 Data exchange between the host and MEDUSA

As outlined above, the execution of one MEDUSA time step involves the preparation (e.g., averaging), transformation (e.g., unit changes) and exchange of a number of data arrays with the host model. There are a number of ways

to implement the related work flow. The following one has proven to offer a good compromise between ease of use and clarity:

- Develop a module `MOD_HOST_MEDUSA_O2S` that contains
  - public arrays (following the 1D, 2D, or 2DT2D grid ranks and shapes) where the host model can store the data required by MEDUSA as inputs;
  - a subroutine, typically called `OCEAN_TO_SEDIMENT` to transfer the data held in the arrays of the module to `MOD_SEAFLOOR_CENTRAL`, taking care of any remapping and unit conversion if necessary;
- Develop a module `MOD_HOST_MEDUSA_S2O` that contains
  - public arrays (following the 1D, 2D, or 2DT2D grid ranks and shapes) where the host model can retrieve the sediment-to-ocean fluxes that MEDUSA provides as output;
  - a subroutine, typically called `SEDIMENT_TO_OCEAN`, to transfer the results from `MOD_SEAFLOOR_CENTRAL` to the arrays of the module, taking care of any remapping and unit conversion if necessary.

N.B.: it is of course possible to merge both modules into a single one, which one might call `MOD_HOST_MEDUSA_OXS`, e.g.

After the MEDUSA code generation is complete, there are template files `mod_host`*TT*`_o2s_template.F` and `mod_host`*TT*`_s2o_template.F` in the directory `src-med/gen/templates`, where *TT* stands for either `1D`, `2D` or `2DT2D`, as adequate for the application being developed. The template modules also include additional subroutines to set up the module (to allocate the array space, etc.), to reset the arrays to zero, etc.

Once the MEDUSA central and core systems have been set up (previous section), it is time to set up those communication modules. If the interface outlined in the templates is adopted, the following code will carry out the required operations

```
CALL SETUP_HOST_O2S()      ! From MOD_HOST_O2S
CALL SETUP_HOST_S2O()      ! From MOD_HOST_S2O
CALL CLEAR_S2O_DATASET()   ! From MOD_HOST_S2O
```

The last command furthermore sets the arrays in `MOD_HOST_S2O` to zero, so that the host biogeochemical model can run with meaningful sediment-to-ocean flux values before MEDUSA. At the initialization stage, the arrays from `MOD_HOST_S2O` may be updated from a previously generated NETCDF file, which may be required for, e.g., restart runs (see section 3.5.2 below).

3.5 Defining the initial state of MEDUSA

The initial state of MEDUSA can of course be completely controlled by hand, using the interface routines `SAVE_COLUMN` from `MOD_SEAFLOOR_CENTRAL`. For convenience, there is a module `MOD_SEAFLOOR_INIT`, which provides two standard procedures to initialize MEDUSA: by namelist, and from NETCDF files generated in a previous simulation experiment.

3.5.1 Initialisation by NAMELIST (basic)

A basic but easy to use method for defining the initial state of MEDUSA is via a namelist file. A template for such a namelist file can be found in `medusa_seafloor_init.nml_template`. All columns in the model are then homogeneously initialized with the data provided in that namelist file. The subroutine `InitSeafloorFromNamelistFile` from `MOD_SEAFLOOR_INIT` does this:

```
SUBROUTINE InitSeafloorFromNamelistFile(cfn_seafloorinit)

CHARACTER(LEN=*), INTENT(IN) :: cfn_seafloorinit
```

3.5.2 Initialisation from NETCDF files

The namelist based initialisation does not offer sufficient flexibility. Therefore, `MOD_SEAFLOOR_INIT` furthermore offers the possibility to initialise the sediment state from a previously generated NETDCF REACLAY file with the subroutine `InitSeafloorFromNetCDFFiles`:

```
 SUBROUTINE InitSeafloorFromNetCDFFiles(
&     cfn_reaclay, cfn_flx, i_rec)

 CHARACTER(LEN=*), INTENT(IN)           :: cfn_reaclay
 CHARACTER(LEN=*), INTENT(IN), OPTIONAL :: cfn_flx
 INTEGER         , INTENT(IN), OPTIONAL :: i_rec
```

where

- `cfn_reaclay` holds the name of the NETCDF REACLAY file to read from;
- `cfn_flx` optionally sets the name of the NETCDF FLX file to initialize the fluxes from: this may be required for coupled simulation experiments that are restarted, and where the host model runs with the sediment remineralization fluxes from time step $[t_{i-1}, t_i]$ to carry out time

step $[t_i, t_{i+1}]$; if `cfn_flx` is not present in the call, or if it has the default value `"/dev/null"`, this step is ignored. The flux values are read into `MOD_SEAFLOOR_CENTRAL` from where they still need to be retrieved to be made available to the host model (by the `SEDIMENT_TO_OCEAN` subroutine from `MOD_HOST_S2O` if the recommended code organization is followed) – see example code in section 3.5.3 below.

- `i_rec` optionally allows to select a particular record in the files (the same in both files); by default, the last record in each file is read.

`InitSeafloorFromNetCDFFiles` must not be called with `cfn_reaclay` set to `"/dev/null"` or some non existing file.

3.5.3 Practical recommendation

In practice, for the MEDUSA applications developed so far, the convention has been adopted that, if both the namelist file `cfn_nmlin_init` and the NETCDF file `cfn_ncin_init` have non default values, the NETCDF file overrides the namelist file. The `INIT_FILELIST_MEDUSA()` subroutine enforces this convention strictly, by resetting `cfn_nmlin_init` to default (`"/dev/null"`) if `cfn_ncin_init` does not have the default value. The following piece of code can then be used universally in the main program or the wrapper to initialize MEDUSA:

```
 IF (cfn_ncin_init /= "/dev/null") THEN
   CALL InitSeafloorFromNetCDFFiles(cfn_ncin_init
&                                   cfn_ncin_flx)
   IF (cfn_ncin_flx /= "/dev/null") THEN
     CALL SEDIMENT_TO_OCEAN(...)
   ENDIF
 ELSEIF (cfn_nmlin_init /= "/dev/null") THEN
   CALL InitSeafloorFromNamelistFile(cfn_nmlin_init)
 ELSE
   WRITE(jp_stderr,
&    '("Error: no valid initialisation file given:")')
   WRITE(jp_stderr,'(" - cfn_nmlin_init = """, A, """)')
&                              TRIM(cfn_nmlin_init)
   WRITE(jp_stderr,'(" - cfn_ncin_init = """, A, """)')
&                              TRIM(cfn_ncin_init)
   WRITE(jp_stderr,'("Aborting!")')
   CALL ABORT_MEDUSA()
 ENDIF
```

3.6 Time-stepping sequence

For each sediment time step running, say, from t_i to t_{i+1}, a series of operations must be carried out.

3.6.1 Reset MOD_HOST_O2S arrays

A the very beginning of a time step, the arrays in MOD_HOST_O2S that MEDUSA uses to fetch its boundary conditions should be cleared (re-set to zero), so that the host biogeochemical module, which will have to perform the time-step from t_i to t_{i+1} first, can use them to accumulate the data that it must provide to MEDUSA:

```
CALL CLEAR_O2S_DATASET()
```

Afterwards, set the time and time-step length for MEDUSA

```
atime  = ... ! t_i [yr]
datime = ... ! t_{i+1} - t_i [yr]
```

3.6.2 Time-stepping the host biogeochemical module

Next, the biogeochemical module of the host model should be integrated over the time interval t_i to t_{i+1}. While doing so, it should prepare the boundary condition data required by MEDUSA and store them in the arrays in MOD_HOST_O2S:

- Temperature, salinity and depth for each grid point can either be averaged over $[t_i, t_{i+1}]$, or the values at time t_{i+1} can be used; similarly for grid-element surface areas, if they are varying in time. When using the recommended MOD_HOST_O2S module-based approach, these variables should go into
  seafloor_temp(...)
  seafloor_sali(...)
  seafloor_dept(...) and
  seafloor_surf(...), respectively.

- Extra parameter vales that you prefer to control by the host model instead of having them recalculated in MEDUSA (e. g., saturation concentrations, ...) data should go into the respective seafloor_*xxxx*(...) arrays that have been added to the actually used MOD_HOST_O2S in comparison to the templates

- sea-floor concentrations of the solute components can again either be averaged over $[t_i, t_{i+1}]$, or their values at time t_{i+1} can be used; these data should go into the `seafloor_wconc_xxxx(...)` arrays.

  For consistency reasons, it is always recommended that the host model does all the speciation calculations required (e.g., for the carbonate system). This can, however, also be done in the `OCEAN_TO_SEDIMENT` subroutine.

- Sea-floor fluxes of solids should, for mass conservation reasons, always be averaged over $[t_i, t_{i+1}]$. With the `MOD_HOST_O2S` module-based approach, the averaged values of these these data should then go into the `seafloor_wfflx_xxxx(...)` arrays.

The host biogeochemical module should only carry out the averaging and leave the data on its own native grid. It should furthermore not apply any unit conversions. These will be made by the `OCEAN_TO_SEDIMENT` subroutine from `MOD_HOST_O2S` that will be called next. This way, each model only needs to know about its own unit's system.

3.6.3 Time-stepping MEDUSA

Once the host biogeochemical module has completed its time step from t_i to t_{i+1}, we need to transfer the new contents of the arrays in `MOD_HOST_O2S` into `MOD_SEAFLOOR_CENTRAL`, where MEDUSA expects to find them:

```
CALL OCEAN_TO_SEDIMENT(...)
```

`OCEAN_TO_SEDIMENT` is in charge of any required remapping, re-sampling, unit conversion or speciation calculation that the host model has not done, and registers the boundary conditions into `MOD_SEAFLOOR_CENTRAL`.

Now MEDUSA's `SOLVSED_ONESTEP` can be found to perform the sediment time step

```
CALL SOLVSED_ONESTEP(atime, datime, n_columns,
&                    iflag, n_trouble)

atime = atime + datime
```

Once `SOLVSED_ONESTEP` completes, the MEDUSA core system must be regularized (cleaned up and purged, if necessary):

```
CALL REACLAY_X_CORELAY(atime)
```

REACLAY_X_CORELAY brings the core layer buffer into a controlled state: it checks the transition layer for overflow and creates new historical layers if required, or for low-stand and returns the contents of previously buried sediment layers into the transition layer. If the internal stack of layers has reached a threshold level, the oldest layers are written to the SEDCORE file (referred to above by cfn_ncout_sedcore), or deleted if no such file is used. If the stack runs empty, the most recent layers written to the file are transferred back onto the internal stack. REACLAY_X_CORELAY furthermore keeps track of eroded layers (mixed back into the transition layer as result of sufficiently strong chemical erosion).

If required, the sediment state can now also be written to file:

```
CALL WRITERES_NCFILES_MEDUSA(atime)
```

Finally, the next sediment time step has to be prepared. The flux values just calculated by MEDUSA are transferred from MOD_SEAFLOOR_CENTRAL to MOD_HOST_S2O, where the host biogeochemical module can retrieve them:

```
CALL SEDIMENT_TO_OCEAN(...)
```

SEDIMENT_TO_OCEAN should carry out any aggregation or unit change, so that the host biogeochemical module can retrieve the sediment-to-ocean fluxes on its own native grid and expressed in its own native units.

3.7 Terminating

3.7.1 Closing the NETCDF result files

If MOD_FILES_MEDUSA has been adopted, this step is performed by

```
CALL CLOSE_NCFILES_MEDUSA()
```

3.7.2 Finalizing the core system

The sediment core layers have to be finalized: if a SEDCORE file is open, the information about the core layers and eroded layers still remaining in the internal buffer needs to be written to the SEDCORE file and the overhead information in the file has to be completed; if no SEDCORE file is used, the stacks are cleared. The general purpose command to perform these actions is

```
CALL SEDFIL_FINALIZE()
```

3.7.3 Finalizing the execution environment (MPI only)

Similarly to the initialization of the execution environment, nothing has to be done if MEDUSA is not running under MPI.

Under MPI, a graceful termination of the main program requires a call to `MEDEXE_MPI_FINALIZE` at the end of the main program. If the initial `MEDEXE_MPI_INIT` call had to initialize MPI, `MEDEXE_MPI_FINALIZE` will now also finalize the MPI environment (i.e., call `MPI_FINALIZE`), then erases the MPI related information stored in `MOD_EXECONTROL_MEDUSA` and resets the module; else `MEDEXE_MPI_FINALIZE` simply erases the MPI related information stored in `MOD_EXECONTROL_MEDUSA` and resets the module.

MPI

4 Special Themes

4.1 Restart simulation experiments

If the code sequence and module organisation is adopted, it is straighforward to run restart simulation experiments. A simulation experiment (say '01') that is to be restarted must produce the following files (as listed by their keywords in the `&nml_cfg` namelist in `medusa_files.cfg`):

```
cfn_ncout_reaclay  = 'medusa_reaclay_01.nc'
cfn_ncout_flx      = 'medusa_flx_01.nc'
cfn_ncout_sedcore  = 'medusa_sedcore_01.nc'
```

The continuation experiment (say '02') which is meant to seamlessly extend experiment ' '01' then requires the following input files to be included in the `&nml_cfg` namelist in `medusa_files.cfg`:

```
cfn_ncin_init      = 'medusa_reaclay_01.nc'
cfn_ncin_flx       = 'medusa_flx_01.nc'
cfn_ncin_sedcore   = 'medusa_sedcore_01.nc'
```

Please notice that

- for continuation (restart) simulation experiments, the initial state must be read in from the NETCDF REACLAY file produced by the preceeding experiment (i.e., the file given by the `cfn_ncout_reaclay` in that simulation) so that the initial state of the continuation experiment corresponds to the final state of its preceeding experiment;

  !

- the input FLX file (specified by `cfn_ncin_flx`) is only required with asynchroneous coupling (the most common case).

The names of the output files to be produced by experiment '02' must be different from their analogues produced by experiment '01' (otherwise, these latter will be overwritten), except for `cfn_ncout_sedcore`:

- if `cfn_ncout_sedcore` and `cfn_ncin_sedcore` point to the same file, the core layer results of experiment '02' are appended to that file, updating the results from experiment '01' if required;
- if `cfn_ncout_sedcore` and `cfn_ncin_sedcore` point to different files, the data for the topmost layers from the file that `cfn_ncin_sedcore` points to are first copied into the file specified by `cfn_ncout_sedcore` and the results from experiment '02' then appended to that file. The `cfn_ncin_sedcore` file will not be modified but any existing copy of the `cfn_ncout_sedcore` file will first be erased. The two files will have overlapping contents.