Multi-Model-Driver (MMD) User Manual

Version 2.0

Astrid Kerkweg[1,2], Christiane Hofmann[1,2], Gregor Pante [1,3] and Patrick Jöckel[4]

[1] Institute for Atmospheric Physics
University of Mainz
55099 Mainz, Germany

[2] Meteorological Institute
University of Bonn
53121 Bonn, Germany
kerkweg@uni-bonn.de

[3] Institute of Meteorology and Climate Research
Department Troposphere Research (IMK-TRO)
Karlsruhe Institute of Technology (KIT)
76131 Karlsruhe, Germany
gregor.pante@kit.edu

[4] Deutsches Zentrum für Luft-und Raumfahrt (DLR),
Institut für Physik der Atmosphäre,
Oberpfaffenhofen, D-82234 Weßling, Germany
patrick.joeckel@dlr.de

This manual is available as electronic supplement of our article "Expanding the Multi-Model-Driver (MMD v2.0) for 2-way data exchange including data interpolation via GRID (v1.1)" in Geosci. Model Dev. Discuss. (2016), available at: http://www.geosci-model-dev.net

Date: January 20, 2017

Contents

1 Introduction

This manual is one part of a detailed description of the on-line coupling via the Multi-Model-Driver (MMD v2.0). MMD couples models following a client-server approach. It consists of two parts:

- The MMD library managing the data exchange between the different executables/models,
- the MESSy submodel MMD2WAY consisting of two sub-submodels
  - MMD2WAY_PARENT, providing the coarse grid data required by the client/child model and requesting the data coupled back to the parent model, and
  - the client MESSy sub-submodel MMD2WAY_CHILD, requesting the input data from the server / parent, subsequently interpolating these data for use in the model and providing the regridded data to the parent model.

The MMD library is described in the MMD library manual, which is part of the same electronic supplement as this manual. This manual, in contrast, is dedicated to the MMD MESSy submodel MMD2WAY. In the current implementation, the ECHAM5/MESSy general circulation model is supported as server/parent model, and the limited-area model COSMO/MESSy as server/parent and/or client/child model. Fig. 1 illustrates such a coupling setup for the MECO(n) system. The coupling layout, (i.e., which and how many model instances are operated concurrently and which instance operates on how many (and which) processing entities (PEs)) is determined within the MMD library. The child model sub-submodel MMD2WAY_CHILD organises the data transfer from the parent to the child model[1].

Within the MMD2WAY_CHILD namelist file the coupling frequency, i.e., how often data is exchanged between the parent and the child model, and the *exchange fields*[2] are specified.

After the data exchange from the parent to the child model, MMD2WAY_CHILD interpolates the coarse grid data to the COSMO model grid using its submodel INT2COSMO. INT2COSMO is based on INT2LM as provided by the German Weather Service (DWD) for the interpolation of the initial and boundary data for the COSMO model. INT2COSMO and INT2LM contain basically the same code, but for distinction we hereafter refer to the MMD2WAY_CHILD submodel as INT2COSMO, i.e., the on-line preprocessing tool, and to INT2LM as the standard off-line (or stand-alone) application.

For the backward coupling, the child model maps the data from the child model grid to the parent model grid using the MESSy infrastructure submodel GRID. These interpolated data are afterwards sent to the parent model.

The basic tasks of the parent model are to impose its time and date setting (except the time step length) on the child and to provide the requested data to the client. Additionally, if 2-way coupling is required, it requests data from the child model and applies these to the parent model variables.

In the first part of the manual those files, which have to be modified by a user of the system are explained:

[1]The terms "model", "model instance" and "instance" do not mean exactly the same thing. A "model" is the model itself (e.g., for the MECO(n) system these are COSMO/MESSy or EMAC). In an MMD coupled system different instances of these models are run concurrently. Thus a "model instance" or "instance" is one realisation of the model configuration within the coupled setup. However, as it is intuitively clear whether a "model" or an "instance" is addressed, we will use these terms synonymously.

[2]The Appendix contains a glossary explaining some terms repeatedly used here. The terms from the glossary are written in italics throughout the article. Especially, Fig. 17 in the glossary illustrates the meaning of the different *coupling fields*.

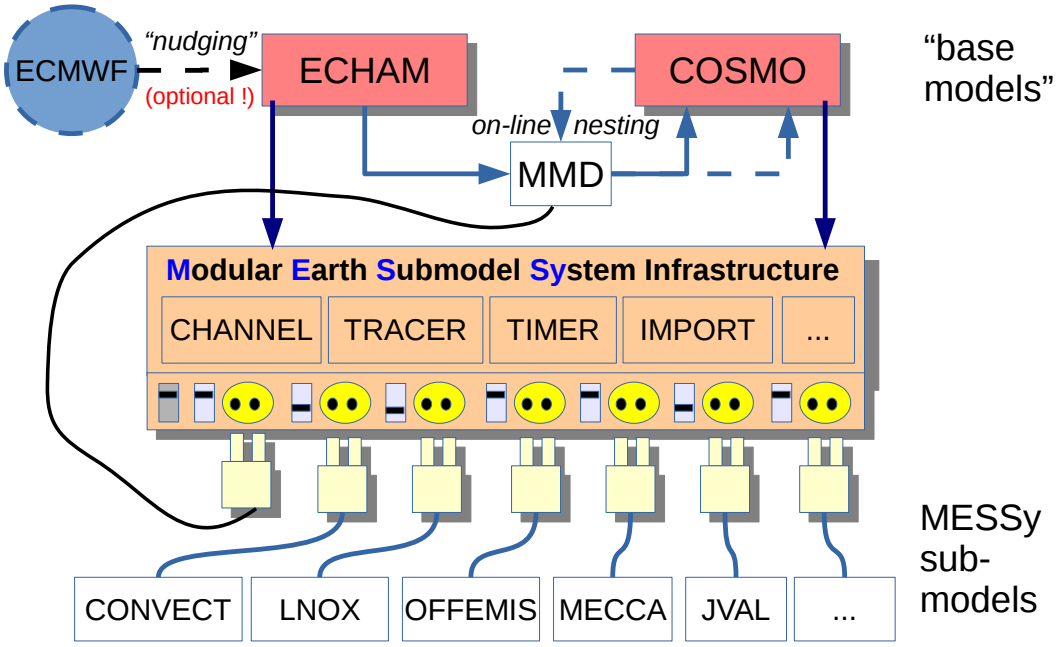

Figure 1: Illustration of a MECO(n) coupled setup. Each of the basemodels, ECHAM and COSMO, are coupled internally to MESSy. Externally, different instances of these model are nested into each other via MMD.

- Sect. 2 illustrates the run-script settings,
- Sect. 3 describes the details about the individual namelist entries in the MMD2WAY namelist file, which contains namelists for the child coupling (Sect. 3.3) and for the parent coupling (Sect. 3.4), and
- for an overview, Sect. 4 roughly summarises the coupling work flow.

The second part of the manual is for those users, which want to expand the coupling or simply want to know more details about the coupling implementation: in Sect. 5 the 1-way coupling procedure is explained, while Sect. 6 describes the additions which have been implemented for the backward (or parent-to-child) coupling. The superposition of the parallel decomposed COSMO model and INT2COSMO grids poses a specific challenge, which solution is discussed in Sect. 5.1.4.

Last but not least, the code changes of the individual model codes, which are required in order to enable the on-line coupling, are listed for the different code sources, i.e., INT2COSMO, the COSMO model and the ECHAM5 model, in the Sections 7, 8 and 9, respectively.

2 The run-scrip `xmessy_mmd`

The run-script consists of three major sections:

1. The first section contains batch job scheduler (queueing system) settings. All schedulers MESSy was, so far, used with, are listed in this run-script. New queueing systems can be added easily. The user has to activate the appropriate setup for the computing system he/she is using.
2. The second section is the one which needs specifications according to the intended model simulation. Thus this section is subject to changes by every user for a specific simulation.
3. The last section contains all MESSy and machine specific settings. These should normally not be changed by a model user. Only, if a new system is added, changes are also required in this part of the run-script.

Here, we focus on the second block of the run-script, i.e., that part that needs to be adapted by the user for a specific model setup. Its start is indicated in the run-script by the comment:

```
###############################################################################
### USER DEFINED GLOBAL SETTINGS
###############################################################################
```

The settings are described one after the other as they are aligned in the script:

- `EXP_NAME`: This is the name of the experiment. This variable is copied to the CHANNEL[3] namelist. All CHANNEL output files will start with this experiment name. Its maximal length is 14 characters.
- `WORKDIR`: This is the directory in which the simulation is actually performed. In this directory subdirectories are created by the run-script and all data written during the simulation are placed into these (sub-)directories. For each model instance (see below) a subdirectory named by the instance number is created. Note: for most scheduling systems the log-file will be placed in the directory from which the run-script is submitted.
- `START_YEAR`, `START_MONTH`, `START_DAY`, `START_HOUR`, `START_MINUTE`, `START_SECOND`: These are the start date/time components. They are copied to the TIMER[4] namelist defining the start date/time of the simulation. Additionally, they are copied to all namelists which require start time dependent entries. For instance, for an emission file containing monthly averaged emission fluxes the knowledge of the month in which the simulation starts is required. Nudging is another important example depending on the start date components.
- `STOP_YEAR`, `STOP_MONTH`, `STOP_DAY`, `STOP_HOUR`, `STOP_MINUTE`, `STOP_SECOND`: These date components are copied to the TIMER and nudging namelists to determine the end of a simulation.
- `RESTART_INTERVAL=1`, `RESTART_UNIT=months`: These entries determine the frequency with which restart files are written. `RESTART_UNIT` provides the unit of the interval, while `RESTART_INTERVAL` defines the number of steps in unit `RESTART_UNIT`. These parameters are copied to the TIMER namelist. —

[3]The CHANNEL submodel is described in detail in the electronic supplement of Jöckel et al. (2010).
[4]The TIMER submodel is described in detail in the electronic supplement of Jöckel et al. (2010).

- `NML_SETUP`: This variable determines which namelist setup is used. In the subdirectory `messy/nml/` within the MESSy distribution different namelist setups are available. `NML_SETUP` selects the name of the subdirectory, which should be used. If an ECHAM5/MESSy-only simulation is performed, the subdirectory contains only the namelists required for an ECHAM5/MESSy simulation. For a COSMO/MESSy only simulation, the namelist subdirectory contains only the namelists required for one COSMO/MESSy model instance. For the coupled simulations the respective directory contains as many subdirectories as coupled instances exist. The numbers in the coupling layout (see below) are the same as the numbers of the namelist subdirectories. The namelists for the coupled setups are all placed in a subdirectory called `MMD`.

- `OFT` (Output File Type): At the time being it can be chosen between netCDF[5] and parallel-netCDF[6], if the latter is available. This flag is copied to the CHANNEL namelist.

- `QWCH`: This should be set to the available wall-clock hours defined by the schedule and is copied to the QTIMER[7] namelist.

- `INSTANCE`: This gives the number and type of model instances running simultaneously in the MPI environment. The names "ECHAM5" and "COSMO" indicate whether in this instance an ECHAM5/MESSy model or a COSMO/MESSy model is executed.

```
### ========================================================================
### SELECT MODEL INSTANCES:
###  - ECHAM5, MPIOM, CESM1 (always first, if used)
###  - COSMO
###  - other = MBM
### ========================================================================
INSTANCE[1]=ECHAM5
INSTANCE[2]=COSMO
INSTANCE[3]=COSMO
```

  If ECHAM5/MESSy is the coarsest parent model (*master parent* or *patriarch*), this needs to be the **first** instance. If ECHAM5/MESSy or COSMO/MESSy are run alone, only one instance is set. The run-script can also be used to run other MESSy models, e.g., the generic MESSy basemodel BLANK (`INSTANCE[1]=blank`), CAABA (`INSTANCE[1]=caaba`, Sander et al. (2011)), MPIOM (`INSTANCE[1]=mpiom`, Pozzer et al. (2011)) or CESM1 (`INSTANCE[1]=CESM1`, Baumgaertner et al. (2016)) can be run as autonomous model with the same run-script.

- `MMDPARENTID`: For each model instance the server of the model needs to be determined. The server of a model is defined by its instance number.

```
### ========================================================================
### SET MMD PARENT IDs (-1: PATRIARCH)
### ========================================================================
MMDPARENTID[1]=-1
MMDPARENTID[2]=1
MMDPARENTID[3]=2
```

[5] *http://www.unidata.ucar.edu/software/netcdf/*
[6] *http://www.mcs.anl.gov/parallel-netcdf*
[7] The QTIMER submodel is described in Jöckel et al. (2010).

The *patriarch* is indicated by "-1" because the *patriarch* has no server itself. For the above example, the server of instance number 2 is the first instance, i.e., the ECHAM5/MESSy model. The second instance itself is server to the third instance. `MMDPARENTID[3]=1` would mean that the third model also gets its data directly from ECHAM5/MESSy.

- `NPX`, `NPY` and `NVL`: `NPX` and `NPY` determine the parallel domain decomposition of the model instances in x and y direction, respectively. For the COSMO model these are copied to `nprocx` and `nprocy` in the COSMO namelist `&RUNCTL` in the `INPUT_ORG.nml` namelist file. For ECHAM5/MESSy these entries are copied to `NPROCA` and `NPROCB` in the `&RUNCTL` namelist of the `ECHAM5.nml` namelist file. So far, `NVL` was only of importance for ECHAM5/MESSy, as the vector length `NPROMA` is set to `NVL`. For the COSMO model versions containing the unified COSMO-ICON model physics `NVL` also defines the block length used in the COSMO physics.
- ECHAM5 specific settings:
  - `ECHAM5_HRES`,`ECHAM5_VRES`: Spectral and vertical resolution of ECHAM5. `ECHAM5_HRES` is for example one of `(T106, T85, T63, T42, T31, T21, T10)`, whereas `ECHAM5_VRES` is for example, one of `(L19, L31ECMWF, L41DLR, L39MA, L90MA)`. Note that ECHAM5 always requires initial data in the chosen resolution.
  - `MPIOM_HRES`, `MPIOM_VRES`: horizontal and vertical resolution of MPIOM, when it is chosen as a submodel. `MPIOM_HRES` is for example one out of `(GR60, GR30, GR15, TP04, TP40)` and `MPIOM_VRES` one out of `(L3, L20, L40)`.
  - `ECHAM5_NUDGING`: This LOGICAL is set to T, if nudging of the ECHAM5 model is requested. The nudging coefficients in the ECHAM5 namelist file (namelist `&NDGCTL`) must be set accordingly.
  - `ECHAM5_LAMIP`: Switch for using sea-surface temperature (sst) and sea-ice forcing via AMIP-like data for ECHAM5.
  - `NML_ECHAM`: name of the ECHAM5 namelist file. As the ECHAM5 namelists include some resolution dependent entries, it is convenient to work with resolution dependent ECHAM5 namelist files.
- user-defined specific namelist files, e.g., depending on the resolution, the start date, etc.
- COSMO specific settings: `COSMO_SUBDIR`, `COSMO_EXTNAME` and `COSMO_EXTGRID`: These entries are used to determine the INT2COSMO namelist entries required to access the external data file. `COSMO_EXTNAME` fills the namelist entry `ylmext_lfn` and `COSMO_EXTGRID` contains the horizontal grid sizes of the external data file, which naturally depend on the external data file and are required as individual entries in the INT2COSMO namelist `&DATA`. `ylmext_cat` is filled by `COSMO_EXTDIR`, which is composed of a general data input path (`INPUTDIR_COSMO_EXT`) and a subdirectory (`COSMO_SUBDIR`) in this input path. `COSMO_SUBDIR` has to be defined individually for each instance, while a default value (as part of the standard MESSy input directory) exists for `INPUTDIR_COSMO_EXT`. However, `INPUTDIR_COSMO_EXT` can also be user-defined for each individual instance.

```
### -> COSMO_EXTDIR[.] = ${INPUTDIR_COSMO_EXT[.]}/$COSMO_SUBDIR[.]
COSMO_SUBDIR[1]=
COSMO_EXTNAME[1]=
COSMO_EXTGRID[1]=
```

```
COSMO_SUBDIR[2]=climatology
COSMO_EXTNAME[2]=europe.nc
COSMO_EXTGRID[2]="ie_ext=101, je_ext=107,"

COSMO_SUBDIR[3]=external
COSMO_EXTNAME[3]=lm_d1_g0.165_463x383
COSMO_EXTGRID[3]="ie_ext=463, je_ext=383,"
```

The number in brackets is the instance number. In the example with ECHAM5/MESSy as first instance, the block with instance number 1 must be empty.

The above listed variables need to be set. Here, additional variables are listed, which can be set, if the default settings should not be used:

- `BASEDIR`: This is the directory of the model distribution.
- `DATABASEDIR`: Base directory for model initial data.
- `INPUTDIR_ECHAM5_INI`: Directory containing the input data for the ECHAM5 model.
- `INPUTDIR_ECHAM5_SPEC`: Directory containing the `*_spec` and `*_surf` files for the ECHAM5 initialisation. These depend on the resolution and start date. With `INPUTDIR_ECHAM5_SPEC`, the user can put the initial files specific for the start date of his/her simulation into a private directory and use the default directory for the others.
- `INPUTDIR_NUDGE`: Directory of the nudging data files for ECHAM5.
- `FNAME_NUDGE`: Name template of the ECHAM5 nudging data files. This differs depending on the source of the data. (ERA40/ERA-Interim or analysis (ANALY)).
- `INPUTDIR_AMIP`: Directory of the sst and sea-ice data for ECHAM5.
- `INPUTDIR_MPIOM`: Directory containing MPIOM input data.
- `INPUTDIR_COSMO_EXT`: Directory containing COSMO input ("external") data.
- `INPUTDIR_COSMO_BND`: Directory containing COSMO initial and boundary data, if COSMO is run in stand-alone mode or as *patriarch*.
- `INPUTDIR_CESM1`: Directory containing CESM1 input data.
- `INPUTDIR_MESSY`: Directory containing input data for the MESSy submodels.
- `USE_PREREGRID_MESSY`: It is possible to provide the MESSy input data on the specific horizontal Gaussian grid, i.e., in pre-regridded form. This is used when `USE_PREREGRID_MESSY = T`. Note: this only works for ECHAM5/MESSy.
- SPECIAL MODES:
  - `SERIALMODE`: switch on, if basemodel was compiled without MPI[8] parallelisation.
  - `TESTMODE`: Test mode of the run-script, exits before starting the executable.

[8] message passing interface

- `MEASUREMODE`: Measure memory use. This is only available on specific machines.
- `PROFMODE`: A special mode, for performance monitoring, which is only available on specific machines: Possible settings are `TPROF`, `VAMPIR` or `SCALASCA`. Additionally, the parameter `PROFCMD` is required.

The user specified block ends with the marker:

```
###############################################################################
###############################################################################
### =========================================================================
###############################################################################
### DO NOT CHANGE ANYTHING BELOW THIS LINE !!!
###############################################################################
### =========================================================================
###############################################################################
###############################################################################
```

3 The MMD2WAY namelists

The namelist file of the submodel MMD2WAY consists of five different namelists:

- `&CTRL`: the overall control namelist read by the child model
- `&CPL_PARENT`: a namelist generally driving the coupling from the parent model side (read by the parent model).
- `&CPL_PAR_CHILD`: an arbitrary number of these namelists, specifying the transfer of child fields to the parent model individually for each coupled instance (read by the parent model).
- `&CPL_CHILD`: a namelist generally driving the coupling from the child model side (read by the child model).
- `&CPL_CHILD_ECHAM` or `&CPL_CHILD_COSMO`: namelist specifying which fields are required from the parent model for the child model (read by the child model). For simplicity reasons two different namelists are provided for the two possible parent models ECHAM5 or COSMO.

In the following the individual namelists are described. First the namelists read by the child models are explained.

3.1 &CTRL

The `&CTRL`-namelist (Fig. 2) consists of two blocks of namelist variables: one to trigger original INT2LM output and the other for an improved initialisation of the soil variables:

- two variables are required to trigger the original output: one LOGICAL switch and one *event*[9]:

[9]The generic submodel TIMER and the definition and functionality of *events* are described in the manual about TIMER within the electronic supplement of Jöckel et al. (2010).

```
! -*- f90 -*-
&CTRL
! WRITE ORIGINAL INT2COSMO  OUTPUT
l_I2Cori_output = .FALSE.
! do not use steps
WRITEI2C_IOEVENT  = 1,'hours','exact',0
! INITIALSE VARIABLES
!l_forcevars = .TRUE.,
!forcevars = "T_SO;W_SO;T_S;W_I;QV_S;W_SNOW;T_SNOW",
!forcefile = "/DATA/COSMO/soil_ini/lffd2001010100.nc"
/
```

Figure 2: Example `&CTRL`-namelist of the MMD2WAY namelist file (`mmd2way.nml`)

  - If the LOGICAL `l_I2Cori_output` is set `.TRUE.` (default: `.FALSE.`), INT2COSMO produces its original output, i.e., the initial and boundary files are written during the on-line coupled simulation. They can be used later on to perform off-line COSMO simulations. Note: this only works for original INT2LM output, implying that MESSy specific boundary data (e.g., for chemical species) are not written to these files.
  - The *event* (`WRITEI2C_IOEVENT`) determines the temporal interval in which files are written, if `l_I2Cori_output = .TRUE.`.

- the forcing of specific variables requires three namelist variables:
  - the LOGICAL `l_forcevars` to switch on this specific feature (default: `.FALSE.`).
  - the string variable `forcevars` containing the names of all variables to be re-initialised by this procedure.
  - the string variable `forcefile` providing the path and the name of the file containing the data.

  Note: the COSMO domain definition in `forcefile` has to be exactly the same as in the simulation performed.

3.2 &CPL_CHILD

The `&CPL_CHILD`-namelist (Fig. 3) is read by MMD2WAY_CHILD and determines the interval of coupling between this specific child and its parent model. The namelist contains two *events*:

- the first (`CPL_IOEVENT`) determines the interval of the coupling to the parent model, i.e., how often data are exchanged between parent and child model,
- the second *event* (`READEXT_IOEVENT`) determines the update interval of the external data required by `INT2COSMO`.

For the definition of the coupling *event* the user has to be aware of two limitations:

- To simplify the data exchange between the coupled models, the coupling interval is internally converted to seconds. As this conversion is not well defined for the units `'months'` and `'years'`, the coupling interval must be specified in `'steps'`, `'seconds'`, `'minutes'`, `'hours'` or `'days'`.
- As for all other *events*, the user has to define a multiple of the model time step length, otherwise the simulation is terminated. Especially, the coupling interval is a multiple of the time steps of the child and the parent model.

```
&CPL_CHILD
CPL_IOEVENT      = 10,'minutes','first',0
READEXT_IOEVENT  = 1,'years','none',0
/
```

Figure 3: Example `&CPL_CHILD`-namelist of the MMD2WAY namelist file (`mmd2way.nml`).

3.3 &CPL_CHILD_ECHAM and &CPL_CHILD_COSMO

The second part of the namelist file, which is child model relevant, contains a list of *exchange fields* required to fully initialise and drive the respective child COSMO/MESSy model. The *exchange fields* are unambiguously identified by their *channel* and *channel object* names. As these usually differ between ECHAM5/MESSy and COSMO/MESSy, the namelists depend on the parent model.

Figure 4 shows a typical `&CPL_CHILD_ECHAM` namelist for the coupling to ECHAM5/MESSy and Fig. 5 shows the namelist `&CPL_CHILD_COSMO` used for the coupling to a COSMO/MESSy model as parent. The structure of the two namelists is identical. Each *exchange field* is defined by one namelist entry of TYPE `FIELD`:

```
FIELD(.) = 'PARENT_CHANNEL', 'PARENT_OBJECT', 'CHILD_CHANNEL', 'CHILD_OBJECT'
         , 'INTERPOL_METHOD', L_INITIAL, L_BOUND, L_INPUT, 'CHILD_REPR'
```

`FIELD` is a variable of TYPE `T_C_EXCH_IO`:

```
  TYPE CHAOBJ_NAMES
     CHARACTER(LEN=STRLEN_CHANNEL)  :: CHA = '' ! CHANNEL NAME
     CHARACTER(LEN=STRLEN_OBJECT)   :: OBJ = '' ! OBJECT  NAME
  END TYPE CHAOBJ_NAMES

  TYPE T_C_EXCH_IO
     TYPE(CHAOBJ_NAMES) :: PARENT
     TYPE(CHAOBJ_NAMES) :: CHILD
     CHARACTER(LEN=4)   :: C_INTERPOL = ''      ! INTERPOLATION METHOD
     ! Specify target field
     LOGICAL            :: L_INITIAL  = .FALSE. ! INITIAL FIELD
     LOGICAL            :: L_BOUND    = .FALSE. ! BOUNDARY FIELD
     LOGICAL            :: L_INPUT    = .FALSE. ! INPUT FIELD
     CHARACTER(LEN=STRLEN_MEDIUM) :: C_REPR ='' ! REPRESENTATION STRING
  END TYPE T_C_EXCH_IO

  ! MAXIMAL NUMBER OF EXCHANGE FIELDS
  INTEGER, PARAMETER                             :: NMAX_EXCH = 1000
  TYPE(T_C_EXCH_IO), DIMENSION(NMAX_EXCH), SAVE :: FIELD
```

- The first two structure components of TYPE CHARACTER specify the *channel* and *channel object* name of the *exchange field* on the parent side. For instance, the surface pressure field in ECHAM5/MESSy is defined in the *channel* 'g3b' with the *channel object* name 'aps' (see `FIELD(1)` in Fig. 4).
- The third and fourth structure components of TYPE CHARACTER name the *channel* and *channel object* of the *exchange field* in the client model. For `FIELD(1)` in Fig. 4 this is the *channel* 'COSMO_ORI' and the *channel object* 'PS'.

```
&CPL_CHILD_ECHAM
! ############################################################################
!
! ############################################################################
! ### MANDATORY FIELDS
! ############################################################################
!****************************************************************************
FIELD(1)   = 'g3b','aps', 'COSMO_ORI', 'PS', '', F, F, F , ''
!****************************************************************************
FIELD(2)   = 'ec2cosmo','T_S', 'COSMO_ORI','T_S', '', T, T, F, ''
!****************************************************************************
FIELD(3)   = 'g3b','slf', 'COSMO_ORI','FR_LAND', '', T, F, F, ''
!****************************************************************************
FIELD(4)   = 'g1a','tm1', 'COSMO_ORI','T', '', T, T, F, ''
!****************************************************************************
FIELD(5)   = 'g1a','qm1', 'COSMO_ORI','QV', '', T, T, F, ''
!****************************************************************************
FIELD(6)   = 'g1a','xlm1', 'COSMO_ORI','QC', '', T, T, F, ''
!****************************************************************************
FIELD(7)   = 'g1a','xim1', 'COSMO_ORI','QI', '', T, T, F, ''
!****************************************************************************
FIELD(8)   = 'g2a','um1', 'COSMO_ORI','U', '', T, T, F, ''
!****************************************************************************
FIELD(9)   = 'g2a','vm1', 'COSMO_ORI','V', '', T, T, F, ''
!****************************************************************************
FIELD(10)  = 'g3b','geosp', '#XXX','FIS', '', F, F, F, ''
!****************************************************************************
FIELD(11)  = 'g3b','wl', 'COSMO_ORI','W_I', '', T, F, F, ''
!****************************************************************************
FIELD(12)  = 'g3b','sni', 'COSMO_ORI','W_SNOW', '', T, T, F, ''
!****************************************************************************
FIELD(13)  = 'g3b','tsi', 'COSMO_ORI','T_SNOW', '', T, T, F, ''
!****************************************************************************
FIELD(14)  = 'ec2cosmo','W_SO_REL', 'COSMO_ORI','W_SO', '', T, F, F, ''
! ############################################################################
! ### OPTIONAL FIELDS
! ############################################################################
FIELD(20)  = 'Test','Test_Ar', 'mmd2way_child','Test_Ar', '', F, F, F, ''
!****************************************************************************
FIELD(21)  = 'tracer_gp_m1','O3', 'tracer_gp','O3', 'QFTV', T, T, F, ''
!****************************************************************************
FIELD(22)  = 'ptrac_gp','wetradius', 'ptrac_gp','wetradius', 'QTFV', T, F, F, ''
!****************************************************************************
FIELD(23)  = 'jval_gp','J_O1D', 'mmd2way_child','J_O1D', 'QFTV', F, F, T, 'GP_3D_MID'
!****************************************************************************
FIELD(24)  = 'import_grid','RGT0012_CO','mmd2way_child','RGT0012_CO','M',F,F,T,'#UNKNOWN'
! ############################################################################
/
```

Figure 4: Example `&CPL_CHILD_ECHAM` namelist of MMD2WAY namelist file (`mmd2way.nml`).

For the child *channel object* names wildcards are allowed. A `'*'` replaces an arbitrary number of characters or digits, whereas `'?'` replaces exactly one character or digit. Based on wildcards, it is possible to address a number of *channel objects* of one *channel* with one namelist entry. The only restriction for wildcard usage is that the name of the *channel object* on the parent side must be identical to that on the child side[10], because the names for the parent *channel object*s are over-

[10]This is usually not the case for the basemodels ECHAM5 and COSMO, but for the MESSy submodels.

```
&CPL_CHILD_COSMO
! ##############################################################################
!
! ##############################################################################
! ### MANDATORY FIELDS
! ##############################################################################
FIELD(1)  = 'COSMO','ps', 'COSMO_ORI','PS', '', F, F, F, ''
!******************************************************************************
FIELD(2)  = 'COSMO','t_s', 'COSMO_ORI','T_S', '', T, T, F, ''
!******************************************************************************
FIELD(3)  = 'COSMO_ORI','FR_LAND', 'COSMO_ORI','FR_LAND', '', T, F, F, ''
!******************************************************************************
FIELD(4)  = 'COSMO','tm1','COSMO_ORI','T', '', T, T, F, ''
!******************************************************************************
FIELD(5)  = 'COSMO','qv','COSMO_ORI','QV', '', T, T, F, ''
!******************************************************************************
FIELD(6)  = 'COSMO','qc','COSMO_ORI','QC',  '', T, T, F, ''
!******************************************************************************
FIELD(7)  = 'COSMO','qi','COSMO_ORI','QI',  '', T, T, F, ''
!******************************************************************************
FIELD(8)  = 'COSMO','um1','COSMO_ORI','U',  '', T, T, F, ''
!******************************************************************************
FIELD(9)  =  'COSMO','vm1', 'COSMO_ORI','V','', T, T, F, ''
!******************************************************************************
FIELD(10) = 'COSMO','t_so', 'COSMO_ORI','T_SO', '', T, F, F, ''
!******************************************************************************
FIELD(11) = 'COSMO','w_so','COSMO_ORI','W_SO',  '', T, F, F, ''
!******************************************************************************
FIELD(12) = 'COSMO','t_snow', 'COSMO_ORI','T_SNOW', '', T, T, F, ''
!******************************************************************************
FIELD(13) = 'COSMO','w_snow', 'COSMO_ORI','W_SNOW', '', T, T, F, ''
!******************************************************************************
FIELD(14) = 'COSMO','w_i', 'COSMO_ORI','W_I', '', T, F, F, ''
!******************************************************************************
FIELD(15) = 'COSMO','qv_s', 'COSMO_ORI','QV_S', '', T, T, F, ''
!******************************************************************************
FIELD(16) = 'COSMO_ORI','FRESHSNW', 'COSMO_ORI','FRESHSNW', '', T, F, F, ''
!******************************************************************************
FIELD(17) = 'COSMO_ORI','HSURF', 'COSMO_ORI','HSURF', '', T, F, F, ''
!******************************************************************************
FIELD(18) = 'COSMO','ppm1', 'COSMO_ORI','PP', '', T, T, F, ''
!******************************************************************************
FIELD(19) = 'COSMO_ORI','SOILTYP', 'COSMO_ORI','SOILTYP', '', T, F, F, ''
!******************************************************************************
! ##############################################################################
! ### OPTIONAL FIELDS
! ##############################################################################
FIELD(20) = 'Test','Test_Ar', 'mmd2way_child','Test_Ar', '', F, F, F, ''
!******************************************************************************
!******************************************************************************
FIELD(21) = 'tracer_gp_m1','all', 'tracer_gp','*', 'QFTV', T, T, F, ''
!******************************************************************************
! ##############################################################################
/
```

Figure 5: Example `&CPL_CHILD_COSMO`-namelist of MMD2WAY namelist file (`mmd2way.nml`).

written by the child *channel object* names, if wildcards are used. For instance, an entire tracer set can be coupled by setting `'CHILD_CHANNEL', 'CHILD_OBJECT'` to `'tracer_gp', '*'` or all

photolysis rates are coupled by `'jval_gp', 'J_*'`. Due to the initialisation of prognostic variables at the beginning of each time step in COSMO/MESSy in the subroutine `initialize_loop` in `lmorg.f90` the parent *channel* for the coupling of the tracers needs to be `'tracer_gp'`. In case of ECHAM5 the fields in `'tracer_gp'` and `'tracer_gp_m1'` are identical at the beginning of the time loop.

- The fifth structure component is a CHARACTER of length 4. It determines the interpolation method. This is only required for the *additional fields*, as for the *INT2COSMO inherent fields* the interpolation method is determined inside of INT2COSMO. Possible interpolation methods are: 'Q' for quadratic; 'L' for linear and 'M' for match point interpolation. Additionally 'C' for conservative remapping via the MESSy submodel GRID has been added to INT2COSMO. Thus the first character must be set to one of 'Q', 'L', 'M' or 'C'. The second and the third character demand monotonicity and positive definiteness, respectively, if set to T. The default value, however, is F. If the fourth character is 'V' or 'W', the field will additionally be interpolated in the vertical direction via the INT2COSMO inherent spline method ('V') or via NREGRID ('W'). However, this is only possible for 3D- or 4D-fields of which the number of vertical levels equals the number of vertical levels in the model. For instance, the fifth string of FIELD(21) in `&CPL_CHILD_ECHAM` determines that the ozone tracer is interpolated horizontally by quadratic interpolation and in addition vertically using the spline method. No care is taken to ensure monotonicity, but positive definiteness is requested.

- The next three logicals in the FIELD(:) entry indicate the data destination (*initial*, *boundary* or *input*) of the interpolated field. *Mandatory fields* can be *initial* and *boundary fields*. For the *mandatory fields* the entries for the data destination types in the namelist could be omitted, as they are set according to the COSMO variables `yvarini` and `yvarbd`. These variables list the *initial* and *boundary fields* required for the chosen COSMO setup. If *initial* or *boundary fields* are required according to `yvarini` or `yvarbd` and the data destination flags are not set `.TRUE.` in the namelist, the namelist settings are ignored. If a field destination is requested (in addition to `yvarini` or `yvarbd`) as *initial* or *boundary field*, however, this request is not overwritten. In other words, if the field is requested in the namelist or the COSMO model, it will be processed.

  For the *optional fields* the choice of *initial* and/or *boundary* and of *input* destination is exclusive, as *input* already implies *initial* and the provision of *boundary* data is meaningless, since the field is overwritten each coupling time step. For instance, for the prognostic variables water vapour and cloud water (`FIELD(5)` and `FIELD(6)` in Fig. 4) the calculation of the *initial* and *boundary fields* is requested, whereas for the land fraction (`FIELD(3)` in Fig. 5) only the *initial field* is calculated. As tracers are prognostic variables, *initial* and *boundary fields* are requested for ozone (`FIELD(21)`). In contrast, the fields `FIELD(23)` and `FIELD(24)` are *input fields*.

- The last component of the variable `FIELD` contains the *representation*[11] of the childs *channel object*. It is only required for *additional fields*, for which `L_INPUT` is `.TRUE.`. In this case the memory for a field is neither defined by a MESSy submodel nor by the basemodel itself and consequently, the submodel MMD2WAY_CHILD has to define the respective *channel object* itself, which is indicated by giving `'mmd2way_child'` as child *channel* name in the third `FIELD` entry in the `&CPL_CHILD_ECHAM` namelist. For these fields the *representation* must be known as MMD2WAY_CHILD needs to allocate the memory for the respective field itself.

  For instance, in `FIELD(23)` in the `&CPL_CHILD_ECHAM` namelist, the photolysis rate of O^1D from the ECHAM5/MESSy submodel JVAL (*channel* name `'jval_gp'`, *channel object* name `'J_O1D'`)

[11] For a description of *representations* see the CHANNEL manual, which is part of the electronic supplement of Jöckel et al. (2010).

is defined as *input field* of the regional model. If JVAL is not switched on in COSMO/MESSy, MMD2WAY_CHILD needs to define the *channel object* itself. The photolysis rates are defined at the center of the grid boxes. Thus the *representation* of a photolysis rate is a priori known and the *representation* name for the child model can be specified (here, `'GP_3D_MID'`).

In cases where the *representation* is not a priori known, it is deduced from the *representation* of the parent *channel object*. This heuristic procedure, triggered by the entry `'#UNKNOWN'` (see `FIELD(24)` in Fig. 4), is described in detail in Sect. 5.1.1.2.

In addition to the coupling of standard 2D and 3D data fields, the coupling of 4D data fields is implemented. They are treated exactly in the same way. However, due to differences in the implementation of tracers (Jöckel et al., 2008) and the implementation of prognostic variables in the COSMO model, it is not possible to couple the 4D tracer field directly. Nevertheless, each individual tracer can be coupled, as the individual tracers are accessible as 3D *channel objects* (e.g., `FIELD(21)` in Fig. 4). To simplify the handling of large tracer sets, wildcards can be used for the child *channel object* names in the namelist: `'*'` replaces an arbitrary number of characters, `'?'` replaces exactly one character. For instance, `FIELD(25)` would request all tracers available in the *channel* `'tracer_gp_m1'`. Of course, wildcards in the *channel object* names can be used for other *channels* as well.

3.4 &CPL_PAR_CHILD

```
&CPL_PAR_CHILD
INSTANCE='002'
lgrh     = .FALSE.,
ldiagonly = .TRUE.,
i_rmy_px = 21,
rdefpc = 30000.,
itype_fw = 2,
PFIELD(1)  = 'g1a','qm1', 'scnbuf','qte', 'tracer_gp','QV', 'GP_3D_MID',1,1,1.
PFIELD(2)  = 'g1a','xlm1','scnbuf','xlte','tracer_gp','QC', 'GP_3D_MID',1,1, 0.5
/
```

Figure 6: Example `&CPL_PAR_CHILD`-namelist of MMD2WAY namelist file (`mmd2way.nml`). This namelist is specific for the case of ECHAM5/MESSy as parent model.

```
&CPL_PAR_CHILD
INSTANCE='003'
RCF = 100,
RCF_IN = 100,
i_rmy_px = 25

PFIELD(1)  = 'tracer_gp','QV', '','', 'tracer_gp','QV', 'GP_3D_MID',1,1, 1.
PFIELD(2)  = 'tracer_gp','QC', '','', 'tracer_gp','QC', 'GP_3D_MID',1,1, 0.5
PFIELD(5)  = 'COSMO_ORI','PS',' ',' ','COSMO_ORI','PS','GP_2D_HORIZONTAL',1,1,1.
/
```

Figure 7: Example `&CPL_PAR_CHILD`-namelist of MMD2WAY namelist file (`mmd2way.nml`). This namelist is specific for the case of COSMO/MESSy as parent model.

Figures 6 and 7 show two typical examples for parent coupling namelists. Figure 6 is specific for ECHAM5/MESSy as parent model and Fig. 7 for COSMO/MESSy as parent model.

The following parameters can be part of the `&CPL_PAR_CHILD`:

- `INSTANCE`: a mmd2way.nml namelist file may contain an arbitrary number of `&CPL_PAR_CHILD` namelists. The CHARACTER string of length 3, '`INSTANCE`', is required to attribute each of these namelist blocks to one specific coupling instance. The number provided by the namelist parameter `INSTANCE` refers to the instance number as set in the MMD coupling namelist file `MMD_layout.nml` written and defined by the run-script `xmessy_mmd`.
- `itype_fw`: weight function (see page 72). Default: `itype_fw = 2`
- `icosexp`: factor as required for `itype_fw = 1`. Default value is `icosexp = 14`.
- `damprel`: factor as required for `itype_fw = 2`. Default value is `damprel = 0.02`.
- `i_rmy_px`: number of grid boxes which are not coupled back in addition to the damping zone of the child COSMO model domain. Default is `i_rmy_px = 0`.
- `RCF`: scaling factor to avoid grid rounding errors as much as possible (depends on child model grid resolution). Default value: `RCF = 10000.`
- `RCF_IN`: scaling factor to avoid grid rounding errors as much as possible (depends on parent model grid resolution). Default value: `RCF_IN = 10000.`
- `PFIELD`: definition of the coupling fields. PFIELD has the form

```
PFIELD(.)= 'parent_channel','parent_object',
           'parent_tendency_channel','parent_tendency_object',
           'child_channel','child_object', 'representation',
            interpol_method, application_method, nudg_fac
```

  - `parent_channel`, `parent_object`, `parent_tendency_channel`, `parent_tendency_object`: specification of the data object which is the target of the exchange process:
    - If the tendency of a prognostic variable should be changed, `parent_tendency_channel` and `parent_tendency_object` contain the specification of the tendency object and `parent_channel` and `parent_object` the specification of the data field of the previous ('-1') time step.
    - If the field itself is changed directly, `parent_channel` and `parent_object` contain the specification of the field to change and `parent_tendency_channel` and `parent_tendency_object` are empty strings. In the special case, that the submodel MMD2WAY_PARENT needs to allocate the memory for the field itself, `parent_channel` equals `'mmd2way_parent'`.
  - `child_channel`, `child_object`: specification of the field in the child model coupled back to the parent model.
  - `representation`: *representation* of the parent object, if it is created in MMD2WAY_PARENT itself.
  - `interpol_method`: defines the interpolation method. Currently, only conservative remapping (`interpol_method = 1`) is implemented.
  - `application_method`: determining the application method. Currently, only `application_method = 0` for *input fields*, which should not be weighted in any way and grid point space (`application_method = 1`) are defined (see page 69).

  - `nudg_fac`: nudging factor determining the strength of the forcing of this specific parent field

- `lgrh` (experimental): use generalised humidity for back remapping to mirror procedure of INT2COSMO. Default value `lgrh = .FALSE.`.
- `lfreeslip` (experimental): allows for free (non-nudged) layers at the surface. Default value `lfreeslip = .FALSE.`.
- `rdefpc` (experimental): control level (in Pa) for vertical integration, should be the same as defined for INT2COSMO. Default value `rdefpc = 30000. Pa`, i.e., identical to the COSMO namelist default value.
- `lcpl_gs` (experimental): child to parent coupling in `global_start`. Default value: `lcpl_gs = .FALSE.`.
- `itype_VI` (experimental): type of vertical interpolation for child-parent coupling. Currently, only `itype_VI = 1`, i.e., interpolation via NREGRID is implemented
- `ldiagonly`: avoid iteration for vertical interpolation, if only diagnostic fields are coupled. Default value: `ldiagonly = .FALSE.`

4 Basic coupling setup

The diagrams in Fig. 8 and Fig. 9 sketch the sequence of operations in a coupled simulation.

For the basic coupling of the models, five phases are passed through, described in the following sections 4.1 to 4.5:

4.1 MMD setup

Before any submodel specific initialisation takes place, the Multi-Model-Driver (MMD) is initialised. The MMD library routines setting up the message passing interface (MPI) environment are called from the basemodels. The determination of the model topology and the communicator definition are explained in an extra manual about the MMD library[12], as these routines work inside the MMD library. As model topology, we understand the layout of all parent and child model dependencies and the distribution of the models on the available number of process entities (PEs) or MPI tasks.

The topology is determined by the MMD library namelist `MMD_layout.nml`, which is written by the run-script `xmessy_mmd` as determined by the user. The MMD library namelist file `MMD_layout.nml` is read, broadcasted and interpreted within the MMD library. All communicators for intra- and inter-instance communication are determined in accordance to the model topology.

4.2 Synchronisation

To ensure that all instances start, *restart* and stop at the same date and time, the date and time settings of all coupled models need to be synchronised. This is achieved, if one instance determines the timing of all other instances. If in each parent-child model pair the parent dictates the time setup,

[12]The MMD library manual is part of the same electronic supplement as this manual.

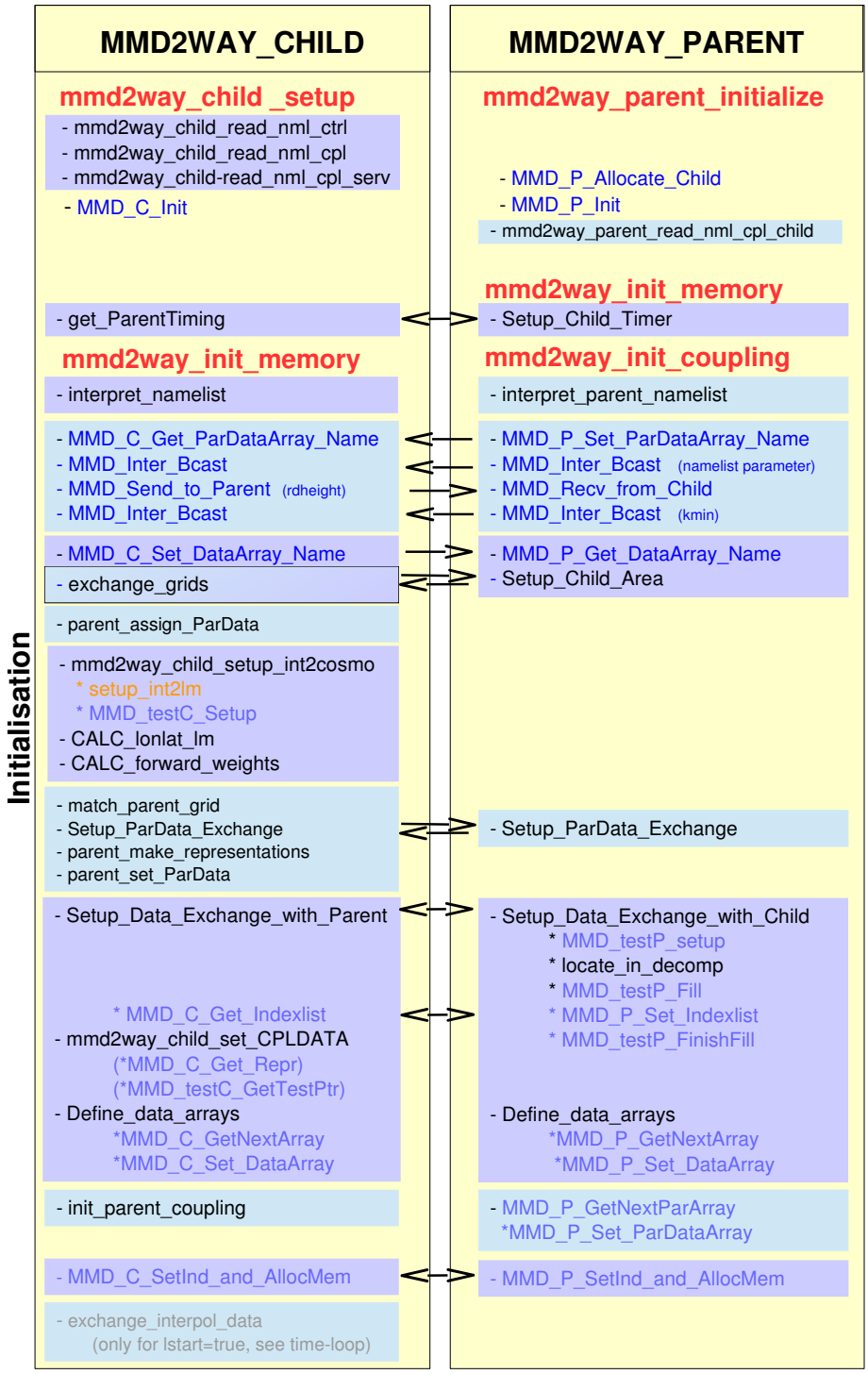

Figure 8: Call sequence of the 2-way on-line coupling routines in the parent and child submodels in ECHAM5/MESSy ($\rightarrow$ COSMO/MESSy)^n in the initial phase: Colour code of subroutine names: the MESSy entry points directly called by `messy_main_control`: red; MMD library routines: blue; original INT2LM routines: orange. The dark blue and light blue boxes indicate subroutine calls required for child-to-parent (1-way) and the parent-to-child coupling, respectively. Arrows indicate the direction of the data exchange between child and parent.

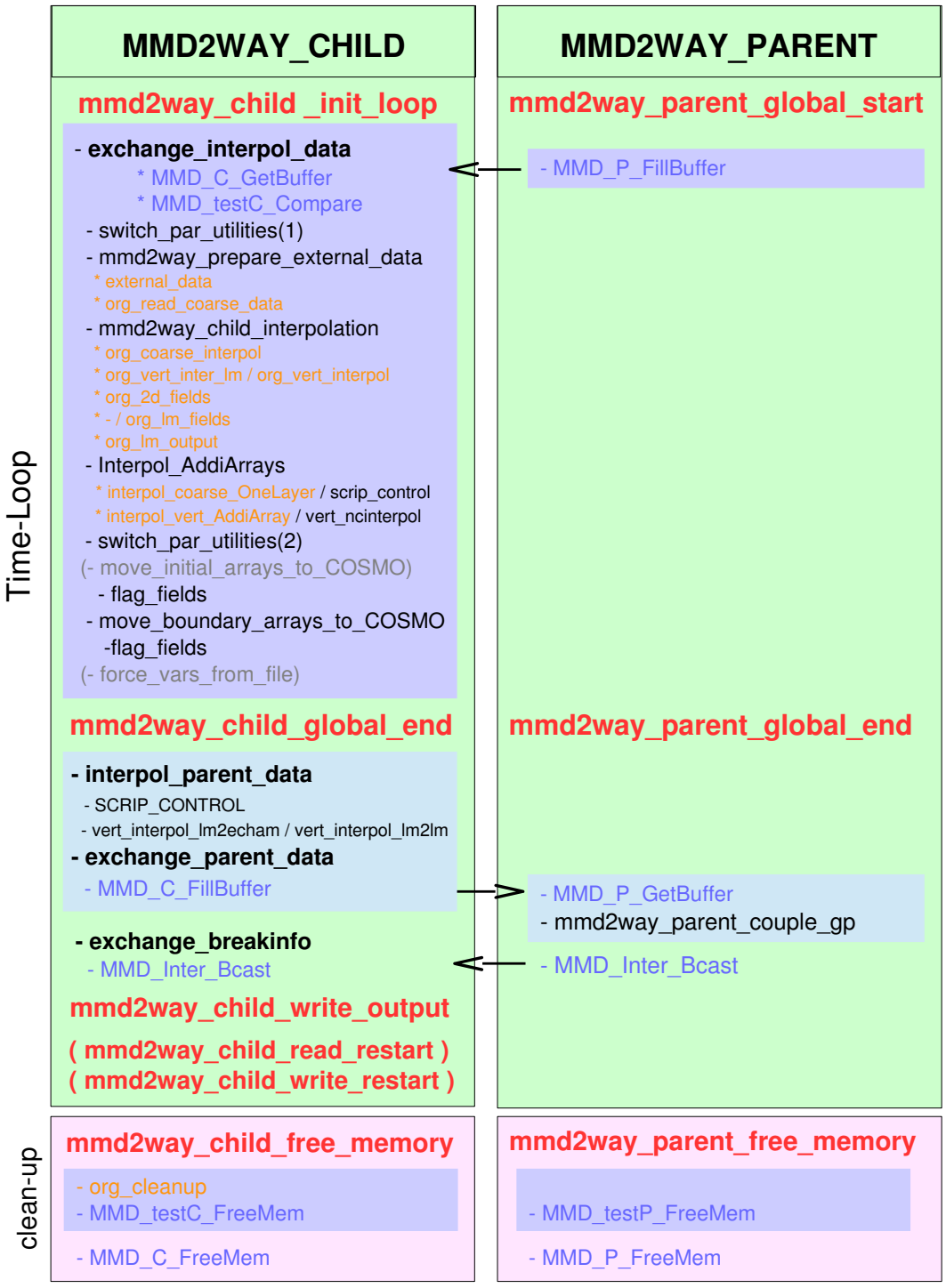

Figure 9: Call sequence of the 2-way on-line coupling routines in the parent and child submodels in ECHAM5/MESSy (→ COSMO/MESSy)^n during the time loop and in the finishing phase: Colour code of subroutine names: the MESSy entry points directly called by `messy_main_control`: red; MMD library routines: blue; original INT2LM routines: orange. The dark blue and light blue boxes indicate subroutine calls required for child-to-parent (1-way) and the parent-to-child coupling, respectively. Arrows indicate the direction of the data exchange between child and parent.

Table 1: LOGICAL switches determining the *restart* behaviour.

switch	description (if .TRUE.)
lbreak	interruption or stop of model simulation
lstop	stop of model simulation
l_rerun	write restart files
l_TRIGGER_RESTART	force model interruption

in the end the *patriarch* determines the timing of all instances. Consequently, the TIMER namelist of the *patriarch* determines the time setup of all child instances. However, it is important to note, that only the date and time are synchronised, but each model instance uses its own time step length.

4.2.1 Exchange of stop and *restart* triggers

To ensure the synchronisation of the models, information is exchanged during the integration phase of the simulation, whether the simulation is to be interrupted. Such an exchange is necessary, as, apart from the scheduled *restart*, exceptional simulation interruptions are triggered, e.g., by QTIMER, when the available scheduler time is consumed. Table 1 lists the LOGICAL variables, which determine the behaviour of the simulation.

For the synchronisation, an exchange of this information has to take place every parent model time step. The timing of the data exchange is realised by an *event*. The so-called `BREAK_EVENT` is set up in the child model triggering for each parent model time step the exchange of the required LOGICALs. If the *event* is scheduled, the subroutine `exchange_breakinfo` is called. The parent model sends the contents of its TIMER LOGICALs `lbreak`, `l_rerun` and `lstop`, via the MMD library routine `MMD_Inter_Bcast`. As this subroutine is not overloaded for LOGICALs, these are transferred as INTEGERs: `.TRUE.` is 1 and `.FALSE.` is 0. `lbreak` indicates, if a simulation is going to be interrupted, `lstop` shows, if the simulation will be stopped and `l_rerun` signals, if restart files shall be written at the end of the time step.

The contents of the parent models `lbreak` and `l_rerun` switches are directly written to the respective child model switches. There is only one case, which requires more thoughtful action: if the parent `lstop`-switch is `.TRUE.`, this indicates that the model simulation is going to be terminated. But in most cases, the child model uses a shorter time step than the parent model. In this case, the child models `stop_date` is not yet reached and the child model must not directly trigger a *restart* and exit, but rather continue the simulation until the `stop_date` is reached. Thus, if the parent indicates the end of the simulation, the variable `lServstop` is set `.TRUE.`, indicating, that the parent model terminates the simulation. If the child model itself does not yet indicate the end of the simulation (i.e., `lstop = .FALSE.`), `lbreak` will be reset to `.FALSE.`, to allow the child model to continue its calculation until the `stop_date` is reached.

Additionally, if `lbreak` is `.TRUE.`, `l_rerun` and `L_TRIGGER_RESTART` are set `.TRUE.`. Table 2 lists the settings of the respective LOGICALs of the child model dependent on the status of the parent model.

4.3 Data exchange initialisation

Different types of data are exchanged between the parent and the child model and vice versa. While each model requests the data required for the forcing of the same model, the child model determines

Table 2: Setting of child model LOGICALs, which determine the *restart* behaviour. If the simulation is being terminated (parent status "stop") `lbreak` and `l_stop` of the child model are usually set `.FALSE.`. Only if the parent and the child model use the same time step length, `lbreak` and `l_stop` are also `.TRUE.` for the child model.

parent status	lbreak	l_rerun	l_stop
continue	.FALSE.	.FALSE.	.FALSE.
write restart file, continue	.FALSE.	.TRUE.	.FALSE.
restart	.TRUE.	.TRUE.	.FALSE.
stop	.FALSE. (.TRUE.)	.TRUE.	.FALSE. (.TRUE.)

the timing of the forcing. Note: the frequence of the data transfer of the parent-to-child coupling is always equal to the frequence of the child-to-parent coupling.

- In contrast to the time settings of the instances, the timing of the data exchange during the coupling process is completely controlled by the child instance, i.e., the child namelist `&CPL_CHILD` determines the frequency of the data exchange for all exchanged data.
- Each model instance requests the data from its remote model, i.e.,
  - the child model instances request data for their specific model domain from the parent model. The child model namelists `&CPL_CHILD_ECHAM` or `&CPL_CHILD_COSMO` determine which data fields are exchanged. These requests are processed by the parent model during the initialisation phase.
  - the parent model instances request the required data from their children according to the `&CPL_PAR_CHILD` namelists (see Sect. 3.4). Note, that from each child model instance different data fields can be requested. If the same *target fields* are named and the child domains overlap, the child data is applied successively from each child model, giving the last coupled child the highest weight in determining the new value of the *target field*. These requests are processed by the child models during the initialisation phase.
- The child and the parent acquire POINTERs to the data fields required for the data exchange and (in a child instance) for the interpolation.
- Additionally, the buffers for the data exchange are allocated within the MMD library.

4.4 The data exchange

During the time loop or the integration phase the *exchange fields* are made available by the parent submodel MMD2WAY_PARENT (subroutine `MMD_P_FillBuffer`). These fields are copied by the child instance (subroutine `MMD_C_GetBuffer`) and interpolated according to the namelist settings. Afterwards, the child instance interpolates the data requested by the parent instance to the parent grid and sends these fields using the MMD library subroutine `MMD_C_FillBuffer`. Finally, these fields are copied by the parent model using the MMD library subroutine `MMD_P_GetBuffer`.

4.5 Finalisation phase

At the end of the integration coupling specific memory is deallocated.

5 Coupling of a child model instance to a parent model instance (1-way, child-to-parent coupling)

In case of a regional model instance driven by a parent model instance, data exchange in the direction from the parent to the child model is indispensible. Initial and boundary data for all prognostic variables and initial data for additional fields are required. Therefore, this section is dedicated to the 1-way data exchange between the parent and the child instance, as required for a simple 1-way coupling. First, the program flow on the child side and afterwards on the parent side are discussed.

5.1 The child instance

For the child, the MMD2WAY submodel MMD2WAY_CHILD provides everything required for the coupling of the child model to the parent model. All information required during the coupling process are contained within the variable `CplData`, which TYPE is a Fortran95 structure (`T_C_COUPLE_DATA`) and which is allocated to the actual number of *coupling fields*.

```
TYPE PTR_4D_ARRAY
   REAL(DP), DIMENSION(:,:,:,:), POINTER :: PTR => NULL()
END TYPE PTR_4D_ARRAY

TYPE CHAOBJ_NAMES
   CHARACTER(LEN=STRLEN_CHANNEL)    :: CHA = '' ! CHANNEL NAME
   CHARACTER(LEN=STRLEN_OBJECT)     :: OBJ = '' ! OBJECT  NAME
END TYPE CHAOBJ_NAMES

TYPE T_C_COUPLE_DATA
   ! CHANNEL AND CHANNEL OBJECT NAMES IN PARENT AND CHILD
   TYPE(CHAOBJ_NAMES)                    :: PARENT
   TYPE(CHAOBJ_NAMES)                    :: CHILD
   ! ORDER OF AXES IN REPRESENTATION ('X','Y','Z','N')
   CHARACTER(LEN=4)                      :: AXIS= ''
   ! DIMENSION LENGTH
   INTEGER, DIMENSION(4)                 :: ldimlen=0
   ! INTERPOLATION METHOD (only valid for arrays not included in vartab)
   ! 1.CHAR 'Q' quadratic; 'L': linear; 'M' match interpolation;
   !        'C' conservative remapping
   !
   ! 2.CHAR if 'T' positive definiteness  is required
   ! 3.CHAR if 'T' monotonicity           is required
   ! 4.CHAR if 'V' vertical interpolation is required
   !        if 'W' vertical interpolation via NCREGRID is required (only
   !               possible with 'C' horizontal interpolation and only for
   !               additional fields
   CHARACTER(LEN=4)                      :: C_INTERPOL
   ! INPUT FIELD DELIVERED BY MMD
   REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr_in  => NULL()
   ! INTERMEDIATE FIELD OF INT2COSMO
```

```
    REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr_i2c => NULL()
    ! POINTER(S) TO COSMO/MESSy FIELD(S): DIMENSION == number of time levels
    TYPE(PTR_4D_ARRAY), DIMENSION(:), POINTER :: cosmo => NULL()
    ! POINTER TO COSMO/MESSy BOUNDARY FIELDs:
    !  (DIMENSION IS ALWAYS TWO FOR THE TWO BOUNDARY LAYER TIME LEVELS)
    TYPE(PTR_4D_ARRAY), POINTER, DIMENSION(:) :: cosmo_bd => NULL()
    ! RANK OF CHILD FIELD (WITHOUT TIME LEVEL DIMENSION)
    INTEGER                                   :: rank        = 0
    ! INDICATOR, IF FIELD IS IN VARTAB
    LOGICAL                                   :: lvartab     = .FALSE.
    ! NAME OF VARIABLE IN VARTAB
    CHARACTER(LEN=10)                         :: vartab_name = ''
    ! INDEX OF FIELD in var_lm
    INTEGER                                   :: vartab_idx   = 0
    ! INITIAL FIELDS REQUIRED ?
    LOGICAL                                   :: L_INITIAL   = .FALSE.
    ! BOUNDARY FIELDS REQUIRED ?
    LOGICAL                                   :: L_BOUND     = .FALSE.
    ! INPUT FIELD REQUIRED ?
    LOGICAL                                   :: L_INPUT     = .FALSE.
    ! NAME OF REPRESENTATION
    CHARACTER(LEN=STRLEN_MEDIUM)              :: C_REPR      = ''
    ! Number of parent field (requested for shortcut test)
    INTEGER                                     :: scn = -99
  END TYPE T_C_COUPLE_DATA
  TYPE (T_C_COUPLE_DATA), DIMENSION(:), ALLOCATABLE :: CplData
```

This structure contains

- the *channel* and *channel object* names of the *exchange fields* for the parent model (`TYPE(CHAOBJ_NAMES) :: PARENT`) and
- the *channel* and *channel object* names for the child model (`TYPE(OCHOBJ_NAMES) :: CHILD`).
- information about the *dimensions* of the fields:
  - The *axis string* (`AXIS`) indicates the order of the 'X', 'Y', 'Z' and 'N' direction, and
  - `ldimlen` contains the length of these four *dimensions*[13].
- `C_INTERPOL` is the flag specifying the interpolation method.
- The structure contains four POINTERs or *POINTER ARRAYs* for the access to the different data fields during the interpolation procedure and to the *target fields*. Depending on the source (*exchange field* or from external data) and the *target field* (i.e., *boundary* or *initial* and *input*

[13]These are properties already defined and provided by the CHANNEL submodel. See the CHANNEL manual available in the electronic supplement of Jöckel et al. (2010) for further information.

fields) not all four POINTERs or *POINTER ARRAYs* are used for all *coupling fields*[14]:

- The first POINTER (`ptr_in`) is used for the *in-fields*, i.e., the raw data sent from the parent model.
- The second POINTER (`ptr_i2c`) is associated to the *intermediate fields* generated by the horizontal (and vertical) interpolation within INT2COSMO.
- The *POINTER ARRAY* `cosmo` is associated with the *target field* in the COSMO/MESSy model. For prognostic variables the dimension of the *POINTER ARRAY* is given by the number of time levels. Each of the POINTERs in the *POINTER ARRAY* is associated to one time level of the *target field*. For diagnostic variables the array dimension is always 1.

  For instance, the prognostic field for the temperature in COSMO is dimensioned by the three space dimensions and a time level dimension. Depending on whether a two or three time level integration scheme is used, this fourth dimension is allocated to 2 or 3. All time levels have to be made accessible in MMD2WAY_CHILD for the respective *target field*. Thus the *POINTER ARRAY* `cosmo` is also dimensioned according to the number of the time levels required by the integration scheme. This yields the POINTERs `CplData(ii)%cosmo(nt)%ptr`, where `nt` is an index ranging from 1 to the number of time levels used in the COSMO model, allowing to access the different time levels of the *target field*. Thus, `nt` is one of the time level indices `nnew`, `nnow` and `nold`, respectively. In contrast, a diagnostic variable does not depend on the integration scheme, thus one POINTER is sufficient to access a diagnostic *target field*.
- If boundary data is required for a field, the *POINTER ARRAY* `cosmo_bd` is allocated to a length of 2 according to the two time levels required for the boundary data in the COSMO/MESSy model. Each of the POINTERs is associated to one level of the boundary data array.

To actually perform the coupling, additional information is required:

- The `rank` of the field,
- the information, if a field is part of the variable table in INT2COSMO (LOGICAL `lvartab`), i.e., if the field is an *INT2COSMO inherent field*,
- the name in the variable table (`vartab_name`), if `lvartab = .TRUE.`, and
- the index of the variable in the variable table of INT2COSMO (`vartab_idx`), if `lvartab = .TRUE.`,
- the LOGICALs `L_INITIAL`, `L_BOUND` and `L_INPUT` indicating if initial, boundary or input data is required, and
- the string `C_REPR`.

[14]Note: the different meaning of the different fields:

- *exchange fields* are those fields exchanged with the parent, they are not necessarily associated to a *target field*, as they might also be required for the interpolation and not as input for the child model.
- *coupling fields* are all those fields contained in the variable `CplData`, i.e., either *exchange fields* or fields additionally provided by INT2COSMO, e.g., calculated from the external data.
- *target field* can be an *input* or *initial field*, or the respective *boundary field* for prognostic variables.

The meaning of the individual fields is clarified within the remainder of the child description and in the glossary.

The meaning of these variables was already illustrated in the section about the namelist (Sect. 3) and will become clearer in the remainder of the child model description.

5.1.1 Initialisation Phase

The main entry point for submodel initialisation in MESSy is `messy_initialize`. In contrast to this, MMD2WAY_CHILD uses an even earlier entry point, i.e., `messy_setup`. This is necessary, as the *patriarch* determines the date und time setup, of all instances in the cascade, which is performed very early during the model setup. Thus a very early entry point for MMD2WAY was required. The second entry point used from the MESSy infrastructure is `messy_init_memory` in which the INT2COSMO setup is completed, all required data is allocated and the first coupling is performed.

5.1.1.1 mmd2way_child_setup:

This subroutine performs the basic setup of MMD2WAY_CHILD and defines the date and time setup of the child instance.

- **Setting of model wide variables:** At the beginning two LOGICALs need to be set, which determine the information flow in the basemodel and the MESSy generic submodels:
  - The LOGICAL variable `L_IS_CHILD` defined in `messy_main_data_bi` is set `.TRUE.`. It is used in the COSMO model itself to switch off certain parts of the code dealing with the import of initial and boundary data (in `src_input.f90`).
  - `lforcedtime` is required for the synchronisation of the parent and child model instances. Setting `lforcedtime = .TRUE.` prevents the calculation of the trigger of the RERUN *event* `l_rerun` in `timer_global_start`. If `l_rerun` would be determined in the child instances TIMER itself, the child model could finish unnoticed by the parent model and a dead lock in MPI would occur resulting in a model hang up.
- **Namelist input:** Subsequently, the MMD2WAY_CHILD namelists are read and the content is written to the log-file.
  - First, the `&CTRL`-namelist is read in subroutine `mmd2way_child_read_nml_ctrl`.
    The `&CTRL`-namelist contains five entries. One switch, which forces the original output of int2lm to be written (`l_I2Cori_output`) and the respective *event* (`WRITEI2C_IOEVENT`) determining the frequency of this output. The additional three variables define whether (`l_forcevars`) certain variables (`forcevars`) should be overwritten at the start of a new model simulation. This is required, e.g., for a more comprehensive soil moisture initialisation. `forcefile` provides the path and the filename of the file containing the data required for this procedure (see Sect. 3.1).
  - Second, the `&CPL_CHILD`-namelist is read in subroutine `mmd2way_child_read_nml_cpl`. This namelist defines the two `IO_TIME_EVENT`s `CPL_IOEVENT` and `READEXT_IOEVENT` scheduling the coupling dates and the dates at which new external data should be read (see Sect.3.2).
  - Third, the parent model specific coupling namelist (`&CPL_CHILD_ECHAM` or `&CPL_CHILD_COSMO`, see Sect. 3.3) is read in the subroutine `mmd2way_child_read_nml_cpl_serv`. Whether the ECHAM5/MESSy or the COSMO/MESSy specific namelist is read, is determined with the help of the MMD library function `MMD_C_GetParentType` (located in `mmd_child.f90`).

As reading and printing is performed by one task only, the namelist content is broadcasted to all tasks afterwards.

- **Initialisation of MMD library:** In addition to the child submodel setup, the child model part of the MMD library has to be initialised. This is done by the MMD library routine `MMD_C_Init`. Within this subroutine the C-language group communicators are determined and the number of PEs covered by the parent model is retrieved. Variables of the MMD library internal information structure (named "`Me`"), of which the dimensions depend on the number of parent PEs are allocated within `MMD_C_Init`[15].

- **Adjustment to parent time setting:** To get a meaningful simulation, the time and date setups of the coupled instances need to be synchronised. To achieve this, two important informations are exchanged between the child and the parent in the subroutine `get_ParentTiming`:

  1.) The coupling interval determined in the child namelist `CPL_CHILD` is sent to the parent model:
  In the subroutine `get_ParentTiming` first the time interval of the coupling *event* `CPL_IOEVENT` is converted into seconds, as this is the unambiguous unit to exchange between the two models. The number of seconds equivalent to the coupling interval is sent to the parent, which accordingly defines a coupling *event*. If the coupling interval is not a multiple of the parent model time step length the simulation is terminated.

  2.) The parent model sends the complete date and time settings to the child model in order to initialise the date and time of the child basemodel:
  The `current_date`, the `resume_date`, the `start_date` and the `stop_date` are sent from the parent to the child. The child re-initialises its date settings according to the parent dates. However, the following has to be taken into account:

  * If the child simulation is starting (`lstart=.TRUE.`), the child's `start_date` is set to the `resume_date` of the parent model. This is done, as it is possible, that the parent performs a *restart*, while the child is started for the first time. Note that, if `lstart` is also `.TRUE.` for the parent, the `resume_date` and the `start_date` are equal anyway. The `current_date` of the parent is copied to the `current_date` of the child.
  * If the child simulation is continued (`lresume=.TRUE.`), the `current_date` is not set at all, as it is set by the TIMER later on anyway. More important, the child's `resume_date` is not identical to the parent model `resume_date`. It needs to be calculated from the parent model's `resume_date`. This is necessary, as the restart files for child and parent are usually not output at the same time: They both stop after `l_rerun` was set `.TRUE.`, thus the child's `resume_date` is behind the parent's `resume_date` by the difference of the parent and the child time step lengths. Based on this, the child's `resume_date` is calculated from the time step lengths of the two instances and the parent's`resume_date`.

  The child's `stop_date` is always a copy of the parent's `stop_date`. Additionally, the COSMO variables `hstop` and `nstop` are calculated.

  The COSMO/MESSy date and time settings and counters are re-initialised within the subroutine `messy_timer_COSMO_reinit_time` provided by `messy_main_timer_bi`.

  Additionally to the dates and times, the time step length of the parent is sent to the child. This is important as the parent time step sets the interval for the check-pointing. From this information the so-called `BREAK_IOEVENT` is defined.

[15]The MMD library routines are described in detail in the MMD library manual, which is part of the same electronic supplement as the MMD user manual.

Last but not least, the TIMER-Manager needs to be initialised in case of a new simulation (`lstart = .TRUE.`) at this point, directly after the date and time setting.

5.1.1.2 mmd2way_child_init_memory

The second part of the initialisation takes place in `mmd2way_child_init_memory`, as the memory allocation and field definitions are to be conducted here.

- **Initialision of *events*:** Due to technical reasons, the *events* themselves can not be initialised before the MESSy entry point `messy_init_memory`. Thus first of all, the four TIMER *events*

  (i) for the coupling (`CPL_EVENT`),

  (ii) for reading the external data (`READEXT_EVENT`),

  (iii) for the output interval of INT2LM original output (`WRITEI2C_EVENT`) and

  (iv) for the check-pointing (`BREAK_EVENT`)

  are initialised. The `BREAK_EVENT` is used to encounter every parent time step, whether the simulation is interrupted at the end of the time step. This is inevitably necessary to ensure that the parent and the child are interrupted at the same time. Otherwise, if the check-pointing information would not be exchanged each parent time step, one instance hangs up in MPI communication, because the other instance was interrupted and does not answer the MPI calls anymore. This implies that the parent time step length needs to be a multiple of the child time step length.

- **Namelist interpretation:** After these preparations, the contents of the MMD2WAY_CHILD namelist are interpreted. The subroutine `interpret_namelist` serves two purposes:

  – The wildcards in the child *channel object* names are analysed and translated into individual *exchange fields*.

  For the tracer channel `tracer_gp` two exceptions are made from the general application of the wildcards:

  * The tracers defined by the COSMO model itself (i.e., the humidity / water variables), are not requested from the parent model and have to be listed individually;
  * If ECHAM is the parent model, the liquid and ice tracers (usually defined by SCAV) are automatically omitted.

  – The namelist settings for the *mandatory fields* are cross-checked with the COSMO variables `yvarini` and `yvarbd`:
  The LOGICALs `L_INITIAL` and `L_BOUND` are set `.TRUE.`, if the field is required by the COSMO model. Furthermore, fields listed in `yvarini` or `yvarbd`, but not in the coupling namelist, are added to the variable `CplData`. These are data fields, which are calculated by INT2COSMO, but do not require direct input from the parent. Examples are the external parameters root depth, leaf area index or orography. Note: *mandatory fields* requiring an input field from the parent model, have to be listed in the namelist. Otherwise the information about the *channel* and *channel object* names in the parent model are missing.

  The settings for the INT2COSMO *inherent fields* can partly be overwritten by the namelist settings in `CPL_CHILD_XXX`. Later on in the subroutine `mmd2way_child_set_CplData` the interpolation flags set in the vartab of INT2COSMO are overwritten, if interpolation flags are

given in the `CPL_CHILD_XXX`. In this way, the INT2COSMO *inherent fields* can also be horizontally remapped by conservative remapping using SCRIP (flag 'C') or vertically transformed by NREGRID (flag 'W').

`CplData` contains data of different sources. The first part of `CplData` stems from the namelist and lists the *exchange fields*, i.e., fields that are provided by the parent. During the namelist interpretation other fields are added to `CplData`. These fields are calculated by INT2COSMO and are required by the COSMO model. Both types of fields are summarised by the term *coupling fields*. Therefore, two important numbers characterising `CplData` are determined during this analysis:

- `NEXCH` is the number of fields that need to be exchanged with the parent.
- `NCOPY` is the dimension of the variable `CplData` containing all *coupling fields*, i.e., the *exchange fields* and the ones determined from external data and copied from INT2LM to the basemodel variables.

Additionally, the subroutine `interpret_namelist` checks for the interpolation methods required. If conservative remapping using SCRIP or vertical transformation via NREGRID is required, the LOGICAL `l_i2cscrip` is set `.TRUE.`. This variable is used later on, to determine, if the calculation of the weights for the conservative remapping is necessary.

At the end of the subroutine `interpret_namelist` the final list of all `CplData` fields is output to the log-file.

- **Definition of the test field:** The MMD library includes the possibility to test the horizontal grid exchange. Therefore a *channel object* for the `test_array` (*channel* name = `'mmd2way_child'`, *channel object* name = `'Test_Ar'`) is created.

- **Exchange of field information with parent instance:** The information set in the namelist relevant for the MMD library and the parent, i.e., the *channel* and *channel object* names and the *representation* of the respective *exchange field*, are forwarded to the MMD library using the MMD library subroutine `MMD_C_Set_DataArray_Name`. The subroutine is called for each of the *exchange fields*. Within the library, the MMD internal information structure on child and parent side are set up within this subroutine and its counterpart (`MMD_P_Get_DataArray_Name`) of the parent. The end of the list is indicated by the presence of the optional parameter `LastEntry` which must be set `.TRUE.` to end the list.

- **exchange_grids:** The parent automatically determines the segment of the parent domain required for the interpolation in INT2COSMO using the geographical information about the child domain. This is provided by the child within the subroutine `exchange_grids`. The local fields `rlon` and `rlat` containing the geographical coordinates of the grid points are gathered, yielding one non-decomposed field and sent to the parent. In addition, the number of *exchange fields* is sent to the parent. From the geographical information the parent calculates the size of the domain, which is required to interpolate the *initial* and *boundary fields*. The grid definition for the in-coming data fields is afterwards sent back from the parent to the child. The child uses this information to define the *in-grid* and consequently determines the *dimensions* / the *representation* of the *in-fields* (`ptr_in`). The grid definition received from the parent replaces the `&grid_in` namelist of INT2LM in case of the on-line coupling. Consequently, the following (INT2COSMO) parameters are defined by the parent and sent to the child:

  - PARAMETERs describing the (rotated) parent grid and the type of the soil water content and soil temperature:

`startlat_tot`, `startlon_tot`, `endlat_tot`, `endlon_tot`, `pollat`, `pollon`, `dlat`, `dlon`, `ie_coarse`, `je_coarse`, `ke_coarse`, `ke_soil_coarse`, `itype_w_so_rel`, `itype_t_cl`[16].

- If the parent is a COSMO/MESSy model (`llm2lm = .TRUE.`), additionally the information about the vertical coordinate system and the reference atmosphere of the parent COSMO model setup are required: `vcflat`, `p0sl`, `t0sl`, `dt0lp`, `delta_t`, `h_scal`, `svc1`, `svc2`, `ivctype`, `irefatm`.
- The vertical coordinates (`vct` for ECHAM5/MESSy as parent and `vcoord_in%sigm_coord` or `vcoord_in%vert_coord`, if COSMO/MESSy is parent) and the depth of the soil layers (`czmls_in`) are exchanged.
- Next, the child receives two fields containing the latitude and longitude information for the *in-fields* (`latitude_in` and `longitude_in`).
- Finally, if ECHAM5/MESSy is the parent, the hybrid coefficients for the interface levels `ak_in` and `bk_in` are set using the vertical coordinate variable `vct`, which has already been sent by the parent. Subsequently, the hybrid coordinates for the full levels (`akh_in` and `bkh_in`) and the differences of the interface level hybrid coordinates (`dak_in` and `dbk_in`) are calculated. If COSMO/MESSy is the parent, all four coefficients are calculated by the subroutine `calc_hybrid_coeff`.

- **mmd2way_child_setup_int2cosmo:**
  This subroutine performs the setup of INT2COSMO:
  - At the beginning some switches originally determined in the INT2LM `&CONTRL` namelist are defined:
    - The LOGICALs indicating the driving model (`lgme2lm`, `lec2lm`, `lhm2lm`, `lcm2lm` and `llm2lm`) are set to `.FALSE.`; if ECHAM5/MESSy is parent `lcm2lm` is `.TRUE.`; if a COSMO/MESSy model is parent `llm2lm` is `.TRUE.`.
    - The LOGICAL ARRAYs `lushift_in` and `lvshift_in` indicating if and which type of a staggered grid is used for the horizontal wind components are set: For ECHAM5/MESSy as parent all entries are `.FALSE.`, for the COSMO/MESSy model as parent it is: `lushift_in = (.TRUE. , .FALSE.)` and `lvshift_in = (.FALSE., .TRUE.)`. Additionally, the switches `lcm_hgt_coor` and `lcm_pres_coor` are set `.FALSE.` in both cases.
  - The INT2COSMO LOGICAL variable `linitial` is set `.TRUE.` for the first time step of a new or restarted simulation, as only for this time step initial data need to be calculated, which is indicated by `linitial` in INT2LM. `lcomp_bound` is another INT2LM switch, indicating that boundary data need to be calculated. It is `.TRUE.` except for the first time step.
  - Based on the *in-grid* definition and the longitudes and latitudes of the *in-fields* as received in the subroutine `exchange_grids`, the staggered longitudes and latitudes (`slongitude_in` / `slatitude_in`) are calculated. As ECHAM5 does not use a staggered grid, `slatitude_in` and `slongitude_in` are set to `latitude_in` and `longitude_in`.
  - Subsequently, the original INT2LM subroutine `setup_int2lm` is called. The same code is processed, apart from the initialisation and decomposition of the grid and the initialisation of many namelist parameters, which are determined directly by the setup of the COSMO/MESSy model during the on-line coupling initialisation. The changes made to

[16]For further details about the namelist parameters see the INT2LM documentation: *http://www.cosmo-model.org/content/model/documentation/core/cosmoInt2lm.pdf*: last access: 11.10.2016

the original INT2LM code in order to implement it as MESSy sub-submodel (i.e., directly coupled to COSMO/MESSy) are described in Sect. 7.

- In `setup_int2lm` the INT2COSMO namelists are read, thus the INT2COSMO LOGICAL switch `lbd_frame_cur` is set according to the namelist parameter `lbd_frame` after processing `setup_int2lm`.
- Finally, as the local dimensions of the parallel decomposed *in-fields* have been calculated in `setup_int2lm`, the MMD `test_array` can be allocated by the MMD library subroutine `MMD_testC_Setup` with the corresponding dimensions.

At this point the initialisation of `INT2COSMO` is complete.

- **CALC_lonlat_lm**
  This subroutine calculates the (geographical and rotated) longitude and latitude fields in grid mid points and interfaces for the INT2LM specific grid (which is larger by one grid box in each direction compared to the COSMO grid):
  - rotated longitude / latitude field on grid mid points (`lon_lm`, `lat_lm`)
  - geographical longitude / latitude field on grid mid points (`geolon_lm`, `geolat_lm`)
  - rotated longitude / latitude field on grid interfaces (`loni_lm`, `lati_lm`)
  - geographical longitude / latitude field on grid interfaces (`geoloni_lm`, `geolati_lm`)

- **CALC_forward_weights**
  For MMD v2.0 the conservative remapping via SCRIP for *additional fields* and INT2COSMO *inherent fields* as well as the vertical interpolation via NREGRID (both by calling the generic MESSy submodel GRID_TRAFO) have been implemented. If conservative remapping is required, MMD2WAY_CHILD needs to calculate the remapping weights. The subroutine `calc_forward_weights` performs this calculation by,
  - firstly, defining the *in-grid* and the INT2LM intermediate grid as geo-hybrid grids (see Manual of GRID),
  - secondly, converting the geo-hybrid grids to the data format required by SCRIP by calling `CALC_SCRIPDATA`, and
  - thirdly, calculating the weights by calling the subroutine `CALC_SCRIP_WEIGHTS`.

  The conservative remapping is invoked by placing 'C' as interpolation method in the namelist.

- **Setup_data_exchange_with_Parent:** One of the crucial points of the efficient field exchange by MMD is the index list, which directly associates for each child model PE the grid points of the parallel decomposed *in-field* with the grid points and PE of the parallel decomposed parent grid. The index list consists of six entries for each grid point of the local child model *in-grid*[17]:
  1.) the first horizontal index of the grid point in the local grid of the parent PE_p (i_p)
  2.) the second horizontal index of the grid point in the local grid of PE_p (j_p),
  3.) the first horizontal index (i_c) in the local child model *in-grid*,
  4.) the second horizontal index (j_c) in the local child model *in-grid*,
  5.) the process entity (PE_c) on which the local child grid point is located,

[17] i.e., the index list consists of 6 entries per number of coupled grid points: `index_list(6,number of grid points)`

6.) the parent PE (PE_p) on which the respective grid point is located[18].

This grid association is performed by the parent instance. Thus the child has to send its grid definition and decomposition to the parent:

- On each child PE the longitudes and latitudes of the *in-fields* `latitude_in` and `longitude_in` are written to the local fields `my_lon` and `my_lat`.
- These fields are gathered on one PE in the 3D fields (`all_lon(nx,ny,nPE)` and `all_lat(nx,ny,nPE)`) with `nx`, `ny` number of grid points in x and y direction and `nPE` number of child PEs.
- Finally, the 3D fields are sent to the parent model for further calculations.

Due to their structure, the fields inherently contain the information required to set up the index list. The third index gives the number of the child model PE and the first and second index are equal to the indices in the local grid of the respective child model PE, where the point of the given geographical coordinates is located.

After receiving this list, the parent model associates the (local) source points for each local child grid point to its own parallel decomposed grid and sends back the list containing the sextuples associating the child and the parent model grid points with each other. This list is received and analysed by the child part of the MMD library within the subroutine `MMD_C_Get_Indexlist`, which is called at the end of this subroutine (compare Fig. 8).

- **mmd2way_child_set_CplData:**
  So far, only those parts of the variable `CplData` have been initialised, which are set by the namelist. In the subroutine `mmd2way_child_set_CplData` the POINTERs and *POINTER ARRAYs* to the data fields are associated or allocated. Three different types of *coupling fields* are distinguished:

  A) fields, which require an *in-field* from the parent, which is remapped and afterwards copied to the *initial*, *boundary* or *input field*;

  B) fields exchanged with the parent, which have no direct target variable: one example is the surface geopotential, which is required for the remapping itself, but has no corresponding *target field* in the COSMO/MESSy model. This is indicated by setting the corresponding child *channel* name to `'#XXX'` (see `FIELD(10)` in the `&CPL_CHILD_ECHAM` namelist in Fig. 4);

  C) fields, which result from the INT2COSMO interpolation/preprocessing routines but have no corresponding *in-field*. For instance, all external data fields as orography, leaf area index and so on. These fields are located at the end of the `CplData` variable (indices `NEXCH+1` to `NCOPY`).

  The subroutine contains one loop over all entries of `CplData`. The individual entries are indicated by the loop index `ii` in the following. The loop is split into five logical units:

  1. association / allocation of the `CplData(ii)%cosmo` *POINTER ARRAY*;
  2. association / allocation of the `CplData(ii)%cosmo_bd` *POINTER ARRAY*;
  3. inquiry, if the *coupling field* is an *INT2COSMO inherent field*. If 'yes',

[18] A more detailed example is provided in the MMD library manual, which is part of the same electronic supplement as this manual.

- the structure component `CplData(ii)%lvartab` is set `.TRUE.`,
- the `CplData(ii)%vartab_name` is set, and
- the index of the variable in the INT2COSMO variable table (`CplData(ii)%vartab_idx`) is set.

4. association / allocation of the *intermediate fields* (`CplData(ii)%ptr_i2c`);
5. association / allocation of the *in-fields* (`CplData(ii)%ptr_in`).

At the beginning of the loop, the POINTERs `CplData(ii)%ptr_i2c` and `CplData(ii)%ptr_in` are NULLIF(Y)ied and the structure components `CplData(ii)%rank`, `CplData(ii)%lvartab`, `CplData(ii)%vartab_name` and `CplData(ii)%vartab_idx` are initialised by the default values `0`, `.FALSE.`, `''` and `0`, respectively. In the following the allocation or determination of each of the above listed `CplData` entries is described in detail:

1. **determine** `CplData(ii)%cosmo`**:**
   This part is skipped for entries with child model *channel* name `'#XXX'` as this indicates that the *exchange field* is only required in INT2COSMO.

   For the determination of the memory for the COSMO/MESSy *target field* (`CplData(ii)%cosmo`) basically two times two different cases (in all combinations except for B2A2) have to be taken into account:

   A) The fields can either be
      A1) diagnostic or
      A2) prognostic,

      which require different memory allocation procedures.

   B) The fields are
      B1) either already allocated by another MESSy submodel or the basemodel , or
      B2) required to be allocated within the MMD2WAY_CHILD submodel itself.

   Figure 10 comprises a flow chart showing the basic procedure for the association of the `CplData(ii)%cosmo` *POINTER ARRAY*

Regarding A) The nature of the respective variable (diagnostic or prognostic) determines the dimension of the *POINTER ARRAY* `CplData(ii)%cosmo`:

A1) For a diagnostic variable the dimension is 1, as only one *target field* exists.

A2) For the prognostic variables the dimension equals the number of time levels of the time integration scheme used. For instance, for the leap frog scheme the dimension is 3, whereas for a two-time level scheme (e.g., Runge-Kutta) it is 2. This is due to two reasons:

- For an integration scheme with more than 2 time levels, more than 1 time level needs to be initialised by MMD2WAY_CHILD.
- In the COSMO model prognostic variables are allocated with an extra rank for the time level. For the sake of computational efficiency, the indices indicating the different time levels (`nnew`, `nnew` and `nold`) are shifted instead of copying the newly integrated value to the old field at the beginning of each time step. Thus, it is not a priori known which time level (index in the prognostic field) is required at a specific point in time. Hence, all time levels must be available for the coupling.

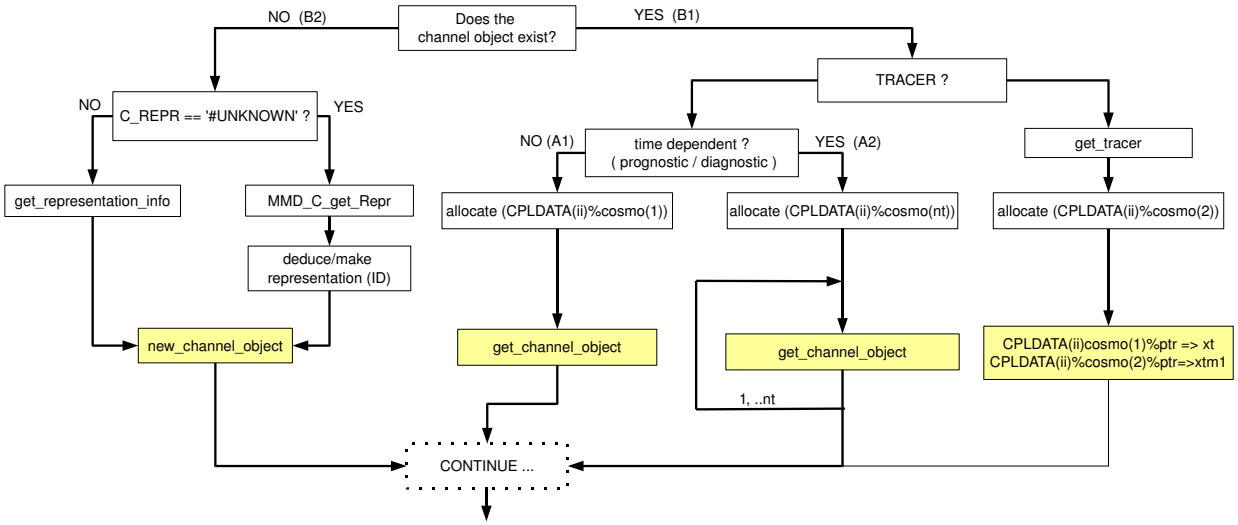

Figure 10: Flow-chart illustrating the association of the `CplData(ii)%cosmo` *POINTER ARRAY*. The labels in brackets (A1, A2, B1 and B2) refer to the respective cases listed in the text. The yellow boxes point to those subroutine calls in which the individual POINTERs of the *POINTER ARRAY* `CplData(ii)%cosmo` are finaly associated.

Each of the POINTERs of the *POINTER ARRAY* `CplData(ii)%cosmo` (`CplData(ii)%cosmo(nt)%ptr`, with `nt` being the index for a time levels) is associated to one time level of the prognostic variable. Thus, the correct time level can be addressed by the indices (`nnew` and `nnow`) usually used in the COSMO model to access the correct time levels.

Regarding B) The child model *channel* name includes the information, if MMD2WAY_CHILD needs to allocate the required memory itself (`'mmd2way_child'` as child *channel* name in the namelist), or if the *target field* is already allocated by other COSMO/MESSy submodels or the basemodel (all other cases). In the latter case the POINTERs of the `CplData(ii)%cosmo` *POINTER ARRAY* are associated to the already existing memory:

B1) The required *channel object* exists already:
First, the nature of the *channel object* is inquired by looking for the *channel object attribute* `number_of_timelevels` using the CHANNEL subroutine `get_attribute`.

A1) If the attribute does not exist, the variable is of diagnostic nature and the *POINTER ARRAY* `CplData(ii)%cosmo` is allocated to the dimension 1 (as only one POINTER is required for a diagnostic variable).
Afterwards, `CplData(ii)%cosmo(1)%ptr` is associated to the respective memory by calling the CHANNEL subroutine `get_channel_object`[19].
Then, the rank (`CplData(ii)%rank`) of the field is acquired by calling the MMD2WAY_CHILD subroutine `get_rank`. In the subroutine `get_rank`, first, the *channel object representation ID* is determined by calling the CHANNEL subroutine `get_channel_object_info` with the input parameters *channel* and

[19]See the CHANNEL manual, which is part of the electronic supplement of Jöckel et al., 2010.

channel object name. Second, with the *representation ID*, the `rank` of a *channel object* with this *representation* is found out by calling `get_representation_info` with the *representation ID* as input and the `CplData(ii)%rank` as output parameter.

A2) If the attribute exists, the *channel object* is of prognostic nature and `CplData(ii)%cosmo` is allocated to the number of time levels as denoted by the attribute (`timelev` is the return value of the subroutine `get_attribute` containing the number of required time levels). Afterwards, a loop over the number of time levels is executed associating for each time level one POINTER of the *POINTER ARRAY* to one time level of the *target field* using the subroutine `get_channel_object`. Note: for all prognostic variables individual *channel objects* for the single time levels must exist. Finally, the `CplData(ii)%rank` is determined as in the diagnostic case.

XT) A special case exists for tracers:
In contrast to the prognostic COSMO variables, the tracer structure provides individual variables for all time levels[20], such that the index rotation instead of the copying of one time level to the other at the end of one time step is not possible for the tracers. Consequently, tracers must be treated differently:

(a) First, the tracer index `idt` is inquired by the TRACER subroutine `get_tracer`.

(b) Next, `CplData(ii)%cosmo` is always allocated to 2 and the first POINTER of the *POINTER ARRAY* always points to the current tracer field `xt` of that specific tracer. The target of the second POINTER depends on the integration scheme. For a 2 time level scheme it points to `xtm1` and for a 3 level scheme to `xtf`.

(c) The rank of a tracer is always 3, thus `CplData(ii)%rank` is set to 3 for tracers and

(d) the flag in the TRACER meta-structure indicating that a tracer is already initialised (`ti_gp(idt)%tp%meta%cask_i(I_MMD_INIT)`) is set to `ON` at simulation start (`lstart = .TRUE.`), otherwise the tracer field would be overwritten by subsequent tracer initialisation routines.

B2) MMD2WAY_CHILD needs to allocate the memory itself, as no other submodel provides the memory for the *exchange field*. This field is calculated from an *in-field* provided by the parent model and supplied to other MESSy submodels (e.g. emission fields could be down-scaled from the coarse grid instead of being directly read in by IMPORT_GRID).
If the *representation* is named in the MMD2WAY_CHILD namelist (`&CPL_CHILD_ECHAM` or `&CPL_CHILD_COSMO`) and stored in the structure component `CplData(ii)%C_REPR`, this is an easy task. The *representation* ID `repr_input` and the corresponding rank are inquired calling the CHANNEL subroutine `get_representation_info`. Knowing the *representation* ID, the new *channel object* can be defined calling the CHANNEL subroutine `new_channel_object`.
But the *representation* is not always a priori known. This is indicated in the MMD2WAY_CHILD namelist by setting the *representation* string to `'#UNKNOWN'`. A classical example are emission fields provided as multi-level emissions. They are in the `Nx2D`-format (see Kerkweg et al., 2006), i.e., N levels attributed to different emission heights containing each 2D emission information. If the number

[20] Detailed information about the TRACER submodel are provided by Jöckel et al. (2008).

of levels is not a priori known by the child model, the parent model provides additional information about the *representation*, when the *representation* string (`CplData(ii)%C_REPR`) is set to `'#UNKNOWN'`. MMD2WAY_CHILD acquires this information by calling the MMD library subroutine `MMD_C_Get_Repr` which provides the *representation* name (`par_repr`), the *axis string* (`par_axis`), the global *dimensions* (`par_gdimlen`) and the height *attribute* (`par_att`) of the *exchange field* in the parent model. Based on this, MMD2WAY_CHILD determines its own *representation*:

i) If the *representation* name is one of `'GP_3D_MID'`, `'GP_3D_INT'` or `'GP_2D_HORIZONTAL'` using the same *representation* names in the child model automatically leads to the correct result, as these are standard *representation*s.

ii) In case of the *representation* `'GP_3D_1LEV'` the *representation* is converted to `'GP_2D_HORIZONTAL'`.

iii) In all other cases MMD2WAY_CHILD has to define a new *representation*:

For the definition of a new *representation* the *dimensions* need to be defined first: This is done by looping over the 4 CHARACTERS of the *axis string* `par_axis`. Simultaneously,

- the `rank` of the array,
- the local dimension length `dim_len` and
- the *axis string* `dim_axis`

of the child array are determined. For instance, if the `il`'s component of `par_axis` is `'X'`, the `rank` is increased by one, `dim_ids(il)` is set to `DIMID_LON`, which is the *dimension* ID for the longitude of the child model and `dim_len(il)` is set to `ie`, which is the local dimension length for the COSMO arrays.

· The horizontal dimensions of the fields are different between the parent and the child, but are implicitly given by the definition of the COSMO grid. Thus for the `'X'` and `'Y'` dimension the sizes are known and the COSMO definitions can be used.

· This is different for the `'Z'` and `'N'` dimensions, which can basically adopt every arbitrary value, but these dimensions have to be the same in the parent and the child. Thus new dimensions are defined for the `'Z'` and `'N'` dimensions, using the dimensions provided by the server model (`par_gdimlen`). The newly defined dimensions are named in a generic way:

(a) For the vertical dimension they start with `'DIM_'` followed by a string containing the number of z-levels, and ending with `'LEV'`. For instance, when the number of z-levels is 5 this yields the name `'DIM_5LEV'`. Before actually defining the dimension, it is tested if this dimension exists already. In this case, the existing *dimension* ID is taken. This test prevents repeated definition of the same *dimension*.

(b) For the number (`'N'`) dimension the same procedure takes place, only the name of the dimension variable ends with `'N'` instead of `'LEV'`.

After the loop over the *axis string*, the `rank`, the local *axis string* and the local dimension lengths have been determined. This allows for the definition of the *representation*:

· If `rank` is 2 and the `'Z'` and `'N'` dimension lengths are zero, the *representation* is equal to the standard *representation* `GP_2D_HORIZONTAL`.

- If the rank is larger than 2 and smaller or equal to 4, a new *representation* needs to be defined,
- in all other cases the *representation* cannot be properly evaluated and the simulation is terminated with the error message `'CANNOT IDENTIFY REPRESENTATION'`.

The ECHAM5/MESSy model uses another order of *dimensions* as the COSMO/MESSy model. Therefore the *axis string* characters, the *dimension* IDs and the *dimension* lengths must be permutated, if ECHAM5/MESSy is server. For instance, `dim_axis = 'XZNY'` becomes `dim_axis = 'XYNZ'` and the `dim_len` has to be changed accordingly.

The new *representation*s are constructed by the subroutine `make_cosmo_representation`. Input to this subroutine are

- the lengths of the `'Z'` and `'N'` dimensions (these are zero if the corresponding *dimension* is not required),
- the *dimension* IDs (`dim_ids`),
- the *axis string* (`dim_axis`), and
- the dimension lengths (`dim_len`).

Output of the subroutine is the *representation* ID of the newly created *representation*. Based on the incoming parameters, the respective *representation* is created. The names of the *representation*s are as generic as the *dimension* names. If `'Z'` and `'N'` dimensions are required, the *representation* is named `'REPR_4D_zzLEV_nnN'` where `'zz'` stands for the number of z-levels and `'nn'` for the number of n-levels. The *representation* names for `'Z'`-dimension only or `'N'`-dimension only are `'REPR_3D_zzLEV'` or `'REPR_3D_nnN'`, respectively. Using the CHANNEL subroutine `get_representation_info`, it is inquired, if the *representation* exists already. In this case the return variable `reprid` is set to the ID of the matching *representation*, otherwise, the *representation* needs to be created via the channel subroutine `new_representation`. In both cases the *representation* ID is handed back to the calling subroutine. After the *representation* is identified or newly created, the new *channel object* of the `'mmd2way_child'` *channel* can be created as described above for the case when the *representation* is known a priori. Additionally, a new *attribute* to the *channel object* will be set, if it was sent from the parent model(`par_att`). This is required in case of Nx2D emission fields, as the layer heights have to be known in addition to the amount given by the field itself.

2. **Determine** `CplData(ii)%cosmo_bd`**:**
   As for `CplData(ii)%cosmo`, this part is skipped, if the child model *channel* name is `'#XXX'`. If `L_BOUND` is `.TRUE.`, boundary data for the specific *coupling field* is required. In this case `CplData(ii)%cosmo_bd` is allocated to 2, as the boundary fields always consist of two time levels. In the standard COSMO model, these contain the fields at the beginning and the end of the time interval for which the boundary data is valid. During this interval the boundary data is linearly interpolated according to the elapsed time. In the on-line coupled setup the two time levels for the boundary data are filled with the same values. Otherwise the parent model needs to be run ahead by one boundary data time interval, which renders a 2-way nesting impossible. As the on-line coupling enables and requires much higher coupling frequencies, the error of this procedure is small.

   Although the levels are filled with the same values and the linear interpolation in time is not required anymore, the procedure is kept in order to leave untouched as much code as possible of the COSMO model.

For a *boundary field*, the 4D-POINTER to the full boundary data field is acquired with the subroutine `get_channel_object`. Afterwards, the POINTERs of the *POINTER ARRAY* `CplData(ii)%cosmo_bd(1:2)%ptr` are set dependent on the rank of the data field. For instance,

```
IF ( (CplData(ii)%rank == 3  .AND.  &
 (TRIM(CplData(ii)%CHILD%CHA) /= 'tracer_gp')) THEN
   CALL get_channel_object(status              &
        , TRIM(CplData(ii)%CHILD%CHA)        &
        , TRIM(CplData(ii)%CHILD%OBJ)//'_BD' &
        , p4=bdptr                            )

...

       CplData(ii)%cosmo_bd(1)%ptr =>bdptr (:,:,:,1:1)
       CplData(ii)%cosmo_bd(2)%ptr =>bdptr (:,:,:,2:2)
       NULLIFY(bdptr)
```

For a `rank=3` field the *boundary field* has 4 dimensions. Thus, the first boundary POINTER is set to the first boundary time level and the second to the second one. Note: this procedure does not work for 4D data fields, for which *boundary fields* would be 5-dimensional. However, a coupling to the individual boundary time levels would still be possible (and easily implementable), if required. So far such fields are not part of the model system apart from tracers.

Tracers are again processed differently. The two time levels of the boundary data are *channel objects* of the two TRACER CHANNELS `tracer_gp_x001` and `tracer_gp_x002`. Thus, the POINTERs to the boundary data can be associated directly by

```
CALL get_channel_object(status             &
   , TRIM(CplData(ii)%CHILD%CHA)//'_x001' &
   , TRIM(CplData(ii)%CHILD%OBJ)          &
   , p4=CplData(ii)%cosmo_bd(1)%ptr        )
```

and

```
CALL get_channel_object(status             &
   , TRIM(CplData(ii)%CHILD%CHA)//'_x002' &
   , TRIM(CplData(ii)%CHILD%OBJ)          &
   , p4=CplData(ii)%cosmo_bd(2)%ptr        )
```

3. **Determine CplData(ii)%lvartab, CplData(ii)%vartab_name and CplData(ii)%vartab_idx:**
   Some data manipulations require a distinction between *INT2COSMO inherent field*s and *additional field*s. All *INT2COSMO inherent fields* are listed in the variable table structure (`var_lm`) in INT2COSMO. This table determines -among other things- the *intermediate* and the *in-fields*, as well as the interpolation method. `CplData(ii)%lvartab`, `CplData(ii)%vartab_name` and `CplData(ii)%vartab_idx` are set in a loop over the variable table of INT2COSMO:
   - The structure component `CplData(ii)%lvartab` is `.TRUE.`, if the variable is element of the INT2COSMO variable table.

- Additionally, the name of the field in the variable table is stored in the structure component `CplData(ii)%vartab_name`. This is useful especially for one variable, the roughness length, as only for this the names (and the meaning) of the variables are different in INT2COSMO and in COSMO. In INT2COSMO the roughness length is called `'Z0'` whereas the COSMO model treats the product of roughness length times gravitational acceleration named `'gZ0'`. These two need to be associated with each other.
- Finally, the location, i.e., the index, of the field in the INT2COSMO variable table is stored in the structure component `CplData(ii)%vartab_idx`.

4./5. **Determine `CplData(ii)%ptr_in` and `CplData(ii)%ptr_i2c`:**
The information, if a *coupling field* is part of INT2COSMO, is required for the association of the POINTERs `CplData(ii)%ptr_in` and `CplData(ii)%ptr_i2c`. If the field is part of the variable table, the memory for the *in-field* and the *intermediate field* have been already allocated in INT2COSMO. Otherwise the memory for these fields is allocated in MMD2WAY_CHILD itself.

a) The memory for the *intermediate* and *in-fields* exists already:
   * The POINTER `CplData(ii)%ptr_in` can be directly associated calling the subroutine `get_channel_object`. For this call the object name is constructed by adding the suffix `_IN` to the *target field* name and the name of the *channel* is `'MMDC4_IN'`[21]. When no object is found the simulation will be terminated.
   * After the POINTER `CplData(ii)%ptr_in` is associated, the *representation* ID of this object is obtained by calling `get_channel_object_info`.
   * This ID is used to acquire the *axis string* (`CplData(ii)%AXIS`) and the local dimensions (`CplData(ii)%ldimlen`). This information is required by the MMD library for the data exchange.
   * Afterwards, `CplData(ii)%rank` is determined dependent on the third dimension of `ptr_in` to be `2` or `3`.
   * Additionally, the INT2COSMO field is marked as read (`var_in(itab)%lreadin = .TRUE.`).
   * The POINTER `CplData(ii)%ptr_i2c` to the *intermediate field* required in INT2COSMO is set by the subroutine `get_channel_object` by using `CplData(ii)%vartab_name` as *channel object* name and `'MMDC4'` as *channel* name.

b) The memory for the *intermediate* and *in-fields* needs to be allocated (*additional fields* only):
   The *representation* of the *intermediate* and *in-fields* is not a priori known. They are determined depending on the `rank` of the field.
   * For `CplData(ii)%rank = 2` the *in-field* and the *intermediate field* are defined using the existing *representation IDs* `REPR_I2C_2D_IN` and `REPR_I2C_2D`, for a 2D *in-field* and a 2D *intermediate field*, respectively.
   * If `CplData(ii)%rank` is `3` and `CplData(ii)%C_REPR` is `'GP_3D_MID'`, the prior defined *representation IDs* `'REPR_3D_MID_IN'` and `'REPR_I2C_3D_MID'` are used.
   * In all other cases, the subroutine `make_i2c_representation` is called with the vertical and number dimensions as input parameters. The subroutine determines

[21] One special case has to be considered: the *in-field* for `'W_SO'` is not necessarily named `'W_SO_IN'`, it can also be `'W_SO_REL_IN'`.

similarly to the subroutine `make_cosmo_representation` the *representation*s for the *intermediate field* and the *in-field*.

Using these *representation*s, the new *channel objects* for the *in-field* and the *intermediate field* are defined. The *channel objects* for the *in-fields* are added to the INT2COSMO *channel* `'MMDC4_IN'`. The *intermediate fields* are added to the *channel* `'MMDC4'` containing all *intermediate fields*.
Additionally, the *axis string* (`CplData(ii)%AXIS`) and the local *dimensions* (`CplData(ii)%ldimlen`) are determined by calling the subroutine `get_representation_info` with the *representation ID* `REPR_IN`.

- **Define_data_arrays:**
  After all data fields are associated or allocated, the respective POINTERs of the *in-fields* can be forwarded to the MMD library routines. To address the correct *exchange fields* within the MMD library, a loop over the *exchange fields* is performed by using the MMD library function `MMD_C_GetNextArray`. For each field the MMD library subroutine `MMD_C_Set_DataArray` is called, handing over the POINTER to the memory allocated for the *in-field*. Additionally, the *axis string* and the local *dimension* length are communicated to the MMD library, which, internally uses this information, later on, to unpack the data received from the parent model.

- **MMD_C_SetInd_and_AllocMem**
  The initialisation is finalised by calling the MMD library subroutine `MMD_C_SetInd_and_AllocMem`. It invokes the MMD internal calculation of the required buffer sizes and the actual memory allocation via `MPI_alloc_mem`.

  With this the initialisation phase for the MMD2WAY_CHILD submodel is complete.

In case of a new start of a simulation (`lstart = .TRUE.`) the actual data exchange and interpolation is performed at the end of `mmd2way_init_memory` instead of `mmd2way_init_loop` during the time integration. This is necessary, as the initialisation of the fields in the COSMO model needs to take place prior to the time loop.

5.1.2 Integration Phase

The procedure explained in this section is part of the subroutine `mmd2way_init_loop`, as the update of the child model fields is required at the very beginning of the respective time step. The very first time step of a model simulation (`lstart = .TRUE.`) builds an exception to this rule, because in the very first initialisation not only *input* and *boundary fields*, but also the *initial fields* are calculated. The latter are already used during the end of the initialisation phase in the COSMO model. Therefore the very first data transfer and interpolation takes already place in `mmd2way_child_init_memory`. Nevertheless, the procedure explained here is the same.

There are two chunks of information, which need to be exchanged during the integration phase of a 1-way coupled simulation: the coarse grid data and the TIMER status of the parent model. For the 2-way coupling, in addition, the data required by the parent is (after processing) sent to the parent.

5.1.2.1 Data exchange, interpolation and supply

First of all, the coupling *event* (`CPL_EVENT`) determines, if data exchange should occur in this time step. If this is the case, the MMD2WAY_CHILD private subroutine `exchange_interpol_data` is called:

1. First, the data exchange is performed by calling the MMD library subroutine `MMD_C_GetBuffer`, which fills all *in-fields* with the updated values sent by the parent model. The subroutine `MMD_C_GetBuffer` also provides a measure for the time required for coupling. It hands back a variable containing the time in seconds, which the child model had to wait until the parent model data was accessible. This information is written to the log-file.

2. As soon as the *in-fields* are filled, the MMD internal check of the consistency of the exchanged horizontal data field is performed. This is invoked by calling the subroutine `MMD_testC_compare`. As the content of the `test_array` does not change with time this check is only performed at the beginning (start or *restart*) of a simulation.

3. The COSMO and the INT2COSMO implementation of the MPI data exchange routines for scattering and gathering fields include dimension checks, which inhibit the exchange of data of different horizontal dimensions. But the standard 2D- and 3D-COSMO fields and the *in-fields* of INT2COSMO are of different horizontal resolution. This was no matter as long as COSMO and INT2LM were independent programs. In the case of on-line coupling the easiest way to cope with this, was to (re-)set the variables used for the dimension checks every time, when changing between INT2COSMO and COSMO parallelisation. This is done within the subroutine `switch_par_utilities(flag)`. When `flag=1` the check environment switches to INT2COSMO parallelisation, in the case `flag=2` it is switched back to COSMO parallelisation. Because the subroutines called in the following in `exchange_interpol_data` are INT2COSMO routines (compare flow chart Fig. 9), the parallel environment checks are switched to INT2COSMO at this place. After those more general preparations the processing of the incoming data starts:

   - **mmd2way_prepare_external_data:** This subroutine collects all data required for the interpolation and the calculation of the *coupling fields*: INT2COSMO basically distinguishes three types of "external data":
     (a) the external parameters pre-defined by the EXTPAR software[22] on the target COSMO model grid,
     (b) the external parameters as provided by the driving instance (parent), and
     (c) the data fields provided by the driving instance (parent).

     External parameters are constant or slowly changing fields given as boundary conditions for the model domain, e.g., the soil type, the leaf area index, the root depth and so forth.

     The external parameters for the target grid are read in from an extra file in the INT2COSMO subroutine `external_data`. Based on the read-in values, the variables required in COSMO and INT2COSMO are calculated later on. At the time being, in INT2LM all external parameters are read, meaning, even for monthly changing variables (in the climate mode of the model) the data for all twelve month is read at once. Thus reading the external parameters is only required at the beginning and the *restart* of a simulation. For the sake of higher computational efficiency, the read procedure is switched off for additional time steps in MMD2WAY_CHILD_INT2COSMO. This is accomplished by the extra LOGICAL variable `lread`, which is parameter to the subroutine `external_data` and switches off the reading procedure[23].

     In the subroutine `external_data`, the reading of the external parameters of the coarse grid is always omitted, because these variables are updated by the on-line data exchange.

[22] `http://www2.cosmo-model.org/content/model/modules/Extpar_201408_user_and_implementation_manual.pdf`

[23] In future it might be desirable to regularly read updated external parameters. For this we implemented the *event* `READEXT_EVENT`. Per default the adjustment of the *event* is set to `'none'`, deactivating the event, i.e., the external data are only read at start or restart of a simulation.

The calculation of the external parameters following the read procedure is kept virtually unchanged and the fields are processed in the same way as in INT2LM. For more details about the code changes see Sect. 7.8.

Similarly as in `external_data`, in the subroutine `org_read_coarse_grid` the reading procedure is omitted, as the fields are already initialised by the on-line coupling.The subsequent analysis of the data, the determination of LOGICAL switches and intermediate fields is kept unchanged except for a few very small changes, which are discussed in detail in Sect. 7.18.

- **mmd2way_child_interpolation:** After the preparation of the data, the interpolation begins. The interpolation of the *INT2COSMO inherent fields* proceeds exactly as in the off-line INT2LM:
  (a) The fields are interpolated horizontally by calling the INT2COSMO routine `org_coarse_interpol`. In this subroutine a field specific interpolation is performed. For 3D fields each vertical level is independently interpolated horizontally. In addition to the original implementation, the conservative remapping of the MESSy submodel GRID can be used.
  (b) The horizontal interpolation is followed by the vertical interpolation. If the COSMO/MESSy model is the server (`llm2lm = .TRUE.`) the subroutine `org_vert_inter_lm` is called, otherwise the subroutine `org_vert_interpol`. As a third option, NREGRID as provided by the MESSy submodel NREGRID can also be chosen for the INT2COSMO *inherent fields* by setting the interpolation flag to `'W'`.
  (c) Subsequently, additional 2D-fields are calculated by the INT2COSMO subroutine `org_2d_fields`.
  (d) If the parent is not a COSMO/MESSy model instance, the fields need adjustment to the non-hydrostatic grid of the COSMO model. This is done within the subroutine `org_lm_fields`.

  It is possible to request the original INT2LM output in the `&CTRL` namelist in namelist file `mmd2way.nml`. If the LOGICAL switch `l_I2Cori_output` is set `.TRUE.`, the original INT2LM output routine `org_lm_output` is called.
- **Interpol_AddiArrays:** The interpolation of the *additional fields* is based on the routines provided by INT2LM. In the subroutine a loop over all *exchange fields* (from 1 to `NEXCH`) is processed. All fields with `CplData(ii)%lvartab = .TRUE.` are skipped because they have already been interpolated in INT2COSMO. Additionally, the MMD `test_array` is excluded as the test is performed for the *in-field*. Two different interpolation procedures can be selected from MMD version 2.0 on. On the one hand side, the additional fields can be interpolated via conservative remapping using the SCRIP implementation in the submodel GRID. This is requested by setting the respective flag in the namelist to `'C'`. On the other hand, the int2lm standard interpolation routines can be called for horizontal interpolation: In a loop over the vertical levels (or vertical levels and number dimension length for 4D fields) the fields are interpolated horizontally by calling the subroutine `interpol_coarse_OneLayer`. This subroutine uses the weights calculated before in `org_coarse_interpol` for the *INT2COSMO inherent fields* and calls for each field the interpolation routines according to the `CplData(ii)%C_INTERPOL` flags set in the namelist (as `org_coarse_interpol` does for the *INT2COSMO inherent fields*).

  After interpolating the fields horizontally, they are vertically interpolated. Again, two procedures are available
  – Either the vertical interpolation proceeds in the subroutine `interpol_vert_AddiArray`. This is indicated by setting the respective flag to

`'V'`. If `'V'` is requested, again the INT2COSMO routines are used. If the parent is ECHAM5/MESSy, the interpolation routines `vert_interpol` and `vert_int_lm`, or, depending on the vertical grid `vert_z_lm` are subsequently called including the adaption to the COSMO non-hydrostatic grid. For `llm2lm = .TRUE.` (parent instance is a COSMO/MESSy model), the subroutine `vert_interp` performs the entire interpolation. The vertical interpolation of the 4D fields proceeds for each number dimension independently.

  - Or the vertical remapping proceeds using NREGRID (provided by the generic MESSy submodel GRID). This interpolation procedure is requested by setting the respective flag to `'W'`. In this case the subroutine `vert_ncinterpol` is called. This subroutine first defines the geo-hybrid grids as required for the pure vertical regridding via NREGRID. Second, the field, that should be remapped is converted to the 1D format as required by GRID. Finally, the fields is vertically remapped using the GRID_TRAFO subroutine `REGRID_CONTROL` and the resulting field is converted back to the usual 3D format.

- Often, the remapping of the soil properties from the parent does not lead to a good initialisation of the soil properties. To improve the soil initialisation, the subroutine `force_vars_from_file` allows to overwrite the soil properties from another simulation. In this case the model domain / grid definition of the model and in the file must be identical.
- After finishing the INT2COSMO routines, the dimension check is reset to the COSMO parallelisation in `switch_par_utilities`.

4. Last but not least, the *intermediate fields* calculated by the interpolation routines (i.e., `CplData(ii)%ptr_i2c`) need to be assigned to the *target fields*.

   - The subroutine `move_initial_arrays_to_COSMO` moves the initial data, only. For the assignment of the COSMO/MESSy fields the `CplData` structure is processed in one loop (index `ii`) over the entries of `CplData`. The field is skipped,
     - if `L_INITIAL` is `.FALSE.`,
     - if it is only an *exchange field* (the child model *channel* name is `'#XXX'`), or
     - if it is the `test_array`.

     Otherwise, dependent on the dimension of the `CplData(ii)%cosmo`, the data is moved to the *target field(s)*:
     - If the dimension of `CplData(ii)%cosmo` is 1, only one field needs to be assigned. Within a loop over the fourth dimension (loop index `iX`) the first three ranks are copied:

```
size3 = SIZE(CplData(ii)%cosmo(1)%ptr,3)

CplData(ii)%cosmo(1)%ptr(istartpar:iendpar   &
     ,jstartpar:jendpar,1:size3,iX) =        &
      CplData(ii)%ptr_i2c(istartcos:iendcos  &
     ,jstartcos:jendcos,1:size3,iX)
```

       Note: the data can only be copied on the "core-regions" (see Sect. 5.1.4) as only these overlap on each PE of the COSMO and of the INT2COSMO grid. This is achieved by using the indices `istartpar, iendpar, jstartpar` and `jendpar` for the COSMO grid and the indices `istartcos, iendcos, jstartcos` and `jendcos` for the INT2COSMO grid. Details about the matching of the decomposition of the two grids are provided in Sect. 5.1.4. As only the "core-regions" of the local domain can be initialised for each

field, the COSMO subroutine `exchg_boundaries` must be called to ensure that the entire local domains are initialised.

– If the dimension of `CplData(ii)%cosmo` is larger than 1, the variable depends on time and three cases are distinguished:

∗ For a 2-time level integration scheme only the "new" time level needs to be initialised:

```
size3 = SIZE(CplData(ii)%cosmo(1)%ptr,3)

CplData(ii)%cosmo(nnew)%ptr(istartpar:iendpar &
     ,jstartpar:jendpar,1:size3,iX) =         &
      CplData(ii)%ptr_i2c(istartcos:iendcos   &
     ,jstartcos:jendcos,1:size3,iX)
```

with `nnew` being the index of the "new" time level in COSMO.

∗ For a 3-time level scheme two time levels are initialised:

```
size3 = SIZE(CplData(ii)%cosmo(1)%ptr,3)

CplData(ii)%cosmo(nnew)%ptr(istartpar:iendpar &
     ,jstartpar:jendpar,1:size3,iX) =         &
      CplData(ii)%ptr_i2c(istartcos:iendcos   &
     ,jstartcos:jendcos,1:size3,iX)

CplData(ii)%cosmo(nnow)%ptr(istartpar_c4:iendpar_c4 &
     ,jstartpar_c4:jendpar_c4,1:size3,iX) =         &
      CplData(ii)%ptr_i2c(istartcos:iendcos         &
     ,jstartcos:jendcos,1:size3,iX)
```

∗ A special case are the tracers, as they are not accessible via index shift: The POINTERs of the *POINTER ARRAY* `CplData(ii)%cosmo` are associated in a way that the first POINTER points to the `nnew` time level and the `nnow` time level is accessed by the second POINTER (see Sect. **??**). Thus for a 2-time level integration scheme only the first, otherwise both fields are initialised.

As in the diagnostic case, the copy statements are placed in a loop over the fourth dimension (`iX`) of `CplData(ii)%cosmo(nt)%ptr` and `exchg_boundaries` needs to be called to complete the initialisation.

- The subroutine `move_boundary_arrays_to_COSMO` copies the *boundary* and the *input fields* to their respective *target fields.*

  One important difference between the boundary data association of the off-line COSMO setup and the on-line coupled setup must be emphasised here: In the off-line COSMO model, boundary data are provided for two time steps (e.g., in 6 hour intervals) and the actual boundary values in-between these time steps are interpolated linearly between these two time steps. If this were to be imitated in the on-line coupling, the parent model had to be one coupling time interval ahead of the child model. This could be implemented for 1-way coupling, but is not applicable for on-line 2-way nesting. In this case, the parent model cannot be ahead of the child model. Therefore it was decided to fill the two time levels of the boundary data with the same actual value. As the on-line setup enables a much higher coupling frequency, i.e., every parent time step, the deviation due to the different handling of boundary data is expected to be small.

For the transfer of the *boundary* and the *input fields* a loop over the entries of `CplData` is established. Again, the `test_array` and the *exchange fields* only required for the interpolation are skipped. If boundary data is requested for a field (`CplData(ii)%L_BOUND = .TRUE.`) the respective *intermediate field* `CplData(ii)%ptr_i2c` is copied to the first time level of the boundary data *POINTER ARRAY*:

```
size3 = SIZE(CplData(ii)%cosmo_bd(1)%ptr,3)
size4 = SIZE(CplData(ii)%cosmo_bd(1)%ptr,4)
CplData(ii)%cosmo_bd(1)%ptr(istartpar:iendpar &
     ,jstartpar:jendpar,1:size3,1:size4) =    &
      CplData(ii)%ptr_i2c(istartcos:iendcos   &
     ,jstartcos:jendcos,1:size3,1:size4)
```

Subsequently, this field is distributed to the full local domains by calling `exchg_boundaries`. Finally, the distributed field is copied to the second boundary layer time slice.

Similarly, if `CplData(ii)%L_INPUT = .TRUE` for a *coupling field*, the *intermediate field* is copied to the `CplData(ii)%cosmo` *POINTER ARRAY*:

```
size3 = SIZE(CplData(ii)%cosmo(1)%ptr,3)
size4 = SIZE(CplData(ii)%cosmo(1)%ptr,4)
CplData(ii)%cosmo(1)%ptr(istartpar:iendpar   &
     ,jstartpar:jendpar,1:size3,1:size4) =   &
      CplData(ii)%ptr_i2c(istartcos:iendcos  &
     ,jstartcos:jendcos,1:size3,1:size4)
```

The transfer is finalised by the boundary exchange via `exchg_boundaries`.

Before copying the *intermediate fields* to the *target fields*, one additional measure has to be taken, since some 2D *INT2COSMO inherent fields* are flagged: The COSMO model tests for undefined values by comparing with a constant value `undefncdf` (defined as "undefined"). Usually in the COSMO off-line setup the initial and boundary data are read from files, in which undefined data points are marked by this special value. Therefore some fields are flagged during the on-line coupling for the sake of consistency:

- If the land/sea mask flag `var_lm(i)%lsm` of a variable is set to `'l'`, the variable will be flagged with `undefncdf` at points over sea.
- The grid points of `'T_SNOW'` will be set to `undefncdf` where `w_snow_lm < 0.`.
- `'Z0'` is multiplied with the gravitational acceleration (`g`) to get the variable `'gZ0'` as used by COSMO.

The procedure explained in detail above is summarized in Fig. 11. Furthermore, it illustrates the usage of the MMD2WAY_CHILD internal POINTERs: `ptr_in` is the *in-field*, which is input to INT2COSMO. During the horizontal interpolation the vertical and number dimensions remain untouched. The result of the interpolation is written to the *intermediate field* `ptr_i2c`. If the *in-field* is 3D in space (number of incoming vertical levels is `ke_in`) vertical interpolation is possible. After the vertical interpolation `ptr_i2c` contains valid data on the vertical levels `1:ke` with `ke` being the number of vertical levels in the child. After the interpolation, the *intermediate field* (`ptr_i2c`) is copied to the *target field(s)*, i.e., those variables used subsequently in the basemodel or other MESSy submodels. For *initial* and *input fields* the data is copied to the variable associated with the `cosmo(.)%ptr`, for *boundary fields* the *intermediate field* is copied to the boundary variable associated with the POINTERs `cosmo_bd(.)%ptr`.

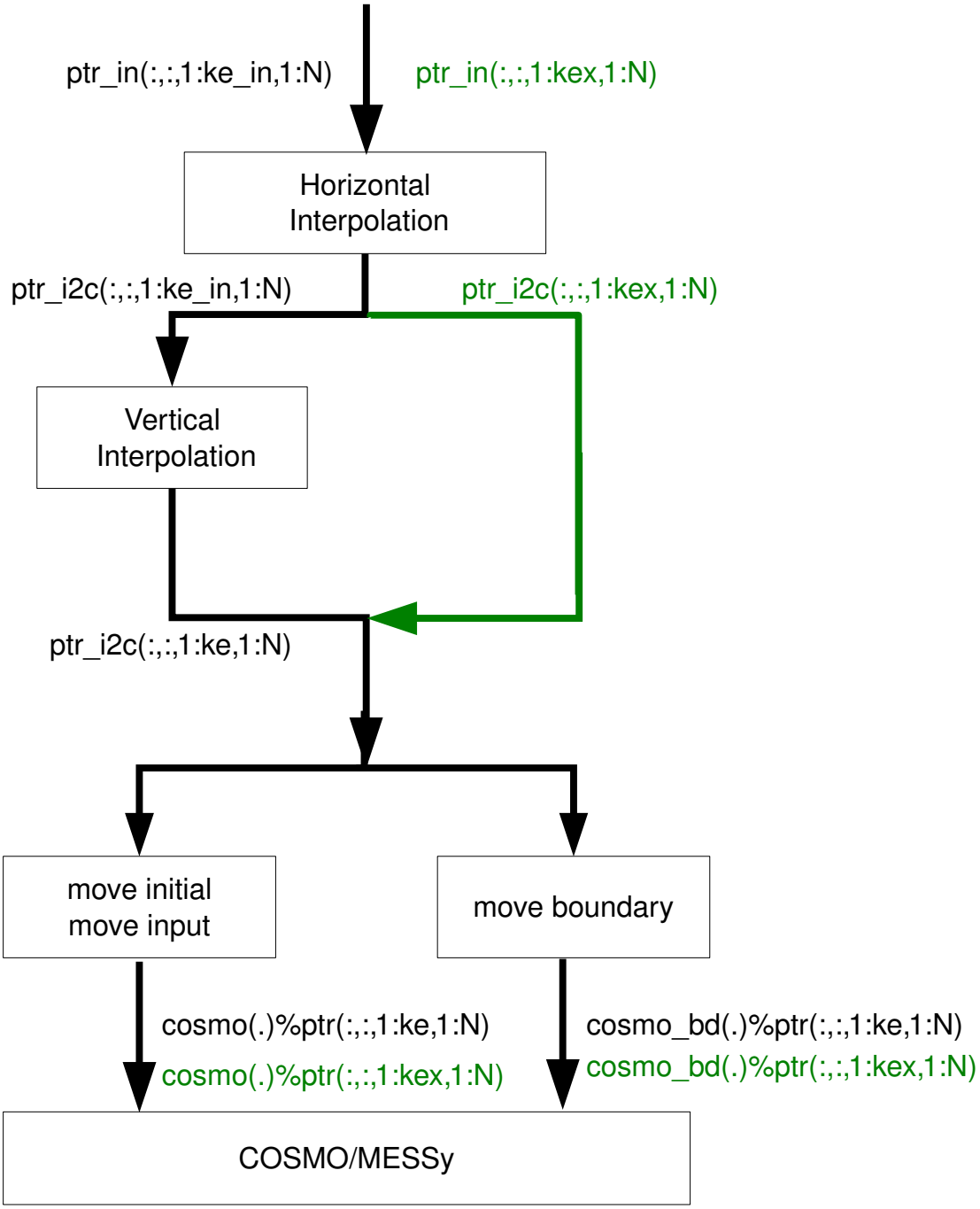

Figure 11: Pointer usage in MMD2WAY_CHILD. `N` is an arbitrary (number) dimension, `ke_in` is the number of vertical levels of the *in-field*, `ke` is the number of vertical levels in the COSMO model and `kex` is an arbitrary number of vertical levels. First, the *in-field* is interpolated horizontally, second, -if requested and possible- the vertical interpolation (black) is performed and third, the *intermediate field* is copied to the COSMO/MESSy target and boundary variables.

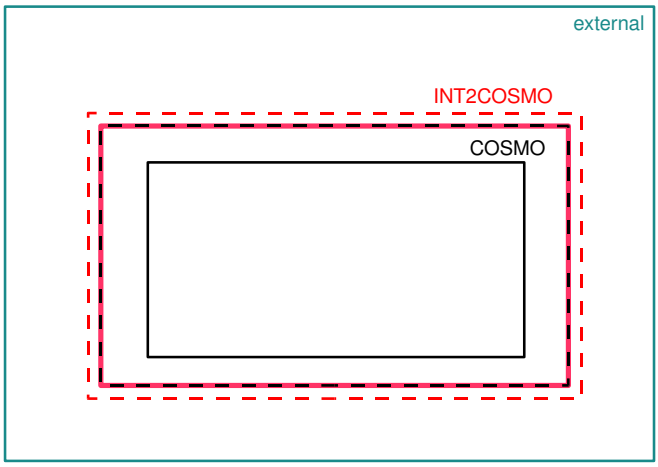

Figure 12: Illustration of the three model domains for the external parameters (turquoise), INT2COSMO (red) and the COSMO model (black). The dashed lines illustrate the entire model domains, whereas the solid lines show the "inner" domains (see text).

5.1.3 Finalisation Phase

After the integration phase at the end of a simulation, the memory allocated in the course of the simulation is deallocated. The MMD2WAY_CHILD subroutine `mmd2way_child_free_memory` releases the memory allocated by INT2COSMO by calling the INT2COSMO subroutine `org_cleanup`. For the MMD `test_array` and the other memory allocated by MMD the MMD library routines `MMD_testC_FreeMem` and `MMD_C_FreeMem` release the memory, respectively.

5.1.4 Grid definitions and parallel decomposition of INT2COSMO

In this section the definition of the different grids and domains used in INT2COSMO and the parallel domain decomposition of INT2COSMO, which differs from the INT2LM decomposition, are illustrated.

5.1.4.1 Domains

INT2COSMO works with data defined on four different domains:

1. The domain of the fields provided by the parent model, i.e., the *in-grid* or the *in-field* domain;
2. the target domain, i.e., the COSMO model domain;
3. the INT2COSMO domain, i.e., the "working" domain of INT2COSMO;
4. the domain on which the external parameters such as root depth, the orography, the leaf area index or the land-sea mask, are defined.

Table 3 lists which structure components of the MMD2WAY_CHILD variable CplData are defined on which domain.

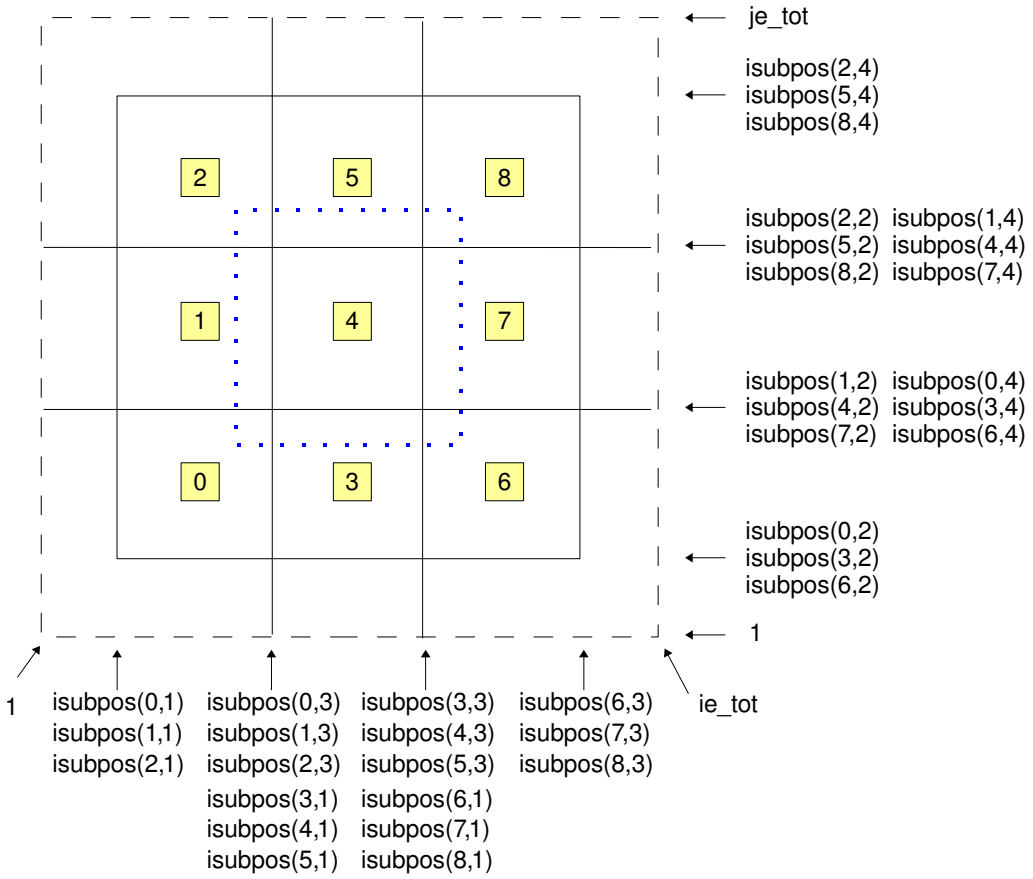

Figure 13: Example for a parallel decomposed COSMO model domain, distributed on 3x3 PEs: The numbers on yellow background are the respective PE numbers. The black dashed line is the border of the entire model domain, the solid lines depict the core-regions of the local domains. The blue dotted line indicates the full local domain of PE 4, i.e., including the halo or ghost boundaries. Additionally annotated are the global indices for of the core-regions.

Table 3: List of CplData structure components and their respective model domain.

structure component	domain
CplData(ii)%ptr_in	*in-field* domain
CplData(ii)%ptr_i2c	INT2COSMO domain
CplData(ii)%cosmo(:)%ptr	COSMO domain
CplData(ii)%cosmo_bd(:)%ptr	COSMO domain

The *in-field* domain is defined by the parent (see Sect. 5.2.1.3) and therefore independent of the target COSMO model domain.

The other three domains, must have the same grid spacing and need to be rotated in the same way. In other words, the only difference between these model domains is their size. The size of the domains is obvious:

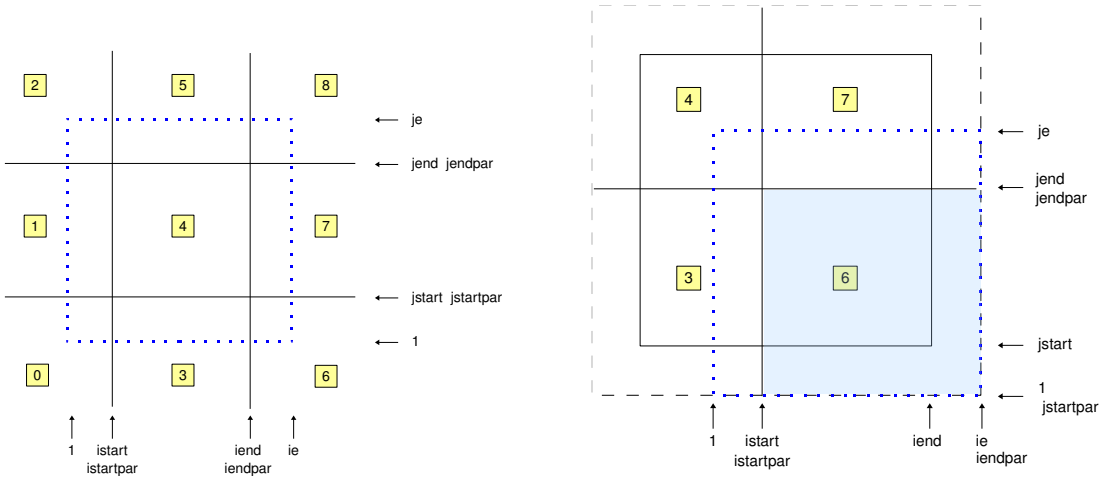

Figure 14: The figure is zooming in on PE 4 (left) and PE 6 (right) of Fig. 13, illustrating the definition of the local indices `istart`, `iend`, `jstart`, `jend`, `istartpar`, `iendpar`, `jstartpar` and `jendpar`.

external parameter domain > INT2COSMO domain > COSMO domain

This hierarchy is caused by the processing order:

- INT2COSMO processes external parameters. Consequently, the external parameter fields need to be larger than the INT2COSMO fields.
- INT2COSMO interpolates data to the COSMO domain, thus the INT2COSMO domain has to be larger than the COSMO domain.

The model domains are illustrated in Fig. 12.

5.1.4.2 The parallel decomposition of the COSMO model grid and the INT2LM grid

INT2LM and the COSMO model use the same decomposition procedure. The domain is split into rectangular parts according to the numbers of PEs in x and y directions (`nprocx` and `nprocy` as namelist entries). Figure 13 shows a decomposition for 9 PEs (3x3). The parts enclosed by the solid lines illustrate the so-called "core-regions" of the local (i.e., PE-bound) model domains. The sum of the core-regions covers the entire model domain, except for a frame at the lateral boundaries, which relevance becomes clear below (i.e., entire domain = "inner" domain + lateral frame).

These core-regions are unambiguously defined by their indices in the entire model domain. The indices are stored in the INTEGER ARRAY variable `isubpos(0:nPE-1,4)`, with `nPE` number of PEs. This variable lists the indices of the lower left and the upper right corner of the core-regions of each PE in the entire model domain:

- `isubpos(:,1)`: first (`'i'`) index of the lower left corner;
- `isubpos(:,2)`: second (`'j'`) index of the lower left corner;
- `isubpos(:,3)`: first (`'i'`) index of the upper right corner;

- `isubpos(:,4)`: second (`'j'`) index of the upper right corner.

Figure 13 illustrates the definition of the individual `isubpos` for the example.

To allow the computationally efficient implementation of processes, which require the data of the neighbour grid points, a "halo" or "ghost boundaries" are defined surrounding the core-region of each PE. Thus, the local decomposed domain on each PE consists of the ghost boundaries and the core-region[24]. The blue dotted line in Fig. 14 illustrates the local model domains of PE 4 (left) and PE 6 (right), including the ghost boundaries and the core-region. Physical processes are calculated on the core-region, whereas the ghost boundaries are used, if the neighbouring value is required during the calculation of a process, e.g., for advection. Before calculating such a process, it must be ensured, that the ghost boundaries contain the correct values for the required fields. This is achieved by the subroutine `exchg_boundaries`, which transfers the data from the PE, on which a grid point belongs to the core-region, to the ghost boundary of the neighbouring PEs.

Figure 14 illustrates the definitions of the local grid and the respective indices for example PEs 4 and 6:

- The lower left corner of the local domain is always the grid point `(1,1)` and the upper right corner is `(ie,je)` with `ie` and `je` being the number of grid points in x or y direction, respectively.
- The lower left corner of the core-region is `(istart,jstart)` and the upper right corner is `(iend,jend)`.

If the PE is not completely surrounded by other PEs, the PE also hosts a part of the lateral frame of the entire model domain. In this case, the physical processes need also to be calculated on the lateral frame. The lateral frame is not a part of a core-region, but it is identical to the ghost boundaries of the respective PE. Thus, the physical processes are also calculated on the ghost boundaries which is visualised by the right hand side of Fig. 14. PE 6 is located at the lower right corner of the entire model domain. Thus, to calculate the processes on the entire model domain, on the eastern and southern border the physical processes are also calculated on the ghost boundaries. While `istart, jstart, iend` and `jend` always refer to the core-region, the indices `istartpar, jstartpar, iendpar` and `jendpar` refer to the grid points on which the physical processes are really calculated. On a PE completely surrounded by other PEs, these index quadruples are equal, as illustrated by the left hand side of Fig. 14. The right hand side of Fig. 14 shows the indices as defined for a PE at the lateral boundary. The solid black rectangle illustrates the core-region, the blue dotted line the entire local model domain. The light blue area indicates the domain part, on which the physical processes are calculated for PE 6.

The width of the ghost boundaries, i.e., the number of grid points (variable `nboundlines`) added at each side to the core-region, depends on the processes taken into account in the simulation. For the COSMO model using the leap-frog time integration scheme `nboundlines`=2 is sufficient. If the Runge-Kutta time integration scheme is used, `nboundlines` is 3 or 4 depending on the order of the Runge-Kutta scheme. In contrast to this, `nboundlines` for the INT2COSMO domain is determined by the order of the interpolation routines. For the currently implemented interpolation algorithms `nboundlines` is always 1.

[24]Note: The width of the ghost boundaries and of the lateral frame are identical. It is defined by the namelist variable `nboundlines`.

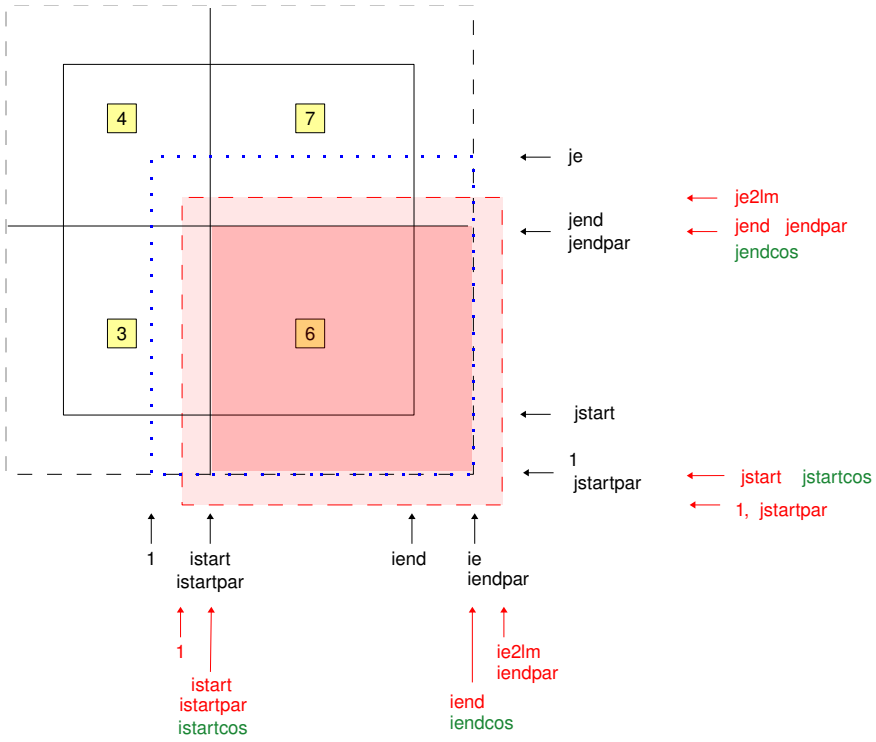

Figure 15: This is an extension of the right hand side of Fig. 14. Additionally the overlapping INT2COSMO regions for PE 6 are indicated by red rectangles. The lighter red shows the ghost latitudes, whereas the darker red indicates the INT2COSMO core-region. The red indices are the indices for the local INT2COSMO domain. Additionally, the position of the four indices `istartcos`, `iendcos`, `jstartcos` and `jendcos` are marked (green colour).

5.1.4.3 The parallel decomposition of the INT2COSMO grid

For the sake of computational efficiency, the local INT2COSMO and the local COSMO model domains should be congruent. If this is not the case, it is required to gather the fields calculated by INT2COSMO on one PE and to scatter them to the local COSMO model domains, afterwards.

Unfortunately, the decomposition algorithm used for COSMO and INT2LM does not lead to a congruent decomposition of both domains, because in the decomposition algorithm

1. an `nboundlines`-wide frame is subtracted from the entire domain,
2. the remaining domain (called "inner" domain) is almost equally distributed on the PEs, which leads to the definition of the core-regions, and
3. the `nboundlines`-wide ghost-boundaries are added to the core-regions to define the entire local model domains.

The INT2COSMO domain is larger than the COSMO model domain, and the number of ghost boundaries is larger for the COSMO model than for INT2COSMO, therefore "inner" domains have always different sizes. This is why the decomposition algorithm cannot result in congruent domains for COSMO and INT2COSMO.

Nevertheless, it is possible, that the COSMO core-regions are always covered by the INT2COSMO core-regions. The coverage of the COSMO core-regions is sufficient to avoid the gather and scatter procedure for the complete fields. Nevertheless, it requires additional data exchange, for the ghost boundaries. This is performed by the COSMO subroutine `exchg_boundaries`. The coverage of the COSMO core-regions by the INT2COSMO core-regions can be achieved by skilfully defining the INT2COSMO core-regions based on the COSMO core-region definition. The procedure is outlined below: The INT2COSMO domain is larger than the COSMO model domain, i.e., the entire COSMO model domain is equal to the "inner" INT2COSMO domain, i.e., the entire domain minus an `nboundlines` wide frame (compare Fig. 12). Therefore, the INT2COSMO core-regions can be made equal to the COSMO core-regions for all PEs, except for those at the lateral boundaries. Here the INT2COSMO core-regions are larger than the COSMO core-regions. To be more precise, (following the calculation outlined below) the INT2COSMO core-regions at the lateral boundaries are equal to the parts of the local COSMO model domain described by the indices `istartpar`, `jstartpar`, `iendpar` and `jendpar`.

Coverage can only be achieved, if the core-regions of the decomposed COSMO model are taken as base for the calculation of the decomposed INT2COSMO grid. Thus, the `isubpos` definition of the COSMO model domain (further on denoted as `isubpos_cosmo`) is taken and the INT2COSMO variable `isubpos` is calculated from this variable. For the PEs not located at the lateral boundaries of the model domain `isubpos` of the INT2COSMO domain is equal to `isubpos_cosmo+nboundlines`, with `nboundlines=1` for INT2COSMO. `nboundlines` must be added, as the global indices are shifted by `nboundlines` in the INT2COSMO grid.

In addition to this shift by `nboundlines`, at the lateral boundaries it has to be taken into account, that the "inner" INT2COSMO domain equals the entire COSMO model domain. Thus, as the `isubpos_cosmo` are the indices of the "inner" COSMO domain, these need to be shifted by `nboundlines_cosmo` in order to equal the indices of the entire model domain. Thus `isubpos` for PEs at the lateral boundaries of the INT2COSMO domain is calculated by:

$$\texttt{isubpos}(.,1) = \texttt{isubpos}_{\mathrm{COSMO}} - \texttt{nboundlines}_{\mathrm{COSMO}} + 1$$

at a western lateral boundary,

$$\texttt{isubpos}(.,2) = \texttt{isubpos}_{\mathrm{COSMO}} - \texttt{nboundlines}_{\mathrm{COSMO}} + 1$$

for a southern lateral boundary,

$$\texttt{isubpos}(.,3) = \texttt{isubpos}_{\mathrm{COSMO}} + \texttt{nboundlines}_{\mathrm{COSMO}} + 1$$

at a eastern lateral boundary and

$$\texttt{isubpos}(.,4) = \texttt{isubpos}_{\mathrm{COSMO}} + \texttt{nboundlines}_{\mathrm{COSMO}} + 1$$

at a northern lateral boundary.

Because of this procedure, the core-regions of the INT2COSMO domain and those of the COSMO model domain are not equal at the lateral boundaries. In order to copy the data from the fields defined on the INT2COSMO domains correctly to the fields defined on the COSMO model domains, additional indices are required to indicate the co-location of the COSMO local domains on which the physical processes are calculated (i.e., those indicated by `istartpar, jstartpar, iendpar` and `jendpar`) and the INT2COSMO core-regions. The indices `istartcos`, `iendcos`, `jstartcos` and `jendcos` are defined in a similar manner as `istartpar, jstartpar, iendpar` and `jendpar` by:

$$
\begin{aligned}
\texttt{istartcos} &= 1 + \texttt{nboundlines} \\
\texttt{iendcos} &= \texttt{ie2lm} - \texttt{nboundlines} \\
\texttt{jstartcos} &= 1 + \texttt{nboundlines} \\
\texttt{jendcos} &= \texttt{je2lm} - \texttt{nboundlines}
\end{aligned}
$$

with `ie2lm` and `je2lm` being the dimension of the local INT2COSMO domains (similar to `ie` and `je` for the decomposed COSMO domain). This is illustrated by Fig. 15.

5.2 The parent instance

For the 1-way coupling, the parent instance fulfills three tasks, which are essential for the on-line coupling:

- It dictates the date/time and *restart* settings of the child model.
- It calculates the `index_list`, i.e., the association of the child and the parent instance grid points on the individual PEs.
- It provides the *exchange fields*.

Comparably to MMD2WAY_CHILD, in MMD2WAY_PARENT the data for the data exchange is organised in a Fortran95 structure:

```
! CplData STRUCTURE
TYPE T_COUPLE_C_DATA
   ! CHANNEL AND CHANNEL OBJECT NAMES IN PARENT MODEL
   TYPE(t_chaobj_cpl)                    :: name
   ! POINTER TO PARENT FIELD
   REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr => NULL()
   ! REPRESENTATION OF PARENT FIELD
   INTEGER                               :: rank
   INTEGER, DIMENSION(4)                 :: ldimlen=0
   ! STRING ORDER OF AXES
   CHARACTER(LEN=4)                      :: AXIS
END TYPE T_COUPLE_C_DATA
```

As for the 1-way coupling the parent instance only provides data, the Fortran95 structure contains considerably less components than the corresponding structure of MMD2WAY_CHILD. Whereas a child can have one parent, a parent can have more than one child. Therefore, the variable of TYPE `T_COUPLE_C_DATA` (`CplData`) is itself component of a structure (TYPE `T_CHILD_DATA`).

```
TYPE T_CHILD_DATA
   ! NUMBER OF EXCHANGED FIELDS
   INTEGER                                          :: NEXCH
   TYPE (T_COUPLE_C_DATA), DIMENSION(:), POINTER :: CplData => NULL()
```

```
    ! POINTER for DATA EXCHANGE TIMING
    REAL(DP), POINTER                              :: Waittime => NULL()
    ! LOGICAL for COUPLING TIME STEP (evaluated in init_loop, required in
    !  init_loop and global start)
    LOGICAL                                 :: lcpl = .FALSE.
  END TYPE T_CHILD_DATA

  TYPE (T_CHILD_DATA), DIMENSION(:), ALLOCATABLE :: CL
```

The Fortran95 variable `CL` is allocated to the number of child instances of the respective parent instances in the initialisation phase.

5.2.1 Initialisation Phase

In the initialisation phase three MESSy entry points are used by MMD2WAY_PARENT: `mmd2way_parent_initialize`, `mmd2way_parent_init_memory` and `mmd2way_parent_init_coupling`.

5.2.1.1 mmd2way_parent_initialise

The parent inquires the number of child instances it has to provide data for. The MMD library variable `MMD_Parent_for_Child` is a list of the parent specific child instance IDs in the MMD setup. In each parent it is allocated to the specific number of child instances the respective parent instance has to deal with. Thus the SIZE of this array equals the number of children of the parent. Subsequently, the MMD library initialisation routines `MMD_P_Allocate_Child` and `MMD_P_Init` are called to initialise the MMD environment of this specific parent instance. `MMD_P_Allocate_Child` accepts a parameter `l2way`: it is `.FALSE.` for 1-way coupling and `.TRUE.` for 2-way coupling.

In MMD2WAY_PARENT itself, the variable `CL` is dimensioned to the number of children. As each child may require a different coupling frequency, the TIMER *event* variables `CPL_EVENT` and `CPL_IOEVENT` must also be available for each child instance individually.

5.2.1.2 mmd2way_parent_init_memory

First, independently for each child the time synchronisation of the respective child-parent pair is triggered by calling `Setup_Child_Timer`:

1. The parent receives the coupling interval in seconds from the child and defines the respective TIMER EVENT (`CPL_EVENT(ic)`).

2. Next, the parent sends its date information and its time step length to the child.
   The following dates must be exchanged to ensure that the models are synchronised: `current_date`, `start_date`, `resume_date` and `stop_date`. The dates are of TYPE `time_days`:

```
  TYPE, PUBLIC :: time_days
     !
     ! relative calendar date and time format
     !
     ! time_days [structure]
     !   day    [integer]  (day in calendar, -2147483648 ..... 2147483647
```

```
   !                        approx. +/-5.8 Mio. years)
   !   second [integer]  (seconds of day, 0,...,86399)
   !
   !PRIVATE
   LOGICAL :: init   = .FALSE.
   INTEGER :: day    = 0
   INTEGER :: second = 0
END TYPE time_days
```

Thus, each date is defined by an INTEGER indicating the day and an INTEGER for the seconds of the day. For all four dates these two INTEGERs are packed into an INTEGER array and sent to the child. As a ninth INTEGER the time step length of the parent is sent. The latter is used by the child to define the `BREAK_EVENT`.

5.2.1.3 mmd2way_parent_init_coupling

The preparations for the 1-way coupling are all performed in the subroutine `mmd2way_parent_init_coupling`. The subroutines described below are processed for each child model individually (indicated below by the index `ic`).

- First, the MMD library subroutine `MMD_P_Get_DataArray_Name` is called, which receives the list of *exchange fields*. After calling this subroutine, these information is available within the MMD library, but not yet made available to MMD2WAY_PARENT itself. This transfer is done within the subroutine `Define_data_arrays` (see last item below).
- Before the *exchange fields* themselves are acquired, the domain section required by the child model for the interpolation of the data, i.e., the domain of the *in-fields* of the child model are determined in the subroutine `Setup_Child_Area`.
  - In a first step the parent model acquires the child model grid information:
    - First, the parent gets the dimensions of the child model COSMO domain (`ie_tot` and `je_tot`) and the number of *exchange fields*.
    - Second, according to the dimensions, fields are allocated for the geographical longitudes and latitudes of the COSMO/MESSy grid points,
    - which are, third, sent subsequently from the child to the parent.
  - Based on the child grid, the domain of the data sent to the child is determined:
    - If a COSMO/MESSy model is the parent, the complete COSMO/MESSy parent domain is used as *in-field* for the child model. In this case simply the information about the parent model COSMO grid are copied to the respective transfer variables: i.e., `startlat_tot`, `startlon_tot`, `endlat_tot`, `endlon_tot`, `pollat`, `pollon`, `dlat` , `dlon`, `ie_coarse`, `je_coarse`, `ke_coarse` , `ke_soil_coarse`, `itype_w_so_rel`, `itype_t_cl`, `vcoord%vcflat`, `svc1`, `svc2`, `vcoord%ivctype` and the derived type `refatm`, with its components defining the reference atmosphere `refatm%p0sl`, `refatm%t0sl`, `refatm%st0lp`, `refatm%delta_t` and `refatm%hscal`, and the depth of the soil layers (`czmls`).
    - If ECHAM5/MESSy is the parent instance, a subset of the global grid is provided as input to INT2COSMO. The size of the domain is determined by the minimum and maximum longitudes and latitudes of the child model domain, including a check whether

the date line is part of the model domain. As INT2COSMO requires a somewhat larger model domain as the COSMO grid to perform the interpolation, four grid boxes are added at each side of the model domain[25]. From this model domain, the location of the corners of the grid in the parallel decomposition of the global model are calculated by the subroutine `locate_in_decomp`.

- The geographical coordinates of the South-West corner determine the start latitude (`startlat_tot`) and longitude (`startlon_tot`) and
- those of the North-Eastern corner the end latitude (`endlat_tot`) and longitude (`endlon_tot`).
- The coordinates of the rotated pole, `pollat` and `pollon`, are always `90._dp` and `180._dp` for a non-rotated grid like the ECHAM5 grid.
- The grid spacings in degrees for the longitudes of the global grid (`dlon`) is easily calculated by dividing `360._dp` by the SIZE of the variables containing the Gaussian longitudes of the global grid.
- The latitudes of a Gaussian grid are not equidistant, however, the grid spacings in degrees is a mandatory input to INT2LM also for the latitudes. Thus, the method implemented in INT2LM when reading and checking the netCDF-file global definitions (subroutine `read_nc_gdefs`)

  ```
  dlat = (MAXVAL(philat) - MINVAL(philat))/(ngl-1)
  ```

  is used. To minimise the error, `dlat` is recalculated after the determination of the section of the grid that is sent to the child using the maximum and minimum latitude of the sent section. Note: as the latitudes in a Gaussian grid are not equidistant this only works satisfactorily if the regional model region is not too close to the poles (with "to close" depending on the ECHAM5 resolution).
- The horizontal dimensions of the exchanged domain `ie_tot` and `je_tot` are calculated according to the difference of the corner indices.
- `ke_tot` is simply `nlev`, i.e., the number of ECHAM5 vertical levels.
- `ke_soil` is `4` as ECHAM5 includes a soil with 5 layers and 4 is the number of interfaces between the soil layers.
- The INT2COSMO variable `itype_w_so_rel`, indicating which type of soil moisture is input to INT2COSMO, must be set to `2` (for both parent models).
- The variable defining the type of the climatological temperature `itype_t_cl` is set to `1`, if ECHAM5/MESSy is the parent, and to `0`, if COSMO/MESSy is parent.
- Additionally, information about the vertical levels of the model domain, i.e., the interface hybrid parameters (`vct`) for ECHAM5/MESSy and `vcoord` for the COSMO/MESSy model are sent to the child.

∗ Last but not least, two fields containing the longitudes and latitudes for each server domain grid point are sent to the child.

- The purpose of the subroutine `Setup_data_exchange_with_Child` is to calculate the `index_list`, i.e., the list which unambiguously associates the grid points with the same geographical coordinates located in the local domains of the child *in-field* and in the parent local domain to each other. Each grid point is defined by the process number (PE) the grid point is located on, and an index pair (i, j) containing the indices of the grid point in the parallel

[25] The size of 4 is arbitrarily chosen, because it worked for the standard ECHAM5/MESSy resolutions so far employed. The really required size depends, e.g., on the rotation of the child grid

decomposed grids. Along with determining this list the `test_array` for the MMD consistency check is filled. Thus, at the beginning of this subroutine the `test_array` is allocated by calling the MMD library routine `MMD_testP_Setup`.

For the calculation of the `index_list` the parent needs the geographic coordinates of the grid points of the local (decomposed) fields from the child. To exchange these, the parent model first receives three INTEGERs: the maximum dimensions for the decomposed horizontal child grid (`ie_max` and `je_max`) and the number of child PEs. According to these dimensions, the parent allocates two three dimensional fields to pick up the decomposed longitude and latitude fields sent by the child.

For each of the grid points in the horizontal domain a list member containing six entries is created. One of these sextuples consists of the child PE (PE_c), the parent PE (PE_p) and the index pairs (i_c,j_c) and (i_p,j_p) of the local decomposed child and parent model domains, respectively, associating the two points with the same geographical coordinates in child and parent grid to each other. The index information about the child grid is inherent in the longitude and latitude arrays. These fields have been gathered in a way, that the index of the third dimension corresponds to PE_c and the indices of the first two dimensions correspond to the indices in the horizontal local grid. The longitude and latitude given by the fields sent by the child are processed by the subroutine `locate_in_decomp`, which locates the respective pair of geographical coordinates on the local decomposed grid of the parent. Output of this subroutine are the PE on which the grid point is located (PE_p) and the indices in the local fields (i_p,j_p). Thus, a sextuple containing all required information about the related grid points of the child and parent domain is complete. Such a sextuple is determined for each of the child instance *in-field* grid points yielding in a field $(i_p,j_p,i_c,j_c,PE_c,PE_p)_n$, with n number of grid points. Additionally, each sextuple is fed into the MMD `test_array` by using the MMD library function `MMD_testP_Fill`. After all grid points have been processed the filling of the MMD `test_array` is finalised by calling the MMD library function `MMD_testP_FinishFill`. The entire `index_list` containing all sextuples is forwarded to the MMD library and analysed within the MMD library routine `MMD_P_Set_Indexlist` establishing the connections between the individual parent and child PEs. A more detailed explanation of this list is given in the MMD library manual, which is part of the same electronic supplement as this manual.

- Last but not least, the POINTERs to the *exchange fields* must be associated during the initialisation phase. This is achieved in the subroutine `Define_data_arrays`:
  - The structure `CL(ic)%CplData`, with `ic` being the index for one child instance, is allocated to the number of required *exchange fields.*
  - The *channel* and *channel object* name of each field and the child model *representation* as listed in the MMD2WAY_CHILD namelist are retrieved by calling the MMD library function `MMD_P_GetNextArray`.
    - If the *channel* name is `'test'` the MMD `test_array` is requested and the POINTER is associated by calling the MMD library routine `MMD_testP_GetTestPtr`.
    - In all other cases the POINTER is associated by calling the CHANNEL subroutine `get_channel_object`. If the object does not exist the simulation is terminated as the required coupling is not possible.
  - When the object exists, the *representation ID* is inquired by calling the CHANNEL subroutine `get_channel_object_info`.
  - The *representation ID* must be known to subsequently retrieve the required dimension informations:

  - * the *axis string*,
  - * the local dimension lengths,
  - * the global dimension lengths, and
  - * the *representation* name.

  The latter two are required, if the *representation* name given in the MMD2WAY_CHILD namelist is `'#UNKNOWN'`. In this case the child needs the additional information plus a possible *attribute* which might contain heights (in case of multi-layer emission fields) to create the correct *representation*. This additional *attribute* is accessed by the CHANNEL subroutine `get_attribute`. All these information are forwarded to the child by the MMD library function `MMD_P_Send_Repr`.

  At the end of the subroutine, the POINTER, the *axis string* and the local *dimensions* are passed on to the MMD library by the subroutine `MMD_P_Set_DataArray` and processed inside the library, i.e., the dimensions and the order information are saved for later use.
- After all data fields are processed a last call during the initialisation phase to the MMD library (subroutine `MMD_P_SetInd_and_AllocMem`) invokes the final evaluation of the dimension information in order to determine the correct buffer size. Subsequently, the actual allocation of the buffer required by MPI takes place calling `MPI_ALLOC_MEM` in the MMD library.

5.2.2 Integration Phase

During the integration phase, in the subroutine `mmd2way_parent_global_start`, the parent provides the *exchange fields* to its child instances. Additionally, in the subroutine `mmd2way_parent_global_end` it informs the child models, whether the simulation is going to be interrupted.

5.2.2.1 mmd2way_parent_global_start

First, the *coupling event* is tested for each child. If the coupling with the respective child is scheduled for the current time step, the MPI Buffer is filled by calling the MMD library subroutine `MMD_P_FillBuffer`. Within this MMD library routine the data is copied to the memory buffer accessible for the child model. To ensure the correct order of accesses to the buffer from the parent and the child, the buffer is locked for that model of a child-parent pair, which latest wrote to/read the buffer. If the workload of the models is not ideally balanced it happens that one of the models has to wait until it can access the buffer again. `MMD_P_FillBuffer` returns the waiting time in seconds for the parent model.

5.2.2.2 mmd2way_parent_global_end

At the end of the time loop, the parent sends the information about the status of the interrupt-switches `lbreak`, `l_rerun` and `lstop`. If the LOGICALs are `.TRUE.`, the respective entry of the INTEGER ARRAY `timeflags` is set to `1`. Otherwise it is set to zero.

5.2.3 Finalisation Phase

At the end of the simulation the allocated memory is released. The subroutine `mmd2way_parent_free_memory` calls, independently for each child, the MMD library subroutines `MMD_testP_FreeMem` and `MMD_P_FreeMem` to release the memory allocated within the library. Additionally, the MMD2WAY_PARENT internal variables `CL(ic)%CplData`, `CL`, `CPL_IOEVENT` and `CPL_EVENT` are deallocated.

6 Coupling of the parent instance to child (2-way, parent-to-child coupling)

The backward coupling from the child to the parent is organized in a similar way as the 1-way coupling, i.e., which input is required from which child for a specific child is determined in the parent namelists (Sect. 3.4).

POINTERs to the respective memory are acquired in the child instance and are interpolated to the parent instance similar "*out-grid*" in the child. Subsequently, the interpolated fields are sent to the parent and applied to the field according to a method determined by the namelist setting independently for each *coupled field*.

In the following, the backward coupling specific routines are described, first for the parent and second for the child instance.

6.1 The parent instance

Four different data types are defined in MMD2WAY_PARENT in order to organise the coupling of the parent to the child.

Two of them are required for the reading and processing of the namelist input of the `&CPL_PAR_CHILD` namelist:

```
TYPE T_EXCH_P_IO
   ! CHANNEL AND CHANNEL OBJECT NAMES IN PARENT
   TYPE(t_chaobj_cpl)                    :: Parentname
   TYPE(t_chaobj_cpl)                    :: Parenttend
   TYPE(t_chaobj_cpl)                    :: Childname
   ! Representation String
   CHARACTER(LEN=STRLEN_MEDIUM)          :: REPR = ''  ! REPRESENTATION STRING
   ! FLAG FOR INTERPOLATION METHOD
   INTEGER                               :: interp = 0 ! INTERPOLATION METHOD
   ! FLAG FOR APPLICATION METHOD overwrite (input field) in GridPoint (appl=0)
   ! FLAG FOR APPLICATION METHOD weighted in GridPoint (1)
   INTEGER                               :: appl = 0   ! application method
   !  weighing factor for "nudging" 1 = hard nudging
   REAL(dp)                              :: fac = 1._dp
END TYPE T_EXCH_P_IO

! MAXIMAL NUMBER OF EXCHANGE FIELDS
INTEGER, PARAMETER                        :: NMAX_P_EXCH = 100
TYPE(T_EXCH_P_IO), DIMENSION(NMAX_P_EXCH) :: PFIELD
TYPE(T_EXCH_P_IO), DIMENSION(NMAX_P_EXCH) :: DEFAULT_PFIELD

TYPE T_P_EXCHG_DATA_IO
   ! use generalised humidity couplings
   LOGICAL                                  :: lgrhum      = .FALSE.
   INTEGER                                  :: i_rmy_px    = 0
   REAL(dp)                                 :: pcontrol_fi = 30000.
```

```
   INTEGER                                    :: itype_fw    = 2
   INTEGER                                    :: icosexp     = 14
   REAL(dp)                                   :: damprel     = 0.2_dp
   INTEGER                                    :: itype_VI    = 1
   INTEGER                                    :: RCF         = 10000
   INTEGER                                    :: RCF_in      = 10000
   LOGICAL                                    :: ldiagonly = .FALSE.
   TYPE(T_EXCH_P_IO), DIMENSION(NMAX_P_EXCH) :: FIELD
   ! allow for free boundary layer
   LOGICAL                                    :: lfreeslip   = .FALSE.
   LOGICAL                                    :: lcpl_gs     = .FALSE.
END TYPE T_P_EXCHG_DATA_IO

TYPE(T_P_EXCHG_DATA_IO), DIMENSION(:), ALLOCATABLE :: PIO
```

Here, `PIO` of TYPE `t_p_exchg_data_io` contains all possible contents of a `CPL_PAR_CHILD` namelist, while `PFIELD` of TYPE `t_exch_p_io` is required to read the array of *exchange fields* (see Sect. 3.4). The 1D array variable `PAR` is of the derived type `T_COUPLE_P` and contains all information required for the coupling of all child instances of a specific parent. It is allocated to the number of coupled child instances early during the initialiation. This structure contains the 1D array variable `CplData`, which is of type `T_COUPLE_P_DATA`, providing all data required for the coupling of the individual field. This is allocated to the actual number of *exchange fields* for the respective child at the end of the namelist interpretation (see Sect. 6.1.3).

```
! PARENT CplData STRUCT
TYPE T_COUPLE_P_DATA
   ! CHANNEL AND CHANNEL OBJECT NAMES IN PARENT
   TYPE(t_chaobj_cpl)                    :: Parentname
   TYPE(t_chaobj_cpl)                    :: Parenttend
   TYPE(t_chaobj_cpl)                    :: Childname
   ! POINTER TO FIELD in Parent model which will be changed
   ! (i.e. for prognostic variables the tendency !)
   REAL(DP), POINTER, DIMENSION(:,:,:,:) :: target => NULL()
   ! POINTER TO FIELD in Parent model required for tendency calulation
   ! (i.e. the m1 field for prog. vars, otherwise  nothing  !)
   REAL(DP), POINTER, DIMENSION(:,:,:,:) :: targetm1 => NULL()

   ! POINTER TO FIELD yielding the interpolated Child fields
   REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr => NULL()
   ! TRACER INDEX
   INTEGER                               :: idt = -99
   ! REPRESENTATION OF PARENT FIELD
   INTEGER                               :: rank
   ! coupling to tendency
   LOGICAL                               :: lte = .FALSE.
   INTEGER, DIMENSION(4)                 :: ldimlen=0
   INTEGER, DIMENSION(4)                 :: gdimlen=0
   ! STRING ORDER OF AXES
```

```
   CHARACTER(LEN=4)                        :: AXIS
   ! FLAG FOR INTERPOLATION METHOD
   ! 1 conservative
   ! 2 bilinear ... (to be implemented)
   INTEGER                                 :: interp = 0
   ! FLAG FOR APPLICATION METHOD
   !  0: as 1: only possible for input fields (i.e., channel objects made by
   !           'mmd2way_parent' itself, not for tendency calculation, not for
   !           tracer (for the parent, the source can of course be a
   !           tracer!)
   !          for appl = 0 the input field is simply transformed to the
   !          full 2D/ 3D parent grid (where frac >0.9999) without
   !          application of any weight function
   !          flagged (i.e., they are /= 0 only where the weigth function is 1
   !  1: field exists only in GridPoint (1) => change tendency
   !     will be worked on in global start
   INTEGER                                 :: appl = 0
   !  weighing factor for "nudging" 1 = hard nudging
   REAL(dp)                                :: fac = 1._dp
   ! Representation String
   CHARACTER(LEN=STRLEN_MEDIUM)            :: REPR
   ! is time dependent ? (relevant Only for COSMO parent)
   LOGICAL                                 :: ltimedep
   ! mmd parent has to define its own memory
   LOGICAL                                 :: l_SentUnit = .FALSE.
END TYPE T_COUPLE_P_DATA

TYPE T_COUPLE_P
   ! NUMBER OF EXCHANGE FIELDS
   INTEGER                                          :: NEXCH
   TYPE (T_COUPLE_P_DATA), DIMENSION(:), POINTER :: CplData => NULL()
   ! POINTER for DATA EXCHANGE TIMING
   REAL(DP), POINTER                                :: Waittime => NULL()
   ! Lon /Lat of incoming data
   REAL(kind=dp),      DIMENSION(:,:,:), POINTER :: all_plon => NULL()
   REAL(kind=dp),      DIMENSION(:,:,:), POINTER :: all_plat => NULL()

   ! weight fractions summed over all child PEs
   REAL(dp),             DIMENSION(:,:), POINTER :: frac => NULL()
   REAL(dp),             DIMENSION(:,:), POINTER :: mask => NULL()
   INTEGER                                       :: kmin
   REAL(dp), DIMENSION(:), POINTER               :: kminfac => NULL()

   ! weighting function
   REAL(dp),             DIMENSION(:,:), POINTER :: wf => NULL()

   ! ENTRIES FOR GENERALIZED HUMIDITY COUPLING:
   ! use generalised coupling
```

```
    LOGICAL                                         :: lgrhum = .FALSE.
    ! data index for vapor
    INTEGER                                         :: idx_qv = -99
    ! data index for cloud water
    INTEGER                                         :: idx_qc = -99
    ! data index for cloud ice
    INTEGER                                         :: idx_qi = -99
    ! data index for meridional wind
    INTEGER                                         :: idx_u  = -99
    ! data index for zonal wind
    INTEGER                                         :: idx_v  = -99
    ! data index for deriv. of meridional wind
    INTEGER                                         :: idx_t    = -99
    ! data index for surface pressure
    INTEGER                                         :: idx_ps   = -99

    ! NAMELIST SWITCHES

    ! treatment of boundary zone for back transition
    INTEGER                                         :: i_rmy_px  = 0
    REAL(dp)                                        :: pcontrol_fi = 30000.

    INTEGER                                         :: itype_fw  = 2
    INTEGER                                         :: icosexp   = 14
    REAL(dp)                                        :: damprel   = 0.2_dp
    INTEGER                                         :: RCF       = 10000
    INTEGER                                         :: RCF_in    = 10000
    LOGICAL                                         :: ldiagonly = .FALSE.
    ! itype_VI switches the vertical interpolation  of C2E
    ! == 1  => NCREGRID
    ! == 2  => INT2LM inverse (MesoTel version)
    INTEGER                                         :: itype_VI  = 1
    !
    ! allow for not forced boundary layer
    LOGICAL                                         :: lfreeslip = .FALSE.
    LOGICAL                                         :: lcpl_gs   = .FALSE.

    REAL(DP), POINTER, DIMENSION(:,:) :: aps => NULL()

  END type T_COUPLE_P

  TYPE(T_COUPLE_P), DIMENSION(:), ALLOCATABLE :: PAR

  ! ------------------------------------------------------------------------
```

The meaning of the components of the derived type variables has already been explained in Sect. 3.4 or will be explained in the following.

6.1.1 mmd2way_parent_initialise

In addition to the inquiry of the number of child instances and the initialisation of the MMD library, which both are required for the 1-way coupling, the variable `PAR` is allocated to the number of child instances and the parent namelists in the namelist file `mmd2way.nml` are read. As explained in Sect. 3 the `mmd2way.nml` namelist file can contain an arbitrary number of `&CPL_PAR_CHILD` namelists. These namelists are attributed to one specific child instance by the entry `INSTANCE`. If the number given by `INSTANCE` is the index of one of the child instances of the parent, this specific namelist is used. However, the first `&CPL_PAR_CHILD` namelist in the namelist file `mmd2way.nml` determines the default setting, which will be used for all instances, for which no individual namelist is provided.

At the end of the subroutine `mmd2way_parent_initialize` the derived type array PIO, containing all information available from the namelist is filled completely.

6.1.2 mmd2way_parent_init_memory

In a loop over all child instances of the specific parent, memory is allocated for fields required for the backward transfer from the child instance:

- The *channel object* `'weightfrac'` is a horizontal field containing for each grid cell the fraction overlapping with the area sent by the child model.
- The *channel object* `'weightfunc'` is a horizontal field and provides the weight with which each grid cell will be weighted, when applied to an existing field of the parent. This weight function depends on the chosen type (namelist parameter `itype_fw`) and is calculated by the child (see page 72).
- The *channel object* `kminfac`, which is a vertical column and contains the vertical weighting coefficients for the vertical application of the field.

For each child model, the index of the child model is appended to the *channel object* name, as the names of the *channel objects* for one parent need to be unique.

6.1.3 mmd2way_parent_init_coupling

The subroutine `mmd2way_parent_init_coupling` performs the main parts for the child-parent coupling.

- **interpret_parent_namelist:**
  The interpretation of the parent namelist entries works in the same way as the interpretation of the namelist of the child for child-to-parent coupling. The namelist can contain wildcards. In this case, the respective child *channel* must contain the same *objects* as the parent model *channel*. The namelist parameters are described in detail in Sect. 3.4. After the analysis of the namelist, the component `CplData` of the derived type variable `PAR` is allocated to the number of respective *exchange fields* and filled with the contents of the intermediate input / output variable `PIO`. As this investigation is performed only in the I/O PE, finally the content of `PAR` is broadcasted to all parent instance PEs.
- In a loop over the child models the namelist content is sent to the respective child model. However, while the control parameters are simply broadcasted, the fields are transfered via the MMD library:

  - **MMD_P_Set_ParDataArray_Name:** The MMD library is filled with required information about the *exchange fields*. These are: the parent *channel* and *channel object* name, the child model *channel* and *channel object* name, the representation, the interpolation method and a LOGICAL determining whether the unit of the *coupled field* should be sent from the child to the parent.
  - The namelist input determining the size and the weights of the area coupled back to the parent, plus some additional information are sent to the respective child model using `MMD_Inter_Bcast`. The respective namelist switches are: `lgrhum`, `i_rmy_px`, `pcontrol_fi`, `itype_fw`, `icosexp`, `damprel`, `RCF`, `RCF_IN`, `itype_VI` and `ldiagonly`.

- In a second loop over the child instances (loop index `ic`),

  1. the parameter `rdheight`, indicating the height in which the damping at the model top starts, is received from the respective child model via the MMD library subroutine `MMD_Recv_from Child`. As the damping zone is not coupled back to the parent model, the parent model subsequently calculates the vertical index of the layer of its own model domain, in which `rdheight` is located and above which the coupling coefficients are zero. This index (`PAR(ic)%kmin`) is sent to the child model and used later on in both models to define *channel objects* of reduced height for the *exchange fields*.
  2. the subroutines `MMD_P_Get_DataArray_Name` and `Setup_Child_Area` are called. They are both required for the 1-way coupling.
  3. the subroutine `Setup_ParData_Exchange` is called. In this subroutine the `index_list` for the parent-to-child coupling is calculated. The procedure is the same as for the child-to-parent coupling as described in Sect. 5.2.1 (more specific page 58). The `index_list` is sent to the child via `MMD_Send_to_Child`. The exchange information required by the parent are finally acquired by calling the MMD library subroutine `MMD_P_Get_ParIndexList`. In addition to the direct grid association information, two additional fields are received:
     - the fractional overlap of the child model grid with the respective parent grid box (`PAR(ic)%frac`), and
     - the weight function (`PAR(ic)%wf`) containing the weights that should be assigned to the respective parent grid box calculated by the child.

     At the end of the subroutine, a mask for the fields that are influenced by more than 99,999% is calculated allowing for simpler postprocessing of data only defined in the coupling region.
  4. At the end of this loop over the child instances the subroutines `Setup_Data_Exchange_with_Child` and `Define_data_arrays` are called. Both have been explained in the context of the 1-way coupling.

- In a third loop over the childrn (loop index `ic`), the memory is set up for the exchange of the *exchange fields*. Additionally, the vertical nudging coefficient is calculated.

  1. The newly calculated index `PAR(ic)%kmin` is used to define two new *representations* for the *coupling fields* with reduced height, i.e., the 3D *representations* `'GP_3D_RDCX'` and `'GP_3D_RDCXp1'` for mid point and interface fields are defined.
  2. The nudging factor for the vertical `Par(ic)%kminfac` is calculated. In all vertical levels below `Par(ic)%kmin` it is preset with `1._dp`. From `Par(ic)%kmin` on to the model top it is preset with `0._dp`. At the moment a quarter cosine function is prescribed, which is zero at level `Par(ic)%kmin` and reaches `1._dp` 10 levels below.

For experimental purposes a similar function can be used at the surface by setting the namelist switch `lfreeslip = .TRUE.`. But this cosine function spreads over 3 levels only.

3. Finally, the memory required for the *exchange fields* is allocated in a loop over the *coupling fields*, established by calling the MMD library function `MMD_P_GetNextParArray`. Different cases are accounted for:
   - A prognostic variable should be changed by the coupling by influencing its tendency:
     * For later use the LOGICAL switch `Par(ic)%CplData(ii)%lte` is set `.TRUE.` indicating that the tendency will be changed.
     * The POINTER to the tendency field (`Par(ic)%CPLData(ii)%target`) is associated to the respective parent model field by calling the subroutine `get_channel_object`.
     * The POINTER to the prognostic field at the end of the last time step (`Par(ic)%CPLData(ii)%targetm1`) is associated to the respective parent model field by calling the subroutine `get_channel_object`.
     * In a next step, the information, if the prognostic field includes a dimension for time levels (as is the case for many COSMO model variables), is generated. If such a dimension exists the LOGICAL switch `Par(ic)%CPLData(ii)%ltimedep` is set to `.TRUE.`.
   - A prognostic variable should be changed by changing the prognostic field itself:
     * The POINTER to the prognostic field (`Par(ic)%CPLData(ii)%target`) is associated to the respective parent model field by calling the subroutine `get_channel_object`.
     * The POINTER (`Par(ic)%CPLData(ii)%targetm1`) is NULLYfied in this case. Note: the POINTER `Par(ic)%CPLData(ii)%target` always points to the object which should be changed.
     * In a next step, the information if the prognostic field includes a dimension for time levels (as is the case for many COSMO model variables) is generated. If such a dimension exists the LOGICAL switch `Par(ic)%CPLData(ii)%ltimedep` is set to `.TRUE.`.
   - The field is only diagnostic input to the parent and the parent has to allocate the memory for this field itself:
     * In this case a new *channel object* is created by calling the subroutine `new_channel_object`. Additionally, if the knowledge of the unit was requested (`Par(ic)%CPLData(ii)%l_SentUnit ==.TRUE.`), the unit is acquired from the child model by calling `MMD_Inter_Bcast`. Subsequently, the received unit is defined as a *channel object attribute*.

   Independent of the *target field*, memory for the *exchange field* needs to be allocated. Dependent on the *representation* of the *target field*, the *representation* of the *exchange field* is determined. Subsequently, the POINTER (`Par(ic)%CPLData(ii)%ptr`) is allocated by calling the subroutine `new_channel_object`.

Afterwards, additional information, as the rank, the axis string and the global and local field length are acquired by calling `get_representation_info`. The newly allocated field, and the information about the local dimensions and the axis string are provided to the MMD library upon calling the MMD library subroutine `MMD_P_Set_ParDataArray`.

To enable a special treatment of certain prognostic variables (especially for diagnostics during the development phase) the indices of these variables are saved.

At the very end of the subroutine `mmd2way_parent_init_coupling` the MMD library routine `MMD_P_SetInd_and_AllocMem` is called. This is required for 1-way and 2-way coupling. Internally, in the case of 2-way coupling the memory of the exchange buffer is allocated to the maximum size required for the data exchange.

6.1.4 mmd2way_parent_global_start

If the coupling to the child is due, the buffer is filled here. The only difference to the 1-way coupling is, that the `Waittime` is not investigated here in case of 2-way coupling.

6.1.5 mmd2way_parent_global_end

The subroutine `mmd2way_parent_global_end` is called three times, therefore the subroutine is called with a parameter `flag`. The call with `flag=2` invokes the exchange of the TIMER status. This has already been discussed in 4.2.1.

In the calls with `flag=1` or `flag=3` the actual change of the parent fields takes place. In a loop over the child instances,

1. it is checked, whether this time step is a coupling time step. If so, for `flag=1` first the buffer exchange takes place by calling the MMD library routine `MMD_P_GetBuffer`. In the case of actual 2-way coupling the `Waittime` is provided by `MMD_P_GetBuffer`. The times are necessarily inquired, after the child model had the access to the exchange buffer. Thus different locations for the `Waittime` inquisition are required.

2. for `flag = 1` or `3` the subroutine `mmd2way_parent_couple_gp` is called. Depending on the flag the tendencies (`flag=1`) or the full fields (`flag=3`) are changed.

If ECHAM5 is the parent instance the calls proceed just one after the other (in the order 1,3,2), while for a COSMO parent, the subroutine is called from three distinct points in the time loop: `flag=1` is called before the subroutine call to `organize_dynamics('compute', ...)`, as in this subroutine the integration of the tendencies proceeds, the tendencies need to be changed beforehand. `flag=3` and `flag=2` are called directly before the COSMO model subroutine call `organize_data('result', ...)`. Note, that the order of the calls is `flag=3`, `flag=2` as the exchange of the timer information is performed last.

Within the subroutine `mmd2way_parent_couple_gp` in a loop over all *exchange fields*, first it is tested whether the current field should be changed in this call, i.e., tendencies are only changed if `flag=1` and `Par(ic)%CPLData(ii)%lte = .TRUE.`, fields are only directly changed, if `flag=3` and `Par(ic)%CPLData(ii)%lte =.FALSE.`. Otherwise the loop is cycled.

For each field loops over the two horizontal (index `jp, jrow`) and the vertical dimensions (`jk`) are performed. For 3D fields, the vertical loop reaches from `Par(ic)%kmin` down to the surface.

For each horizontal grid box it is checked, if the grid box is fully covered by the child domain (`Par(ic)%frac(jp,jrow) > 0.99999_dp`). If so, a weight factor `fc` is calculated as product of

the weight function f_w (`PAR(ic)%wf`) at that point, the fields individual nudging factor f_{field} (`Par(ic)%CPLData(ii)%fac`) and the vertical nudging factor f_{kmin} (`Par(ic)%kminfac(jk)`):

$$fc = f_w(jp, jrow) * f_{field} * f_{kmin}(jk) \tag{1}$$

The actual change of the *target field* F_t, due to an intermediate field F_{coup} is subsequently calculated by:

$$F_t = (1 - fc) \times F_t + fc \times F_{coup} \tag{2}$$

Depending on the coupling method F_{coup} is calculated differently:

- `change of direct field:` For the direct change of the field, F_{coup} is simply the field received from the child model, i.e., the *exchange field* (`Par(ic)%CPLData(ii)%ptr`, F_{exch}).

$$F_{coup} = F_{exch} \tag{3}$$

- `change of tendency:` If the tendency of a prognostic field is changed, F_{coup} is the tendency, invoked by the difference of the field at the end of the previous time step F_{tm1} (`Par(ic)%CPLData(ii)%targetm1`) and the *exchange field*

$$F_{coup} = (F_{exch} - F_{tm1})/dt \tag{4}$$

  As in this case, the *target field* is a tendency field the difference needs to be divided by the time step length (dt).

The above mentioned procedure is the usual one, if the application method "1" is chosen for the field.

A special case has been implemented for the coupling of *input fields*. With application method "0" the weighting of the field is switched off. Thus, the full field as received from the child model is written to the parent *channel object*, on those grid boxes which are completely covered by child model grid boxes.

In the end, for the COSMO model the *target field* needs to be distributed to the halo regions in each local domain. Thus the subroutine `exchg_boundaries` is called for each field.

6.1.6 mmd2way_parent_free_memory

Some additional variables are deallocated in `mmd2way_parent_free_memory`, which have been allocated for the 2-way coupling.

6.2 The child instance

For the parent coupling the MMD child instance sub-submodel MMD2WAY_CHILD contains one additional data structure including all information required for the backward coupling to the parent.

```
TYPE T_P_COUPLE_DATA
    ! CHANNEL AND CHANNEL OBJECT NAMES IN Child
    TYPE(t_chaobj_cpl)                    :: name
    ! POINTER TO CHILD FIELD
    REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr_ori  => NULL()
```

```
    ! POINTER TO INTERPOLATED FIELD
    REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr_int  => NULL()
    ! POINTER TO horizontally INTERPOLATED FIELD
    REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr_hint => NULL()
    ! Interpolation Method
    INTEGER                               :: interpM
    LOGICAL                               :: l_SentUnit
    INTEGER                               :: idt   = -1
    ! Representation String
    CHARACTER(LEN=STRLEN_MEDIUM)          :: repr
    ! ORDER OF AXES IN REPRESENTATION ('X','Y','Z','N')
    CHARACTER(LEN=4)                      :: AXIS= ''
    ! rank of data
    INTEGER                               :: rank = 0
    ! DIMENSION LENGTH
    INTEGER, DIMENSION(4)                 :: ldimlen=0
    ! time dependent prognostic variable?
    ! 0 = not time dependent, 2=time dependent+ horizontal field (2 space dims)
    ! 3 = time dependent, 3 space dimensions
    INTEGER                               :: itimedep = 0
    ! SCRIP DATA
    TYPE(t_scrip_data), POINTER           :: PSD  => NULL()
    ! SCRIP DATA ID
    INTEGER                               :: SD_ID
  END type T_P_COUPLE_DATA
  TYPE(T_P_COUPLE_DATA), DIMENSION(:), ALLOCATABLE :: ParData
```

The individual structure components are:

- the `name` of the *channel* and *channel object* in the child instance,
- the POINTER `ptr_ori` to the original variable in the child, which is coupled back to the parent,
- the POINTER `ptr_int`, which contains the field remapped to the the *out-grid* ($\approx$ parent grid),
- the POINTER `ptr_hint`, which contains the field after horizontal remapping to the *out-grid*, but still without vertical remapping,
- the horizontal remapping method `interpM`, for which currently only `interpM == 1` is implemented, i.e., conservative remapping via SCRIP,
- the LOGICAL `l_SentUnit`. If `.TRUE.` the unit of the *coupled field* is required by the parent model and is sent during the initialisation process.
- the tracer index `idt`. If `idt /= -1`, the *coupled field* is a tracer. This is important information, as the access of the tracer fields is slightly different as for other prognostic fields.
- the representation `repr` of the *coupled field*. The representation is required by child and parent during the initialisation phase for the dimensioning of the interpolated fields.
- the `AXIS` string indicates the order of the dimensions and is required by the MMD library for the correct sorting of the *exchange fields*.

- the `rank` of the *coupled field* is important as fields of different ranks are treated differently, e.g., rank 2 variables do not need to be interpolated vertically.
- the length of the local dimensions of the variable `ldimlen` is required by the MMD library for the internal dimensioning of the *exchange fields*.
- the INTEGER `itimedep` indicates whether the field includes a time dimension. In this case, it has to be taken care that the correct time slice is coupled.
- the derived type variable `PSD` is of TYPE `t_scrip_data` and contains all information required for the remapping via SCRIP (see GRID User manual for more information).
- the INTEGER `SD_ID` provides the ID of the scrip data block.

In addition to the array of derived type variables `ParData`, some additional variables are required for the coupling. These are

- the number of *coupled field*s `PD_NUM`, as requested by the parent.
- the number of additional fields `PD_ADD`, which are required for the interpolation procedure, e.g. the surface pressure.
- a lot of indices indicating the position of a distinct variable in the array `ParData`. Some fields need extra treatment before or during the remapping, i.e., the horizontal wind components `u` and `v` need to be interpolated to the grid box mids if ECHAM5 is the parent.
- additionally, some *in-fields* from the child-to-parent coupling are required for the back transition to the parent.
- an intermediate geopotential field `p_fis` is required.
- the dimensions and representation for the *out-grid* need to be defined and their IDs need to be stored.
- the indices indicating the upper top layer of the coupled domain in the parent (`par_kmin`) and in the child (`chi_kmin`), and the vertical extension of the horizontally remapped (`ke_hint`) and the fully interpolated field (`ke_int`) are declared.
- additionally, the longitudes and latitudes of the rotated *out-grid* (`rlons` and `rlats`) have to be known.
- the full 3D grid and the horizontal grid of the child and the *out-grid* (`cgrid`, `chgrid`, `pgrid` and `phgrid`) are required for the GRID remapping tools.
- the indices of the reduced grid (`iis, iie, jjs, jje`), i.e., those grid boxes of the child grid, which contribute to the field which is coupled back.
- `loverlap` indicates, if at all parts of the grid located on the respective local task are fed back to the parent.
- `conserv_idx` is set to the first variable requiring conservative remapping. This was included, as other remapping methods will be implemented in the future, and as the weight function remapping requires the remapping weights. By setting this index it is omitted to calculate these weights twice.

- `rmy_loc` is a locally defined field, based on the COSMO model variable `rmy` indicating the damping zone. `rmy_loc` indicates the region of the damping zone, plus the number of additionally excluded grid boxes (see page 76).
- a simple `mask`, which is `1._dp` where the fraction of the child model grid box contributes to the *exchange field* by a weight larger than `0.9999_dp`.
- The variable `fractions` contains the fraction to which each of the *out-grid* grid boxes is overlapped by the child's grid. `fracs` is an intermediate version of `fractions` required for the processing.
- `hsurf_full` is an intermediate field required for the vertical backtransition of the fields. The horizontal interpolated field `hsurf_full` must be based on the full (including halo region) local `hsurf` field (in contrast to the coupled fields themselves, which are horizontally interpolated on the reduced grid).
- all parent namelist parameters determining the weighting and remapping procedure have to be known to the child:
  - `i_rmy_px` determines the "additional frame", i.e., the number of grid boxes, which are excluded from the backward coupling in addition to the relaxation zone.
  - `pcontrol_fi` is the control geopotential. It should be the same as for the 1-way coupling. Usually it is 30000 Pa.
  - `itype_fw` determines the type of the horizontal weight function `fw` as calculated on the child grid according to the parent namelist. `fw_int` is the result of the remapping of `fw` to the *out-grid*. `cofw_int` is an intermediate field required for the processing. This weight function `fw` is required to avoid artificial jumps at the borders of the area, where the fields are relaxed to the child variables. Currently, the user can choose between three different implementations by namelist:

    0: f_w is set to 1 everywhere in the child domain. This option is for testing only, as it may lead to artificial jumps in the data.

    1: f_w is implemented as the sum of two cosine functions:

    $$f_w(i,j) \quad = 1. - (cos(x)^e + cos(y)^e) \qquad \text{with } x = \pi \times \frac{i}{i_{max}} \; ; \; y = \pi \times \frac{j}{j_{max}} \tag{5}$$

    i_{max} and j_{max} are the number of grid points in the two horizontal directions, respectively. The exponent e is set by the namelist parameter `icosexp`. Its default value is 14.

    2: f_w decreases in the form of a cosine from 1 in the domain inner part to 0 at the borders of the coupled domain. The width of the damping zone is determined by a namelist parameter `damprel`. Its valid range is `[0,0.5]`. This number determines the relative width of the damping zone. If, for example, `damprel = 0.2` for a model domain consisting of 100 grid boxes in x-direction (index i) and of 50 grid boxes in y-direction (index j), the damping zone in x-direction is 20 grid boxes wide, and in y direction 10 grid boxes wide, respectively.

    All these weight functions are defined on the child grid. They are transformed in the same way as the data will be transformed and sent to the parent for application during the integration phase. Figure 16 displays the different weight functions for a domain over

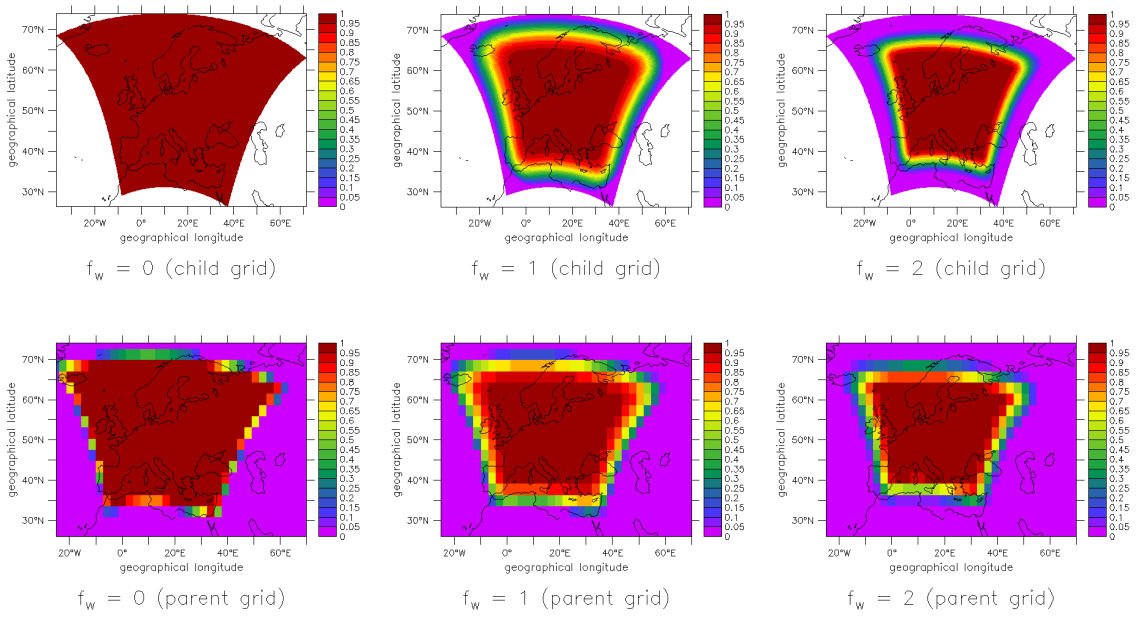

Figure 16: Weight functions (f_w, see page 72) for the different weight types, i.e., `itype_fw` = 0 (left), = 1 (middle) and = 2 (right). Upper row: weight functions as calculated on the child grid. Lower row: weight functions after transformation to the parent grid.

Europe. The upper row shows the weight functions as defined on the child grid. Note, that the coupled domain is smaller than the child domain (with the exception of $f_w = 0$). This is because the damping zone of the regional model itself should not be coupled back to the parent, as this is directly influenced by the parent and thus spurious damping or amplifications for 2-way coupled variables can occur. The lower row of Fig. 16 shows the same weight functions after the transformation to the parent grid.

– `itype_VI` gives the vertical remapping type. At the moment only `itype_VI=1` is implemented, i.e., vertical remapping via NREGRID.

– `RCF` and `RCF_IN` are used to avoid rounding errors in the calculation of the grid longitudes and latitudes. In the COSMO model the longitude of a grid point with i ($lon(i)$) in the local domain is calculated by

```
itot = isubpos(my_cart_id,1) - nboundlines - 1
lon(i) = startlon_tot + (itot + i) * dlon
```

This can lead to rounding errors. Therefore this calculation is shifted to integer arithmetic to minimize these errors:

```
itot = isubpos(my_cart_id,1) - nboundlines - 1
istartlon = NINT(startlon_tot * RCF)
idlon     = NINT(dlon         * RCF)
itmp      = istartlon + (itot+i) * idlon
lon(i)    = REAL(itmp,dp) / RCF
```

Thus, `RCF` and `RCF_IN` should give as many decimals as `dlon` and `dlon_in` have significant decimals. For example, a grid with `dlon = 0.36` should get an `RCF` of `100`.

  - `ldiagonly` is a switch, that can be used, if only fields for diagnostics are coupled. In this case the surface pressure of the parent is used for the backtransition and the iteration of the calculation of the vertical profiles of humidity/ water variables and temperature to calculate the surface pressure is skipped.

- `boxarea_out` provides the area of each grid box of the *out-grid*.
- `l_shortcut` is a switch implemented for testing purposes. If `.TRUE.` the fields as received from the parent are coupled directly back to the parent.
- The 2D POINTER array `iterps` and the arrays `npsiter1`, `npsiter01`, `npsiter001`, `npsiter0001`, and `npsitermax` have been used during the development of the backward coupling. The `npsiterXX` arrays provide the iteration number after which the surface pressure fields deviated less than `1. Pa`, `0.1 Pa`, `0.01 Pa` or `0.001 Pa` from the surface pressure fields of the previous iteration step, respectively. `iterps` contains the surface pressure after each iteration step, where `nitermax` gives the number of possible iterations.

6.2.1 mmd2way_child_setup

The only parent-to-child coupling specific part in `mmd2way_child_setup` is that at the end of the subroutine the parameter `nitermax` is set. It depends on the parent. `nitermax` gives the maximal number of iterations between the calculation of the surface pressure and the vertical profiles of temperature, water vapour, cloud water and cloud ice, which is required during the vertical remapping.

6.2.2 mmd2way_child_init_memory

In `mmd2way_child_init_memory` not only the preparations for both, the 1-way and the 2-way coupling, are performed. In addition:

1. Two *channel objects* are created: the first one contains the horizontal weight function (`fw`) as calculated on the child grid. The other contains the respective `mask`, which is `1` where `fw` is larger than `0.9999._dp`.
2. The MMD library subroutine `MMDC_C_Get_ParDataArray_Name` is called. This subroutine organizes the transfer of the information about the *coupled fields* required by the child for the parent-to-child coupling. Its only parameter is the number of *coupled fields* requested by the parent (`PD_NUM`).
3. The content of the child specific parent `&CPL_PAR_CHILD` namelist is broadcasted using the MMD library subroutine `MMD_Inter_Bcast`. Subsequently, the parameters are saved in the respective local variables, i.e., `lgrhum`, `i_rmy_px`, `pcontol_fi`, `itype_fw`, `icosexp`, `idamprel`, `itype_VI`, `lcpl_global_start`, `RCF`, `RCF_IN` and `ldiagonly`.
4. The height of the damping layer (`rdheight`) is defined in height coordinates. For the coupling this height is required to be in pressure coordinates (`p_rdheight`), which is calculated here. Additionally, the index `chi_kmin` of the highest layer below the damping layer is determined. `p_rdheight` is sent to the parent using the MMD library routine `MMD_Sent_to_Parent`. Subsequently, the parent sends back the corresponding vertical index of this height (`par_kmin`) in the parent grid. This is required for the memory allocation of the *exchange fields*.

5. The subroutine `exchange_grids` serves four purposes: (1) it determines the part of the child domain, which is fed back to the parent; (2) the weight function is calculated, (3) the coupling area for the 1-way coupling is set up (as described on page 31) and (4) the LOGICAL switch `L_gridrotParenteqChild` is set.

   (1) The local COSMO model field `rmy` containing the relaxation coefficients at the borders of the child model domain is gathered and used for the definition of a global mask. `rmy` is zero outside the relaxation zone, whereas it is larger zero in the relaxation zone. Thus, the indices marking where in the global field the relaxation coefficients become zero (`is_p3`, `js_p3`, `ie_p3` and `je_p3`) are determined. These indices are similar to the usual COSMO model notation, as (`istart`, `jstart`) marks the lower left corner of the local COSMO domain, (`is_p3`, `js_p3`) mark the lower left corner of the coupling domain and (`ie_p3`, `je_p3`) describe the upper right corner of the domain coupled back to the parent model. If in addition to the relaxation zone further points should be excluded from the coupling region, the index quadruple is adjusted by adding / substracting the respective number of additional points (namelist parameter `i_rmy_px`). In the global field `rmy_glob` the grid boxes excluded from the coupling are filled with dummy values larger than zero, indicating that they do not contribute to the data coupled back to the parent. In the end the global field `rmy_glob` is distributed again to the local tasks yielding the local decomposed field `rmy_loc`. This is used in `DEFINE_REDUCED_BM_GRID` to determine the local start and end indices for the data entering the remapping routines.

   (2) The available weight functions have been discussed earlier (see page 72). First, the weight function is calculated for a global field `fw_glob`. Afterwards the global weight function field is distributed to yield the local decomposed weight function field `fw`.

   (3) The LOGICAL switch `L_gridrotParenteqChild` indicates, if the grids of the parent and the child are identically rotated. If ECHAM is the parent, this switch is always `.FALSE.`, as ECHAM uses a Gaussen grid and therefore the parent and child grids do not match anyway. For the COSMO model as parent model, it is tested whether `pollon`, `polgam` and `pollat` are the same for both models. In this case `L_gridrotParenteqChild` will be set `.TRUE.`. Alternatively, `L_gridrotParenteqChild` will be set `.TRUE.` if `pollat_in == pollat` and one of `pollon_in` and `polgam_in` is `0._dp` and the other `180._dp`, while `pollon` and `polgam` are defined the other way round. In this case the grid rotation is also the same.

6. In the subroutine `parent_assign_ParData` the definition of the *coupled fields* takes place. Therefore, first `ParData` is allocated to the number of *exchanged fields* (`PD_NUM`) plus a maximum number of *additional fields*, which might be required for the remapping procedure. At this point only the number of namelist requested fields is known, not the fields themselves, therefore the exact number of *additional fields* is not known and therefore the maximum number has to be used. In an endless loop (loop index `ii`) calling the MMD library function `MMD_C_GetNextParArray`, the name of the *channel* (`ParData(ii)%name%CHA`) and the *channel object* (`ParData(ii)%name%OBJ`) of the coupled child field, its representation (`ParData(ii)%repr`), the interpolation method (`ParData(ii)%interpM`) and the information, if the unit of the field should be sent (`ParData(ii)%l_SentUnit`) are saved. For the special case that an array is to be coupled, which is defined by MMD2WAY_CHILD itself, a special action is required during the child-to-parent coupling. Therefore the index of this field in the `ParData` array has to be saved in the derived type variable (`CplData`). Thus, in this case `CplData(ii)%scn` is set to the current loop index `ii`.

7. In the subroutine `match_parent_grid` the weights for the remapping during the backward coupling are calculated. First, the longitudes (`rlons`) and latitudes (`rlats`) of the grid mid points of the rotated *out-grid* are calculated, and afterwards the subroutine `CALC_backward_WEIGHTS` is called. Before the weights can be calculated in subroutine `CALC_backward_WEIGHTS`, the grids for the remapping need to be defined.

   - **definition of reduced child grid**
     As discussed before, not the full child grid is used for the backward coupling. Thus first, the start grid of the remapping, the so-called "reduced basemodel grid", needs to be defined as geo-hybrid grid. This is done in the subroutine `define_reduced_BM_grid`. The reduction of the grid is twofold:

     (a) In the horizontal the points of the relaxation zone plus the "additional frame" as given by the namelist parameter `i_rmy_px` are excluded. In case of a very broad additional frame, it can happen, that the local domains located at the border of the full COSMO domain do not contribute any grid boxes to the backward coupling. In this case the LOGICAL `l_overlap` is set to `.FALSE.` for an easier handling of "empty" or "non-active" local domains.

     (b) The vertical dimension can be reduced by the damping zone at the model top. Its length is thus given by `nlev+1-chi_kmin+1`.

     For the special case that the child and the parent are both COSMO instances and their grids are rotated identically (indicated by the LOGICAL `L_gridrotParenteqChild`) the subroutine `define_rotred_BM_grid` is called instead of the subroutine `define_reduced_BM_grid`. This subroutine takes advantage of the identical rotation, which helps to avoid additional numerical inaccuracies by additional grid rotations.

     For the data processing this 3D grid and a horizontal grid are required. Thus in addition to the reduced child grid (`cgrid`), the reduced horizontal child grid (`chgrid`) is defined by using the GRID subroutines `COPY_GEOHYBGRID` and `SWITCH_GEOHYBGRID`.

   - **definition of the** *out-grid*
     The definition of the 3D and the horizonal *out-grids* (`pgrid` and `phgrid`) follows the same lines as the definition of the reduced child grid. Here, the *in-grid* information as required for the 1-way coupling is used. The length of the vertical dimension is given by `ke_int = ke_in - par_kmin + 1`.

   - **calculation of backward weights**
     For the horizontal remapping, SCRIP, as provided in the generic submodel GRID, is used. Thus, the reduced child and parent grids, are used to define the grid information required by the SCRIP software. This needs to be done for each field individually, as in principle different remapping algorithms can be chosen for each field individually. The GRID subroutine `CALC_SCRIPDATA` extracts the information required by SCRIP from the geo-hybrid grids `cgrid` and `pgrid`. These data are saved in the variable `ParData(ii)%PSD`. Subsequently, these are used to calculate the weights for the horizontal remapping (subroutine call to `CALC_SCRIP_WEIGHTS`). For later use, the index `fw_SD_ID` is set to the index of the first *coupled field* requiring conservative remapping.

   - **remapping of the weight function**
     As all information required for a horizontal remapping is now available, in a last step, the horizontal weight function `fw` is remapped to the *out-grid* yielding the remapped horizontal weight function `fw_int`. If the index `fw_SD_ID` is set, the remapping weights have already

been calculated and can be used for the weight function remapping, otherwise, they are calculated here. By calling the subroutine `RGTOOL_CONVERT_DAT2VAR` `fw` is converted to the 1D format required by GRID. Afterwards, the remapping proceeds by calling `SCRIP_CONTROL`. Finally, the remapped data are converted from the GRID 1D format to yield `fw_int`.

8. **Setup_ParData_exchange**
   The subroutine `Setup_ParData_exchange` serves two purposes:

   (a) It provides the data to calculate the IndexList for the data backtransition and receives the IndexList from the parent. This works in the same way as described for `Setup_data_exchange_with_parent` for the data exchange between parent and child.

   (b) The information which fraction of the *out-grid* is overlapped by the local reduced child grid is retrieved. Principally, the SCRIP data contains the information (`ParData(conserv_idx)%PSD%wghts%dstfrac`). As the SCRIP data is in the GRID 1d format, it needs to be de-alined to yield the required 2d data field `fractions`. In the subroutine `determine_fractions` the destination fraction of a conservativly remapped field is de-alined to yield a 2D array defined on the *out-grid*.

9. **parent_make_representations**
   In this subroutine the *representation* for the *channel objects* defined on the *out-grid* are build. First, six *dimenions* are defined:

   - the first horizontal dimension `'MMDOUT_ie'` (dimension ID: `DIMID_POUT_IE`),
   - the second horizontal dimension `'MMDOUT_je'` (dimension ID: `DIMID_POUT_JE`),
   - the vertical dimension of the remapped field `'MMDOUT_kmin'` of length `ke_int` (dimension ID: `DIMID_POUT_KMIN`),
   - the vertical dimension for the interfaces of the remapped field `'MMDOUT_kminp1'` of length `ke_int+1` (dimension ID: `DIMID_POUT_KMINp1`),
   - the vertical dimension of the horizontally remapped field `'MMDOUT_hint'` of length `ke_hint = ke - chi_kmin + 1` (dimension ID: `DIMID_POUT_HINT`),
   - the vertical dimension for the interfaces of the remapped field `'MMDOUT_hintp1'` of length `ke_hint+1` (dimension ID: `DIMID_POUT_hintp1`).

   Subsequently, by using these *dimensions*, the following *representations* with the decomposition type `DC_MMD_OUT` are defined:

   - `REPR_POUT_2D` (dimension ID: `REPR_POUT_2D`): the horizontal *out-grid*, defined by the dimensions `'MMDOUT_ie'` and `'MMDOUT_je'`.
   - `REPR_POUT_3D_MID` (dimension ID: `REPR_POUT_3D_MID`): the *out-grid* defined on the grid mid-points, i.e., by the dimensions `'MMDOUT_ie'`, `'MMDOUT_je'` and `'MMDOUT_kmin'`.
   - `REPR_POUT_3D_INT` (dimension ID: `REPR_POUT_3D_INT`): the *out-grid* horizontally defined on the grid mid-points, vertically on the interfaces, i.e., by the dimensions `'MMDOUT_ie'`, `'MMDOUT_je'` and `'MMDOUT_kminp1'`.
   - `REPR_PHOUT_3D_MID` (dimension ID: `REPR_PHOUT_3D_MID`): the horizontally remapped grid defined on the grid mid-points, i.e., by the dimensions `'MMDOUT_ie'`, `'MMDOUT_je'` and `'MMDOUT_hint'`.

- `REPR_PHOUT_3D_INT` (dimension ID: `REPR_PHOUT_3D_INT`): the horizontally remapped grid, horizontally defined on the grid mid-points, vertically defined on the interfaces, i.e., by the dimensions `'MMDOUT_ie'`, `'MMDOUT_je'` and `'MMDOUT_hintp1'`.

For the same five combinations of *dimensions* additional five *representations* are defined using the decomposition type `DC_MMD_OUTACC`. Thus, the five respresentations `REPR_POUTACC_2D`, `REPR_POUTACC_3D_MID`, `REPR_POUTACC_3D_INT`, `REPR_PHOUTACC_3D_MID` and `REPR_PHOUTACC_3D_INT` are defined. "ACC" indicates that these objects need to be "accumulated" when written to a file. Normally, the fields on the COSMO grid consist of the so-called "inner domain" and a halo-region. The inner domains are disjunct on all PEs. Thus, for the output the content of the inner domains of each task is dumped to the output file. This is different for the fields remapped to the *out-grid*. Here, each task contains the fraction, which was covered by the reduced child grid. Thus on output, to yield the full field, all fractional contributions of the single tasks need to be summed up. This is invoked by the decomposition type `DC_MMD_OUTACC`.

10. **parent_set_ParData**
In the subroutine `parent_set_ParData`, the *channel objects* for the back transition are defined. These are:

- the interpolated weight function `fw_int`,
- the `fractions` of the *out-grid* covered by the local task,
- the arrays for the diagnostics of the surface pressure iteration, and
- the area of the *out-grid* grid boxes `boxarea_out`.

In a loop over all *coupled field*s

(a) the indices of variables, which require special treatment during the remapping are saved.

(b) depending on the representation of the field, the new *channel objects* for the backtransition, i.e., `ParData(ii)%ptr_hint` and `ParData(ii)%ptr_int` are defined, using the above mentioned *representations*.

(c) the `rank`, `AXIS` string and the length of the local dimensions `ldimlen` are inquired by calling `get_representation_info`.

(d) by calling the MMD library subroutine `MMD_C_Set_ParDataArray`, the memory and the dimension information are provided to the MMD library.

For later use, during this loop an index of a field remapped with interpolation method 1 (`idx_int1`) and of a field which *coupled field* is of representation `GP_3D_MID` (index `idx_p_3d`) is saved.

Finally, additional fields are added to the list of fields that need to be remapped. Basically, these are all variables which are required for the adjustment of the vertical profiles, i.e.,

- for ECHAM as parent instance: the surface pressure `'PS'`, the surface geopotential `'FIS'`, the control level geopotential `'FIC'`, the 3D pressure field `'PRES'`, the 3D temperature field `'T'`, water vapour `'QV'` and cloud water `'QC'`
- for COSMO as parent instance: the 3D temperature field `'T'`, the surface pressure `'PS'`, the surface temperature `'T_S'`, the height of the surface `'HSURF'` and the 3D deviation of the pressure field `'PP'`.

These fields are added by calling the subroutine `add_parent_list_element`.

11. Finally, in the subroutine `init_parent_coupling`, the POINTERs to the child fields of the *coupled field*s are acquired. In the simplest case, this is achieved by calling `get_channel_object`. Additionally, if the unit of the *coupled field* is required by the parent model, the unit is sent here.

    In some very specific cases the memory needs to be allocated in MMD2WAY_CHILD itself, which is done by first finding the correct representation ID by calling the CHANNEL subroutine `get_representation_info` using the representation as sent by the parent `ParData(ii)%repr`. Secondly, the memory is allocated by calling `new_channel_object` using the representation ID.

    Finally, for both cases the information, if the *coupled* child *field* depends on time (`ParData(ii)%itimedep`), is set by inquiring, whether the representation of the child field contains the timelevel dimension ID `DIMID_TLV`.

6.2.3 mmd2way_child_init_loop

In `mmd2way_child_init_loop` no backward coupling specific tasks need to be performed.

6.2.4 mmd2way_child_global_end

Apart from the exchange of the TIMER information, which is already required for the 1-way coupling, in `mmd2way_child_global_end` the remapping of the fields to the *out-grid* and the data exchange from the child to the parent takes place. The latter consists simply of the call of the MMD library routine `MMD_C_FillBuffer`. However, the subroutine `interpol_parent_data` contains the full machinery for the remapping of the child fields to the *out-grid*. Chronologically, in the subroutine `interpol_parent_data` in a loop over the *coupling fields*

1. the vertical extend of each field is determined by setting the two indices `kkmin` and `kkmax`. For a 2D field both indices are `1`, for a 3D field is `kkmin = chi_kmin`, i.e., the first level below the damping layer, and `kkmax = SIZE(ParData(ii)%ptr_ori,3)`, i.e., the vertical dimension of the *coupled field*.

2. the POINTER to the COSMO field, that should be coupled is associated. For the variables, which require specifically precalculated fields and not the original COSMO data the POINTER `PTR` is set to this specific field, otherwise `PTR` is set generically to the *coupled field* assigning automatically the region, that should be coupled, i.e., reduced to the reduced horizontal domain (`iis:iie`, `jjs:jje`), the reduced height (`kkmin:kkmax`) and the correct timelevel (`tlev:tlev`).

3. the data is transformed to the GRID internal 1D format using the GRID subroutine `RGTOOL_CONVERT_DAT2VAR`, if the POINTER is associated,. Afterwards, the horizontal remapping proceeds within the subroutine `SCRIP_CONTROL`. This subroutine hands back the remapped field in the 1D GRID format. This variable is transferred back by calling the GRID subroutine `RGTOOL_CONVERT`, which result (`dat`) is the horizontally remapped field `ParData(ii)%ptr_hint`. Additionally to the data transformation, the intermediate 3D grid `intgrid`, i.e., the grid on which `ParData(ii)%ptr_hint` is defined (i.e., horizontally equal to the *out-grid* and vertically still the reduced COSMO model grid), also output by the subroutine `SCRIP_CONTROL` is saved in the variable `phgrid_3d`, as this is required for the vertical regridding.

Last but not least, the vertical remapping takes place. The sequence of surface pressure adaptation and vertical remapping is adopted from the respective procedures in INT2LM. Different algorithms are called depending on the parent instance:

- ECHAM is parent instance: **vert_interpol_lm2echam**
  - First, a LOGICAL mask (`lcalc`) is defined, which is `.TRUE.` if the horizontal weight function is larger than zero. This mask is used later on, to limit the calculations to those grid boxes, where meaningful values exist.
  - Second, the iteration of the surface pressure takes place. This step is skipped, if `ldiagonly=.TRUE.`. In the latter case the original surface pressure of the parent model is used. At the beginning of the iteration a first guess surface pressure is calculated using the subroutine `CALC_ALPS_1` which works in a similar way as `calc_alps_1` in INT2LM. Then, in an iteration loop, the vertical profiles for temperature, water vapour, cloud water and cloud ice, are calculated based on the current surface pressure guess. Based on this profiles a new surface pressure is calculated by the subroutine `calc_alps_2`, with which, in the next cycle, new vertical profiles are calculated. For diagnostic purposes each step of the iterated surface pressure is saved in a *channel object* and the differences between the surface pressures of the individual iteration steps are analysed.
  - After the iteration, the vertical profiles of all *exchange fields* are calculated using the "final" surface pressure.

  The vertical profiles are calculated using the subroutine `vert_c2p_ncinterpol`. Within this subroutine NREGRID as provided by the GRID submodel is used to perform the vertical remapping. First, the definitions of the geo-hybrid grids of the in-coming grid is adapted to the needs of only vertical regridding. Second, the surface pressure in the grid definition is set by calling the GRID subroutine `SET_SURFACE_PRESSURE`. Afterwards, the data is converted to the 1D GRID format by the GRID subroutine `RGTOOL_CONVERT_DAT2VAR`. The vertical remapping proceeds within the subroutine `REGRID_CONTROL`. Finally, the INTENT(out) variable containing the fully remapped field is filled, after the backtransition into the full three dimensional data field using the GRID subroutine `RGTOOL_CONVERT`.

- COSMO is parent instance: **vert_interpol_lm2lm**
  This subroutine is a copy of the INT2LM subroutine `org_vert_inter_lm`, which was adapted to the backward coupling of the data.
  - First, the reference atmosphere for the intermediate (only horizontally remapped) data is calculated (here the `hsurf_full` field is required to avoid decomposition dependent results).
  - Secondly, after computation of some helper variables, the pressure deviation field is vertically remapped using the subroutine `vert_interp`. This subroutine is again a copy of the INT2LM routine and performs a spline interpolation of the vertical profiles.
  - Third, the vertical velocity is remapped vertically.
  - Next, within a loop, all other variables are remapped vertically to the parent grid.
  - In the end, the pressure deviation field is corrected, taking the new vertical profiles into account.

6.2.5 mmd2way_child_write_output

As explained for the subroutine `parent_make_representation` (page 77) the *channel objects* of decomposition type `DC_MMD_OUTACC` are summed up when the respective *channel objects* are gathered. Thus the fields need to be multiplied by the fraction of the *out-grid*, which is covered by the child grid on the respective task before the call to the MESSy output routines. After output was performed, they must be divided by the fraction to yield back the original fields. This is done in the subroutine `mmd2way_child_write_output`.

6.2.6 mmd2way_child_read_restart

This subroutine calls `mmd2way_child_write_output` for the conversion of the fields into fractions.

6.2.7 mmd2way_child_write_restart

This subroutine calls `mmd2way_child_write_output` for the summation of the fields.

6.2.8 mmd2way_child_free_memory

In `mmd2way_child_free_memory` the memory required for the backward coupling is deallocated, i.e., `p_fis`, `hsurf_full` and `ParData`, in addition to the deallocations required for the 1-way coupling.

7 Changes in INT2LM code required for the MESSy submodel INT2COSMO

Most changes to the INT2LM code have been embedded in the preprocessor directive `I2CINC` (INT2COSMO IN COSMO). However, in order to improve the 2-way coupling additional extensions have been added using the preprocessor directive `MESSYTWOWAY`. In this section the changes and the reasons for them are listed for each code file. The files are listed in alphabetical order. The changes refer to INT2LM version 2.00.

7.1 data_fields_lm.f90 /data_fields_in.f90

As all INT2COSMO fields are allocated as MESSy *channel objects*, they have to be declared as POINTERs, instead of ALLOCATABLE ARRAYs. The definitions are replaced throughout the module for all REAL arrays.

In `data_fields_lm.f90` some extra fields are defined:

- `zfi_fl`, `zhi_fl`, `zps1_lm`, `zkzgr`: in the off-line INT2LM these fields are locally defined intermediate variables, which are used during the remapping. In INT2COSMO these fields are also required for the remapping of the *additional fields*. Therefore, they are declared in `data_fields_lm.f90` and allocated as *channel objects* in `messy_I2CINC_channel_alloc_lm`.
- `oromem_lm`: this field is required in the subroutine `external_data.f90`. Here, the orography or better the surface height at the first call of INT2LM must be saved and used in the next calls, otherwise the results of INT2COSMO could depend on the restart frequency depending on the parent.

- `geolon_lm, geolat_lm, geoloni_lm, geolati_lm`: longitude and latitude fields in geographical coordinates for the INT2COSMO fields. They are required for the definition of the geo-hybrid INT2COSMO grids, if conservative remapping is chosen.
- `lon_lm, lat_lm, loni_lm, lati_lm` contain the (interface) longitude and latitude fields in rotated coordinates for the INT2COSMO fields. They are required for the definition of the geo-hybrid INT2COSMO grids, if conservative remapping is chosen.
- `psiter01_lm, psiter02_lm, psiter03_lm, psiter04_lm` and `psiter05_lm`. These fields are defined for diagnostic output during the development of the 2-way coupling. Thus they only exist, if the preprocessor directive `MESSYTWOWAY` is active.
- `fic2_gl, fic3_gl`: these fields are only defined, if MESSYTWOWAY is active. They are used for some diagnostic purposes during the development of the 2-way coupling.

In `data_fields_in.f90` additionally the fields `lon_in` and `lat_in` are declared, which are required for the definition of the geo-hybrid INT2COSMO grids, if conservative remapping is chosen.

7.2 data_grid_in.f90

In `data_grid_in.f90` the indices for the local decomposed *in-grid* `istartpar_in, iendpar_in, jstartpar_in, jendpar_in` takes place. These indices are required for the gathering of the fields defined on the *in-grid* if they should be written to the output. Additionally, the two geo-hybgrids `pingrid` and `pinhgrid` are declared here. They are required for the conservative remapping.

7.3 data_grid_lm.f90

Instead of declaring and defining the grid in INT2COSMO independently, the number (`ke_soil_lm`) and depths (`czmls_lm` and `czhls_lm`) of the soil layers and the grid dimensions and orientation (`dlon`, `dlat`, `startlon_tot`, `startlat_tot`, `polgam`, `pollon`, `pollat`, `ielm_tot`, `jelm_tot` and `kelm_tot`) are USEd directly from the COSMO model and renamed to their INT2COSMO names:

```
USE data_modelconfig,  ONLY: czmls_lm => czmls, czhls_lm => czhls  &
                           , ke_soil_lm => ke_soil, ielm_tot => ie_tot &
                           , jelm_tot => je_tot, kelm_tot => ke_tot    &
                           , dlon, dlat, pollat, pollon, polgam        &
                           , startlat_tot, startlon_tot
```

Furthermore, four additional index variables are declared, which are required for the grid mapping of the COSMO and the INT2COSMO grid (`istartcos`, `iendcos`, `jstartcos` and `jendcos`). The meaning of these variables is explained in Sect. 5.1.4.

Additionally, for MESSy the knowledge of the hybrid grid coefficients is required during the model initialisation. Therefore the fields `ak_lm` and `bk_lm` are declared.

Furthermore, for the conservative remapping and the backward coupling the respective geo-hybrid grids are defined here:

- `i2cgrid`: the INT2COSMO grid without halos and full vertical height
- `i2chgrid`: the purely horizontal INT2COSMO grid without halos
- `i2cUgrid`: staggered U INT2COSMO grid without halos and full vertical height
- `i2cUhgrid`: the purely horizontal staggered U INT2COSMO grid without halos
- `i2cVgrid`: staggered V INT2COSMO grid without halos and full vertical height
- `i2cVhgrid`: the purely horizontal staggered V INT2COSMO grid without halos.

Finally, the respective SCRIP data variables (`I2C_SD`, `I2C_U_SD`, and `I2C_V_SD`) and the indices of the SCRIP data sets (`I2C_SD_ID`, `I2C_U_SD_ID`, and `I2C_V_SD_ID`) are declared here.

7.4 data_int2lm_control.f90

As INT2COSMO and the COSMO model need to be set up in the same way, some run control variables are directly used from the COSMO model. Thus those declarations in `data_int2lm_control.f90` of the INT2COSMO namelist parameters, which are also declared for COSMO, are omitted.

```
USE data_runcontrol, ONLY: nstop, nstart, llake, lradtopo, lprog_qi  &
                         , itype_calendar, idbg_level, lforest, lsso &
                         , lmulti_layer_lm => lmulti_layer           &
                         , lemiss, lseaice, lstomata                 &
                         , lperi_x, lperi_y, l2dim, itype_aerosol    &
                         , itype_albedo
USE data_io,         ONLY: lbdclim
```

7.5 data_int2lm_io.f90

To make INT2COSMO as inherently consistent with the COSMO setup as possible, the variable `ydate_ini` containing the start date of the simulation is USEd from the COSMO model module `data_io` instead of being declared within this file.

7.6 data_int2lm_parallel.f90

As INT2COSMO is run in the same parallel environment as the COSMO model, most of the variables are USEd from the COSMO module `data_parallel`, instead of being defined within INT2COSMO:

```
USE data_parallel, ONLY: ldatatypes,lasync_io,nprocx,nprocy,nprocio,nproc,      &
                         num_compute, num_io,ncomm_type,my_world_id,my_cart_id, &
                         my_cart_pos,my_cart_neigh,igroup_world,icomm_world,    &
                         icomm_compute,igroup_cart,icomm_cart,icomm_row,        &
                         iexch_req,imp_reals,imp_grib,imp_integers,imp_byte,    &
                         imp_character,imp_integ_ga,imp_logical,lcompute_pe,lreorder
```

7.7 data_parameters.f90

To be consistent, the KIND parameters are overwritten by those determined within the MESSy submodel `messy_main_constants_mem.f90`. `ireals` and `idouble` are set to `dp`, `iintegers` is set equal to `i4` and `isingle` to `sp`.

7.8 external_data.f90

- The MESSy submodel MMD2WAY_CHILD calls the subroutine `external_data` with an additional parameter: `lread`. This LOGICAL indicates whether the external data should be read or not. When `lread` is `.FALSE.`, the initialisation of the LOGICALs (indicating the existence of specific variables in the external data file) with `.FALSE.` and the subroutine `read_lm_ext` are not processed. Additionally, `rootdp_mx` must only be set, if data was read, i.e., `lread=.TRUE.`.
- All parameters and variables determining the coarse grid are directly exchanged with the server via MMD. Thus the subroutine `read_coarse_grid_ext` is not called in INT2COSMO. The LOGICALs `lfis_in` and `lfrla_in` are set to `.TRUE.`, according to the fields exchanged during the on-line coupling.
- The variables `fr_land_in`, `z0_in`, `plcov_in`, `plcmx_in`, `plcmn_in`, `rlaimx_in`, `rlaimn_in` and `root_in` are not deallocated from INT2COSMO. In INT2COSMO these variables are *channel objects* and as such automatically deallocated by the CHANNEL submodel.
- The allocation of `local_iso_points` and its calculations are only performed at the first call of INT2COSMO.
- The possibility to remap all variables with the conservative remapping provided by GRID was added. Two (hard-coded) additional LOGICALs determine, whether the external data should be remapped using SCRIP.
  - `l_interp_crl` is used to change from linear interpolation (interpolation flag 'L') to conservative remapping. This concerns the fields 'FR_LAND', 'FIS_IN' and 'HSURF_IN'.
  - `l_interp_crm` is used to change from match point interpolation (interpolation flag 'M') to conservative remapping. This concerns the fields 'Z0', 'ROOT', 'PLOCV_MX', 'PLOCV_MN', 'LAI_MX' and 'LAI_MN'.

  Both switches are experimental. Therefore they are `.FALSE.` by default.
- As already mentioned for the file `data_fields_in.f90`, it is important, that the calculation of the fields `hsurf_lm` and `fis_lm` takes place with the same surface orography in all time steps. Otherwise the results are restart frequency dependent. Therefore, the variable `oromem_lm` contains a copy of `hsurf_lm` at the last time step, where external data was actually read.

7.9 interp_utilities.f90

This code file contains all subroutines used for the horizontal remapping. Thus, for the on-line coupling a subroutine `interp_c` was added, which links to the MESSy submodel GRID providing the conservative remapping.

7.10 setup_int2lm.f90

- The memory allocation for the fields of INT2COSMO is modified (compared to INT2LM) from allocating the fields to defining *channel objects* for them. For the definition of *channel objects* the *representations* of these objects must be specified. Within the subroutines `messy_I2CINC_channel_make_MMDC4`, `messy_I2CINC_channel_alloc_lm` and `messy_I2CINC_channel_alloc_cg`, which are called from `setup_int2lm`, the *dimensions* and *representations* required for the *channel objects* are created and the *channel objects* are defined.
- As INT2COSMO uses the parallelisation of the COSMO model, the subroutines `init_environment`, `init_procgrid`, `mpe_io_init` and `mpe_io_reconfig` are skipped. On the other hand, the arrays `isubpos_in` and `isubpos_in_red` are allocated and initialised here. They are required for the on-line coupling to address the input fields correctly.
- INT2LM offers the possibility to measure the timing of different phases of the remapping process. Most of the required calls occur in subroutines and in the main program, which are skipped in INT2COSMO. Therefore, the initialisation of the timing located in `setup_int2lm` is skipped for INT2COSMO too.
- The subroutine `clm_setup` provides utilities for the setup of climate simulations. However, as this relies on input files not available for the on-line coupling this subroutine is skipped.
- The debug output file is renamed to `'YUDEBUG_i2cinc'`, because the COSMO model also writes a file named `'YUDEBUG'`.
- At the end of the subroutine, the length of the sent buffer (`isendbuflen`) is determined. The calculation relies on the knowledge of the decomposition of the INT2COSMO grid. In INT2LM the local domains are equally dimensioned, depending on the total number of grid boxes (`ielm_tot` or `jelm_tot`) plus the boundary lines (`nboundlines`) and the number of processes (`nprocx` or `nprocy`). This is no longer correct in INT2COSMO. Here, the local domain size also depends on the COSMO number of boundary lines (`nboundlines_cosmo`). This is taken into account in the calculation of `isendbuflen`.

7.11 src_2d_fields.f90

`t_so_lm` is defined in the vertical from `0:ke_soil+1`. As *channel objects* can only be allocated starting by 1, due to the used POINTER arithmetic, `t_so_lm` is allocated in the vertical by `1:ke_soil+2`. This has to be taken into account for the calculation of `t_so_lm`. For instance,

```
#ifndef I2CINC
          t_so_lm(i,j,0) = t_s_lm(i,j)
        ELSE
          t_so_lm(i,j,0) = undef
#else
          t_so_lm(i,j,1) = t_s_lm(i,j)
        ELSE
          t_so_lm(i,j,1) = undef
#endif
```

All places, where `t_so_lm` is calculated or used, are changed accordingly.

Additionally, the option of conservative remapping has been added for `'w_so_rel'`.

7.12 src_cleanup.f90

In INT2COSMO the subroutine `free_memory` is almost completely skipped, as all fields deallocated in INT2LM in this subroutine are declared as *channel objects* and thus deallocated automatically within the CHANNEL submodel. Only the three LOGICAL (land-sea) masks `lolp_lm`, `lolp_in` and `lmask_lm` are allocated manually in `messy_I2CINC_channel_alloc_*` as they can not be defined directly as *channel objects* and thus are still deallocated within `free_memory`.

7.13 src_coarse_interpol.f90

- In case of INT2COSMO the on-line exchanged data is always handled similar to `'ncdf'` data. Thus, for the undefined values always the undef flag `undefncdf` is used. In INT2LM the value of the variable `undef` is determined by the type of the external data fields, i.e., `undef = undefgrib` for grib-files and `undef = undefncdf` for netCDF-files. In INT2COSMO it is possible, that the external file is in grib format, but the on-line data is processed like netCDF data. To ensure the correct setting of `undef`, it is set to `undefncdf` before the remapping starts in INT2COSMO.
- To be able to treat the soil temperature *in-field* `t_so_in` as *channel object*, its vertical dimension is allocated by `1:ke_soil+2` instead of `0:ke_soil+1`. The change of the indexing has to be taken into account in `src_coarse_interpol.f90`, when calculating `zdt_so` and when the remapping for the surface levels is called: i.e., `CALL interp_l(t_so_in(:,:,1),...)` instead of `CALL interp_l(t_so_in(:,:,0),...)`.
- The option to choose conservative remapping has been added for all fields.

7.14 src_decomposition.f90

- As the parallel decomposition of INT2COSMO is matched with the parallel decomposition of the COSMO model, the decomposition routine for the child model (`decompose_lm`) has been partly rewritten as explained in Sect. 5.1.4.
- The call to the subroutine `read_nc_axis` is skipped, as all information about the parent are exchanged on-line.
- The fields determining the *in-grid*, i.e., `isubpos_in`, `isubpos_in_red` and the indices `istartpar_in, iendpar_in, jstartpar_in` and `jendpar_in`, are calculated in addition.

7.15 src_lm_fields.f90

Changes for the on-line coupling (preprocessor directive `I2CINC`):

- `zfi_fl` is an intermediate variable calculated within the subroutine `org_lm_fields`. As it is required for the remapping of the *additional fields* as well, it is declared in `data_fields_lm` and allocated as *channel object*.
- In order to interpolate the *additional fields*, the subroutines `vert_int_lm` and `vert_z_lm` have been expanded to vertically interpolate all possible input variables and not only those specified in the code by name.

Changes for the backward coupling (preprocessor directive `MESSYTWOWAY`):
In the initialisation phase, `qv_lm` and `qc_lm` are calculated from the generalised humidity. Additionally, the possibility to avoid the generalised humidity and directly interpolate the individial moisture variables has been added.

7.16 src_lm_output.f90

First, due to the unification of the COSMO and INT2COSMO software packages, the routines `write_grib` and `write_gribapi` require slightly different parameters in the off-line and the on-line coupling. Additionally, due to the different handling of the variables defined from `0:ke_soil_lm+1` one if-statement had to be changed.

7.17 src_namelists.f90

Most of the changes in this file are due to the fact, that in INT2COSMO many INT2LM namelist switches are determined by the COSMO model or the parent model. Thus, they must not be read in anymore. The following tables list (for each namelist) those variables excluded from the namelist. The header of the column indicates the place (basemodel or MMD2WAY_CHILD) where the variables are set instead.

&contrl				
Set by COSMO	Set by COSMO (continued)	Set by COSMO (continued)	Set by MMD2WAY_CHILD	not valid for on-line coupling
ydate_ini	lprog_qi	itype_aerosol	linitial	
ydate_bd	itype_calendar	lemiss	lboundaries	lante_0006
hstart	lforest	lstomata	lgme2lm	lpost_0006
hstop	lsso	nhori	lec2lm	yinput_model
hincbound	lradtopo	lseaice	llm2lm	
nincbound	llake	itype_albedo	lhm2lm	
nincwait	ldbclim		lcm2lm	
nmaxwait	lasync_io		licon2lm	
ytrans_in	lreorder		itype_w_so_rel	
ytrans_out	lmulti_layer_lm		itype_t_cl	
nprocx	ldatatypes		l_bicub_spl	
nprocy	ncomm_type			
nprocio	idbg_level			

Two more `&contrl` namelist parameters are not read anymore:

- `nboundlines` is not read from the namelist, as it is always 1 for INT2COSMO.
- `lmulti_layer_in` is set in accordance to `lmulti_layer_lm` for INT2COSMO.

&grid_in		
Set by parent	Set by parent (continued)	Set by MMD2WAY_CHILD
ie_in_tot	czml_soil_in	lushift_in
je_in_tot		lvshift_in
ke_in_tot		
nlevskip		
pollat_in	startlat_in_tot	
pollon_in	startlon_in_tot	
polgam_in	endlat_in_tot	
dlat_in	endlon_in_tot	
dlon_in	ke_soil_in	

&lmgrid	
Set by COSMO	Set by COSMO (continued)
ielm_tot	dlon
jelm_tot	dlat
kelm_tot	startlat_tot
ke_soil_lm	startlon_tot
pollat	czml_soil_lm
pollon	czvw_so_lm
polgam	

&data
not valid/needed for on-line coupling
yinext_cat
yinext_lfn
ybitmap_cat
ybitmap_lfn
yin_cat
ylm_cat
nprocess_ini
nprocess_bd
yinext_form_read
yin_form_read

The namelists `&prictr` and `&epsctl` have not been changed. Checks required for the namelist switches are omitted, if the variables were removed from the namelist.
Additionally, the default setting for the vertical coordinates and the reference atmosphere (subroutine calls `set_vcoord_defaults` and `set_refatm_defaults`) are omitted, as these are called from the COSMO model.

7.18 src_read_coarse_grid.f90

Of this module only the subroutine `org_read_coarse_grid` was modified for the implementation of INT2LM as MESSy sub-submodel INT2COSMO:

- If called from MMD2WAY_CHILD, the subroutine `org_read_coarse_grid` is called without any arguments, as these are only required to read the data files, which is omitted in INT2COSMO.
- The file-type determination and read procedure dependent code blocks are skipped.
- The variable `fic_in` (control geopotential) is used for the remapping. As it is based on *driving model* fields, it needs to be recalculated every time step at which remapping of boundary data occurs, in order to get reproducible results. Thus, `fic_in` is calculated every time in INT2COSMO by omitting the if-statement for `var_in(mzfi_loc_in)%lreadin`. Additionally, this calculation has been modified for MESSYTWOWAY. Here `qc_in` and `qi_in` are taken into account for the calculation of the virtual temperature `ztv`.
- In INT2COSMO the *in-field* for `T_SO` is in the vertical dimension defined from `1` to `ke_soil+2` (instead of `0:ke_soil+1` in INT2LM). Thus, the surface temperature is copied to the index 1 in INT2COSMO.

```
var_in(n)%p3(1:ie_in,1:je_in,1) = &
var_in(mzts_loc_in)%p2(1:ie_in,1:je_in)
```

7.19 src_read_ext.f90

- Due to the unification of the COSMO and INT2COSMO software packages, the routines `read_grib`, `read_gribapi`, `check_input_grid` and `mpe_io_read` require slightly different parameters in the off-line and the on-line coupling.
- Due to the on-line coupling, the calculation of the surface geopotential field must be handled slightly differently.

7.20 src_read_hhl.f90

Due to the unification of the COSMO and INT2COSMO software packages, the routine `read_gribapi` requires slightly different parameters in the off-line and the on-line coupling.

7.21 src_vert_inter_lm.f90

The variable `zhi_fl`, which is only temporarily calculated within the subroutine `org_vert_inter_lm` in INT2LM, is also required for the remapping of the *additional fields*. Therefore, it is converted to a *channel object* in INT2COSMO instead of being defined locally in INT2LM. Additionally, the boundary layer height is stored in the *channel object* `zkzgr`.

Furthermore, the clipping of tracer mass mixing ratio or other *additional fields* smaller than 10^{-12} is omitted for I2CINC.

7.22 src_vert_interpol.f90

- In order to interpolate the *additional fields*, the intermediate variables `zps1_lm` and the boundary layer top `kzgr` are converted to *channel objects* to be available in MMD2WAY_CHILD. As `kzgr` is an INTEGER and *channel objects* need to be of type REAL `kzgr` is stored in a REAL variable called `zkzgr`.
- For test purposes, the possibility to vertically remap the individual hydrometeors instead of the generalised humidity, is added (`lgrhum =.FALSE.`).
- To avoid rounding errors, in the subroutine `uv_correction` the latitude is calculated using the indices of the total fields instead of the local indices.

Additionally, for the tests with the 2-way coupling, the iteration over the calculation of the vertical profiles of temperature and moisture variables and surface pressure calculation has been added.

8 Changes in the COSMO code required for the on-line coupling

The COSMO model code has been changed for two reasons:

1.) The reading of the initial and boundary data files is obsolete and thus skipped, if the data is calculated on-line by the MESSy submodel MMD2WAY_CHILD. The preprocessor directive `I2CINC` (INT2COSMO `IN` COSMO) accomplishes this.

2.) The internal MPI environment settings need to be adjusted to the MPI environment, as required for the on-line coupling and managed by the MMD library. These changes are introduced using the preprocessor directive `MESSYMMD`.

In this section the COSMO model source files changed by these two preprocessor directives are listed and the changes are explained in detail.

8.1 Application of the preprocessor directive `I2CINC`

In `organize_data.f90` the size of a buffer had to be adapted to the larger size required for INT2COSMO. The only other modified file is `src_input.f90`. It manages the reading of the initial and boundary data. When the COSMO model is a child, the preprocessor directive `I2CINC` prevents the opening and reading of the initial and/or boundary files:

```
#ifdef I2CINC
  ! SKIP READ-IN-PROCEDURE IN CASE OF I2CINC FOR 'initial' and 'boundary'
  IF ((ydata /= 'initial' .AND. ydata/='boundary')  &
       .OR. (.NOT. L_IS_CHILD)) THEN
#endif
```

The variable `undef` is usually set in one of these skipped sections, for a defined preprocessor directive `I2CINC` `undef` is set at the end of the section:

```
 #ifdef I2CINC
  IF (yformat /= 'ncdf') THEN
    undef     = REAL(undefgrib, ireals)
  ELSE
    undef     = REAL(undefncdf, ireals)
  ENDIF
#endif
```

In addition, specific variables are deallocated in COSMO without testing, if they were allocated. In case of `I2CINC`, the state of the variables `iblock`, `ibmap`, `ds_grib`, `ds_real`, `dsup` and `idims_id_in` is tested first, before they are deallocated.

8.2 Application of the preprocessor directive `MESSYMMD`

8.2.1 environment.f90

In its usual configuration the COSMO model is run in its own MPI environment. In this case the model wide communicator `icomm_world` is equal to `MPI_COMM_WORLD`. When COSMO is running within an MMD environment, it only runs on a subset of the tasks of the MPI environment. Therefore, the model wide group communicator needs to be provided by MMD. This is done within the subroutine `MMD_get_model_communicator`. The subsequent use of the worldwide communicator `MPI_COMM_WORLD` would lead to errors. Thus `MPI_COMM_WORLD` was substituted by the model wide communicator `icomm_world`. To perform this substitution, the subroutine `init_procgrid` (part of the module file `src_setup.f90`) is called with the additional parameter `icomm_world`.

The memory allocated by the MMD library needs to be released at the end of a simulation. Thus, the MMD library subroutine `MMD_FreeMem_communicator` is called from the COSMO subroutine `final_environment`.

8.2.2 src_setup.f90

According to the changes in `environment.f90` the subroutine `init_procgrid` has an additional parameter (`icomm_world`), which is used instead of `MPI_COMM_WORLD` within the subroutine.

9 Changes in the ECHAM5 code required for the on-line coupling

When ECHAM5/MESSy is the *partiarch* in the MMD setup, the MPI environment needs to be changed accordingly. The preprocessor directive for these changes is the same as in COSMO, i.e., `MESSYMMD`.

9.1 mo_mpi.f90

- If ECHAM5/MESSy is the only executable running in an MPI environment, the communicator required to communicate with all PEs of this model is easily determined by duplicating `MPI_COMM_WORLD` by calling the subroutine `MPI_COMM_DUP` into the model wide communicator `p_all_comm`. When ECHAM5/MESSy is running within an MMD environment, the model wide communicator `p_all_comm` is not equal to `MPI_COMM_WORLD`. Thus, the correct communicator is determined by the MMD library subroutine `MMD_get_model_communicator`.

- Before the simulation is terminated, the memory allocated by the MMD library needs to be released. This is achieved by calling `MMD_FreeMem_Communicator` from the ECHAM5 subroutine `p_stop`.

9.2 scan1.f90

One additional change had to be made, for ECHAM5/MESSy as parent. The temperature is not initialised before the start of the time loop. But, when ECHAM5/MESSy is parent, the first action taken in the time loop is to send the data for initialisation to the child model. Thus the temperature needs to be initialised before the first call to `messy_global_start` in case of the very first model start (`lstart = .TRUE.`). This is achieved by calling the ECHAM5 subroutine `initemp` before `messy_global_start` when `MESSYMMD` is defined.

In the files `mo_spitfire.f90`, `mo_semi_lagrangian.f90` and `mo_tpcore.f90` the preprocessor directive `_XNOZEROINITEND` is activated, which provokes that already calculated tendencies are taken into account, avoiding, that the advection starts only from the 'm1'-value.

Glossary

In addition to the explanation of the individual terms, Fig. 17 illustrates the meaning of the different *coupling fields*.

- *additional field*: An *additional field* is a field requested in the MMD2WAY_CHILD namelist in addition to the fields already taken into account by INT2COSMO. Furthermore, for the backward coupling, *additional fields* are fields which are required in addition to the *coupled fields* for the adaption of the vertically remapped profiles.
- *attributes*: *Attributes* represent time independent, scalar characteristics, e.g., the measuring unit.
- *axis string*: The *axis string* is defined for each *representation*. It indicates the order of the 'X', 'Y', 'Z' and 'N' direction, e.g., a 3D variable in COSMO/MESSy has the *axis string* `'XYZ-'`, whereas the same variable in ECHAM5/MESSy has the *axis string* `'XZY-'`.
- *boundary field*: It is used to prescribe the variables at the model domain boundaries.
- *break event*: The *break event* is an *event* that is triggered each parent time step in order to receive the information from the parent, whether the parent is going to be interrupted after the current time step.
- *channel*: The generic submodel CHANNEL manages the memory and meta-data and provides a data transfer and export interface (Jöckel et al., 2010). A *channel* represents sets of "related" *channel objects* with additional meta information. The "relation" can be, for instance, the simple fact that the *channel objects* are defined by the same submodel.
- *channel object*: It represents a data field including its meta information and its underlying geometric structure (*representation*), e.g., the 3D vorticity in spectral *representation*, the ozone mixing ratio in Eulerian *representation*, the pressure altitude of trajectories in Lagrangian *representation*.

- *coupling event*: This is an *event* scheduling the data exchange from the parent to the child instance. Its time interval has to be a multiple of the child and the parent time step lengths.
- *coupled field*: this term is only used for the backward coupling. This is the original child field, which is requested in the parent namelist.
- *coupling field*: A *coupling field* is either an *exchange field* or a field required during the remapping procedure: either by INT2COSMO, i.e., the fields deduced from the external parameters, e.g. `lai`, `rootdp`, etc., or by the remapping algorithm for the backward coupling.
- *dimensions*: They represent the basic geometry of one dimension, e.g., the number of latitude points, the number of trajectories, etc.
- *driving model*: The parent model as it provides the *in-fields* to INT2LM / INT2COSMO.
- *event*: This is a data type provided by the generic submodel TIMER, which is used to schedule processes at specific (regular) time intervals, e.g., to trigger regular output or input during a simulation. The *event* control is part of the MESSy generic submodel TIMER. The electronic supplement of Jöckel et al. (2010) comprises a manual for TIMER and details about the *event* definition.
- *exchange field*: An *exchange field* is a field requested within the `CPL_CHILD_COSMO`/`CPL_CHILD_ECHAM` or in a `CPL_PAR_CHILD` namelist, which needs to be provided by the sending model. For the 1-way coupling, an *exchange field* can either be a field which is remapped and copied to a child model variable, or a field required for the remapping itself. For the backward coupling, the *exchange field* is the field remapped by the child model, which is sent to the parent.
- *in-field*: The *in-fields* are those fields provided by the parent or *driving model*, which are defined on the *in-grid* which is (a part of) the parent grid, but defined by the child. In other words, *in-fields* are the *exchanged fields* before the remapping.
- *in-grid*: The *in-grid* is defined by the child instance. It is the grid on which the *in-fields* are defined, i.e., a subpart or the full parent grid.
- *initial fields*: One destination type of data fields provided by MMD2WAY_CHILD to the child. *Initial fields* are only used to initialise fields at the very beginning of the simulation.
- *input fields*: One destination type of data fields provided by MMD2WAY_CHILD to the child instance. *Input fields* are *additional fields*. The newly remapped field replaces the field in the child instance, e.g., an emission field, that is down-scaled from the parent.
- *INT2COSMO inherent field*: This is a field, which is considered and remapped within INT2COSMO or INT2LM (it is part of the variable table in INT2LM).
- *intermediate field*: The *intermediate field* is the "work space" for the remapping. It contains the fields after horizontal and/or vertical remapping.
- *mandatory field*: This is an *in-field* absolutely required either by the COSMO model setup or for the remapping itself.

- *master parent*: The *master parent* or *patriarch* is the coarsest model in a model cascade, i.e., that model that has no parent model itself. In the MMD library namelist this model is indicated by a "-1" as associated parent model. In most cases this is a global model. The *patriarch* determines the time setting of the entire model cascade.

- *out-grid*: The *out-grid* is a subpart of the parent model grid, defined by the child submodel MMD2WAY_CHILD. This is the target grid for the remapping of the child model fields to the parent grid before the remapped data is sent back to the parent.

- *patriarch*: The *patriarch* or *master parent* is the coarsest model in a model cascade, i.e., that model that has no parent model itself. In the MMD library namelist this model is indicated by a "-1" as associated parent model. In most cases this is a global model. The *patriarch* determines the time setting of the entire model cascade.

- *pointer array*: is an array of POINTERs of a specific dimension. For instance, a 2D-POINTER array `example_ptr` is defined by:

  ```
  TYPE (PTR_2D_ARRAY), DIMENSION(:), POINTER :: example_ptr  => NULL()
  ```

  with

  ```
  TYPE PTR_2D_ARRAY
   REAL(DP),DIMENSION(:,:),POINTER :: PTR
  END TYPE PTR_2D_ARRAY
  ```

- *representation*: It describes multidimensional geometric structures (based on *dimensions*), e.g., Eulerian (or grid point), spectral, Lagrangian.

- *representation ID*: in the CHANNEL submodel the *representation*s are stored as a list. Thus each *representation* is unambiguously identifiable by an identification number (ID).

- *restart*: *restart* is used as synonym for check-pointing here. It is performed to allow branching off additional simulations, or as fallback option in case anything went wrong during the simulation, or if the computing time allowed by a scheduler of a super-computer is to short to fit in the complete simulation. Check-pointing means, that the simulation is interrupted in between and restarted as a new job. To achieve binary identical results for simulations with and without interruption, restart files are written, of which the contents fully determine the state of a model simulation. These files are read in the initialisation phase during a model *restart*.

- *target field*: For the 1-way coupling this term specifies those fields on which the results of INT2COSMO are written, i.e., those fields used in the COSMO/MESSy simulation. For the parent-to-child coupling these are the parent fields, which are modified by the remapped and exchanged data.

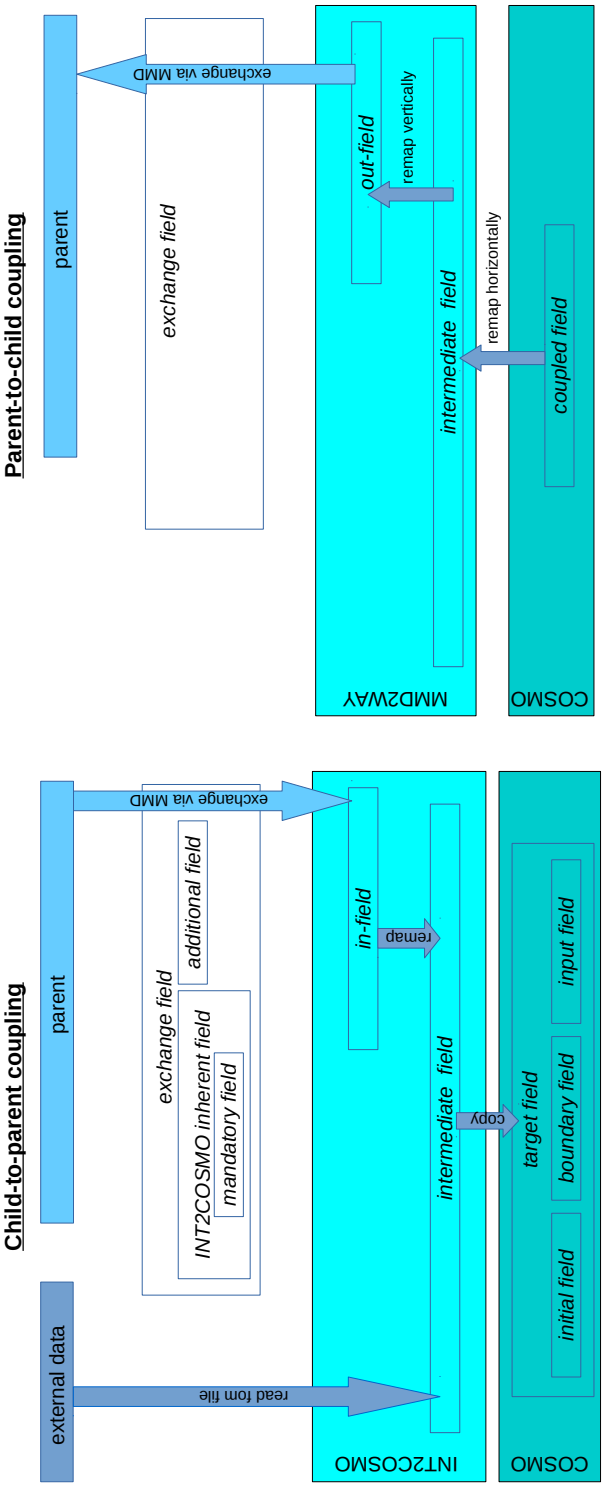

Figure 17: In the manual a lot of different specifications (all listed in the glossary) are used. *Coupling fields* are all fields somehow onvolved in the coupling procedure. The picture illustrates the different stages and the meaning of the specific fields.

References

Baumgaertner, A. J. G., Jöckel, P., Kerkweg, A., Sander, R., and Tost, H.: Implementation of the Community Earth System Model (CESM) version 1.2.1 as a new base model into version 2.50 of the MESSy framework, Geoscientific Model Development, 9, 125–135, doi:10.5194/gmd-9-125-2016, 2016.

Jöckel, P., Kerkweg, A., Buchholz-Dietsch, J., Tost, H., Sander, R., and Pozzer, A.: Technical Note: Coupling of chemical processes with the Modular Earth Submodel System (MESSy) submodel TRACER, Atmos. Chem. Phys., 8, 1677–1687, doi:10.5194/acp-8-1677-2008, 2008.

Jöckel, P., Kerkweg, A., Pozzer, A., Sander, R., Tost, H., Riede, H., Baumgaertner, A., Gromov, S., and Kern, B.: Development cycle 2 of the Modular Earth Submodel System (MESSy2), Geosci. Model Dev., 3, 717–752, doi:10.5194/gmd-3-717-2010, 2010.

Kerkweg, A., Sander, R., Tost, H., and Jöckel, P.: Technical Note: Implementation of prescribed (OFFLEM), calculated (ONLEM), and pseudo-emissions (TNUDGE) of chemical species in the Modular Earth Submodel System (MESSy), Atmos. Chem. Phys., 6, 3603–3609, 2006.

Pozzer, A., Jöckel, P., Kern, B., and Haak, H.: The Atmosphere-Ocean General Circulation Model EMAC-MPIOM, Geoscientific Model Development, 4, 771–784, doi:10.5194/gmd-4-771-2011, 2011.

Sander, R., Baumgaertner, A., Gromov, S., Harder, H., J'öckel, P., Kerkweg, A., Kubistin, D., Regelin, E., Riede, H., Sandu, A., Taraborrelli, D., Tost, H., and Xie, Z.-Q.: The atmospheric chemistry box model CAABA/MECCA-3.0, Geosci. Model Dev., 4, 373-380, doi:10.5194/gmd-4-373-2011, 2011.