TM5-MP USER MANUAL (r454)

Ph. Le Sager, KNMI
2016-11-29

Table of Contents

1. Introduction and conventions

This document is a user guide to TM5-MP, which differs from its predecessor TM5 (referred as TM5-zoom from now on) in its MPI implementation and by the absence of zoom options. All shell commands in the text assume BASH shell.

2. TM5-MP subversion repository cheat-sheet

Here is a very short guide for using TM5-MP subversion repository. It is not an introduction to subversion (svn), but the collection of svn commands used in this manual, gathered here for quick reference.

```
# environment
export repo='https://svn.knmi.nl/svn/TM5-MP'

# start with a frozen release...
svn co $repo/tags/1.0b

# ...or a branch for development
svn cp $repo/trunk $repo/branches/mybranch -m "create my fancy branch"
svn co $repo/branches/mybranch

# save your work
```

```
cd mybranch
svn mkdir DIR         # create DIR and schedule for addition
svn add NEWFILE       # schedule NEWFILE for addition
svn del OLDFILE       # schedule OLDFILE for removal
svn ci -m "message"   # commit
svn up

# get updates committed by other developers of the same branch
svn up

# incorporate changes from the trunk into my clean (without edits) branch
cd mybranch
svn merge ^/trunk

# get information
svn info
svn log
svn stat
svn diff
svn annotate

cd mybranch
svn mergeinfo ^/trunk

# undo
svn merge ^/branches/mybranch -c -NUMBER  # undo locally the commit NUMBER
svn revert FILENAME                       # undo local edit of FILENAME

# reintegrate my branch into the trunk [Administrators only]
cd trunk
svn merge ^/branches/mybranch

# resolving conflict
# TODO

# move your local edits to a new branch
svn copy  $repo/trunk  $repo/branches/new-branch -m "created new branch"
svn switch ^/branches/new-branch
svn ci -m "message"
```

3. Getting the code

3.1. Preliminaries

For convenience, the SVN repository root address is exported to the environment:

```
export repo='https://svn.knmi.nl/svn/TM5-MP'
```

The *repo* variable is used throughout the manual.

3.2. Getting access to the repository

3.2.1. Credentials

To get access, you need a username and password that can be obtained after contacting the repository administrator. These credentials will let you

- visit the repository through a web interface, and

- use the subversion software to get a copy of the code, and commit your own code to the repository.

At some location, subversion accesses internet over a proxy. Ask your administrator for details on how to set up your connection. One proxy example is the ECMWF gateway. See hereafter.

3.2.2. Proxy set up: example of ecgate at ECMWF

To access the subversion server at KNMI from the 'ecgate' computer at ECMWF, first check if the following file exists:

```
~/.subversion/servers
```

If not, run 'svn help' to create one. Add the following lines at the end of the file (in the [global] section) :

```
http-proxy-host = proxy.ecmwf.int
http-proxy-port = 3333
```

3.2.3. First time connection to KNMI-SVN server

The first time you try to use svn (for example on ECMWF ecgate machine), you may get an "error" like :

```
Error validating server certificate for
'https://svn.knmi.nl:443':
- The certificate is not issued by a trusted authority. Use the fingerprint to
validate the certificate manually!
Certificate information:
- Hostname: svn.knmi.nl
- Valid: from Nov 30 00:00:00 2009 GMT until Nov 29 23:59:59 2012 GMT
- Issuer: TERENA, NL
- Fingerprint: ab:47:53:94:cb:de:7b:5.........
(R)eject, accept (t)emporarily or accept (p)ermanently?
```

You must accept permanently. Then enter your password. However, the server first assumes that your KNMI-SVN user id is the same as your client id (ECMWF id for example). So it may not work, and will ask a second time, now asking for both user id and password. Enter your KNMI-SVN ones, and you are all set.

3.2.4. Web interfaces

You can have a look at the TM5 code in the repository through a user-friendly web interface:

- http://dev.knmi.nl/projects/tm5mp/repository
- https://svn.knmi.nl/usvn/project/TM5-MP

Note that the web interface is convenient for peeking at the code and the latest commits, but **cannot** be used to get a copy of the code or run any svn command.

3.3. Get a copy for testing or production runs

You need to checkout the version XYZ you want to use:

```
svn co $repo/tags/XYZ  MYNAME
```

where MYNAME is not mandatory, and defaults to "XYZ".

3.4. Get a copy for development

You need to create and checkout a branch. Most of the time you branch out from the stable trunk, i.e. your branch starts as a copy of it:

```
svn cp $repo/trunk  $repo/branches/mybranch -m "create my fancy branch"
svn co $repo/branches/mybranch MYNAME
```

where MYNAME is not mandatory, and defaults to "mybranch".

4. Get the input data

4.1. Full sets

4.1.1. Meteo

See http://tm.knmi.nl/index.php/ERA_Interim_meteo for a description of the driving met fields, and to find out where to retrieve them from.

4.1.2. Emissions for chemistry projects

The large emissions inventories for year $YYYY can be retrieved from the following locations at ECMWF. For all these 4 inventories, once unpacked, specify their path in your "chem.input.rc". It is suggested to put each of them in its own directory below a common directory: EMISSDIR/MEGAN, EMISSDIR/EDGAR, EMISSDIR/AR5,…

1. MACC-City (1998-2009, anthropogenic + natural)

```
ec:/nm6/EMISSIONS/MACCity/${YYYY}.tar
ec:/nm6/EMISSIONS/MACCity/gridbox_area.nc
```

2. MEGAN (2000-2010, biogenic)

```
ec:/nm6/EMISSIONS/MEGAN/MEGAN_${YYYY}.tar
```

3. EDGAR 4.x

   EDGAR 4.2 (1970-2008, anthropogenic sectors except transport) along EDGAR 4.1 (2005,transport sector only) can replace MACC-City for the anthropogenic emissions. But it is used in any case for CH4, which is missing in MACC-City.

```
ec:/nm6/EMISSIONS/EDGAR4/gridbox_area.nc
ec:/nm6/EMISSIONS/EDGAR4/EDGAR41_2005.tar

ec:/nm6/EMISSIONS/EDGAR4/EDGAR42_${YYYY}.tar
```

   After extraction, put all the EDGAR files in the same TGT target directory (TM5 requirement):

```
mv EDGAR41_2005/*    TGT
mv EDGAR42_${YYYY}/* TGT
```

4. GFED3 (1998-2010, biomass burning)

```
ec:/nks/EMISSIONS/GFED3/GFED_BCOC.tar # for OC/BC
ec:/nm6/EMISSIONS/GFED3/GFED3_$YYYY.tar # for the other species
```

5. AR5

The AR5 inventory provides BC and OC emissions when running with the M7 aerosols module. Other species (anthropogenic and biomass burning) emissions are used when running TM5 coupled to EC-Earth, and will soon be replaced with CMIP6 inventories.

```
ec:/nks/EMISSIONS/AR5/AR5_$YYYY.tar # for historical period (up to 2000
included)
ec:/nks/nmi_ECFS/EMIS/AR5/IPCC_AR5_emissions_$RCP_$YYYY.tar.gz
```

4.1.3. Other inputs

The "common input dataset" (land-sea mask, photolysis data, aerosols, LPJ emissions inventories…) is used by all projects and found in:

```
ec:/nm6/EMISSIONS/MISC/TM5_INPUT_20150520.tar.gz
```

Set the key "my.data.dir" in your 'machine.rc' file to the path where you unpacked this archive. That is:

```
my.data.dir : /YOUR/PATH/TO/TM5_INPUT
```

4.1.4. Restart files

Initial values of the state variables are available for testing the chemistry. Several restart files options are available for testing purposes in the "input data" from the previous section:

```
/YOUR/PATH/TO/TM5_INPUT/testdata_restart
```

You can test restart options 31, 32, 33, and 5 with those data. See the README therein for details.

4.2. Benchmark subsets

For testing purposes, a subset of the larger datasets is available. You still need to retrieve the "common input data set". You can get a subset of 2005-2006 data from MACC-City, MEGAN, GFED3, AR5, EDGAR:

```
ec:/nm6/EMISSIONS/MISC/TM5_EMISS_20160923.tar.gz
```

For the meteo, the following sets are enough to simulate January 2006:

```
ec:/nm6/EMISSIONS/MISC/bench_v4_meteo_20130711_mfuvw.tar.gz
ec:/nm6/EMISSIONS/MISC/bench_v4_meteo_20130711_ml60.tar.gz
ec:/nm6/EMISSIONS/MISC/bench_v4_meteo_20130711_sfc.tar.gz
ec:/nm6/EMISSIONS/MISC/bench_v4_meteo_20140319_conv.tar.gz
```

5. First things after checkout

5.1. TM5 rc files

For a general introduction to the rc files used by TM5, please go to http://tm.knmi.nl/index.php/User_interface_:_RC_files.

To get started, copy a template from the rc sub-directory to the top directory:

```
cd MYNAME

cp rc/main-base.rc.tmpl  my.rc           # for the base only
```

```
cp rc/main-chem.rc.tmpl  my.rc                    # for the chemistry
cp rc/chem-input-default.rc.tmpl  chem-input.rc  # for the chemistry
```

Modify these copies for your environment (next section) and experiment. With the exception of few keys, most of the description on the TM5-zoom wiki (starts here) is still valid, notably the Main RC file, Restart options, and Output options.

Some information that you will not find on these wiki pages:

- The flag "with_optics" is used to computed AOD from M7. It has then no effect if "with_m7" is not set. Note that if "with_m7" is set but not "with_optics", then M7 tracers are transported but not used to compute AODs.

5.2. Environment setup

5.2.1. Requirements

You need to have HDF4 and netCDF4 libraries installed. The netCDF4 and the underlying HDF5 must be installed with parallel I/O enabled if you want to use "exact" restart files or output chemistry time-series. You also need python, subversion (1.6 or above), GNU make (sometimes installed as *gmake*,and sometimes as *make*), and a FORTRAN compiler. To find out the F90 dependencies, we use the *makedepf90* program. It is available at http://personal.inet.fi/private/erikedelmann/makedepf90/. Make sure that it is on your $PATH.

5.2.2. Machine, expert, and compiler rcfiles

There is a set of rcfiles with advanced settings that users rarely modified. They mostly deal with hardware and software specifics, and are found in the 'rc' sub-directory.

All rcfiles that need to be ported from the TM5-zoom still have the "pycasso-" prefix. Make the needed modifications, following files that are already ported and the tips hereafter, and rename them by taking out the "pycasso-" prefix.

As of this writing, files for the following environment have been ported:

- 'neuron', a BULL computer at KNMI (Intel compiler)
- 'cca', a CRAY XC40 at ECMWF (CRAY, Intel, and GNU FORTRAN compilers)
- 'lokhi', a blade at KNMI (gfortran compiler)
- noaa
- surfSARA (NL)
- Wageningen (NL)
- TNO (NL)

5.2.3. Tip for chem-input rcfile

By putting all the emissions inventories under the same common directory you just have to specify:

```
topdir.dynamic  : /path/to/parent/dir/of/emissions/inventories
```

in your *chem.input.rc*. This will automatically set the path to the various inventories:

```
input.emis.dir.ED41
```

```
input.emis.dir.gfed
input.emis.dir.MACC
input.emis.dir.MEGAN
input.emis.dir.AR5
```

5.2.4. Tips for machine file

To modify a machine file from TM5-zoom, there are three (3) potential changes to look for:

- **my.meteo.dir** has been removed (it is now in the main rc file). This was motivated by the fact that several users on the same machine were using a totally different runtime meteo dir, and we want to keep one machine file per machine.
- check path in **#include** statements, if any (typically to a compiler or queue rc file)
- libraries: udunits 1.x (udunits for FORTRAN) and GRIBEX are not maintained anymore, and become more and more difficult to install on recent machines/compilers. We added an interface to udunits v2.x (udunits for C), and to the grib_api.
    - **grib_api** has been added, and still coexists gribex. This is needed for those who process meteo in grib format.
    - udunits is obsolete, and should be replace by **udunits1** or **udunits2**, depending on which version is available on your system. This is needed if you use the netcdf met fields.

5.2.5. Tips for job manager

The job manager rcfiles have not changed from TM5-zoom. If yours is not available, you need to add it to the "bin/submit_tm5_tools.py" script. Beware of qsub: there are several of them around, and not all are handled in the script.

5.3. Differences with TM5-zoom

The basic tree structure with a base, levels, and proj dirs, the later with a collection of project dirs, will be familiar to users of TM5-zoom. However there are few changes in the user interface, which are listed here.

5.3.1. Moved keys

- move **region.rc** out of expert.rc into the main.rc. Motivation: the reduced grid definition may differ and user should be aware of it. For example, there are two versions of the region-glb100x100.rc with two totally different reduced grids.
- move **meteo dir** (too much user dependent) into main.rc, out of machine.rc
- move **my.source.dirs** to expert.rc. In your main.rc, you just provide the list of proj for the **my.source.proj** key. key. See the templates. This simplification is feasible, because we now have only one base and automatic selection of levels code.
- move a set of keys **(jobstep, jobstep.timerange.start, jobstep.timerange.end, prev.output.dir)** into expert rcfile: no reason to keep those in the main rcfile, since they are setup by the script.
- move **par.ntask** at the end of the main rcfile. It is now set by the script, but must stay in that file.

- **par.openmp** and **par.nthread** have moved to the expert rcfile. OpenMP is not used yet in TM5-MP.

5.3.2. New keys in the main rcfile

- **par.nx**, **par.ny**: number of cores along longitudes and latitudes. You should set the maximum of cores in the latitudinal direction first (*par.ny*). Then try to increase *par.nx*. For 3x2 runs, you should use *par.ny:45* and *par.nx:1* or *2*.
- **my.source.proj** : list of project directories with code to gather.
- **convective.fluxes.tiedtke** : to switch between EI convective fluxes and those computed from Tiedtke scheme. It is needed to correctly scale the Lightning NOx emissions. It should be set to F for using ERA-interim convective fluxes. It has no effect (ignored) if you use OD met fields.
- **write.column.budgets** : T by default, so this is a key to switch off some output, namely the Non-Horizontally-Aggregated-Budgets. Most budgets are computed for spatial bins that encompass several grid boxes, vertically and horizontally. The definition of these regions is done in *budget.F90* and there are currently 2 versions of this file with different budget regions definition. Some budgets are not aggregated, and end up using a lot of disk space. It is possible to skip writing of these budgets by setting this key to F. If not present or empty it defaults to T. The differentiation is only on the horizontal binning, since the vertical binning is the same for all budgets. In other words:

```
(if true (default), writes [im,jm,nbud_vg] budgets)
```

5.3.3. Key with new behavior

- **output.pdump.vmr.???.dhour**: time step of the chemistry time series can be less than an hour. It is now entered as fractional hour (0.5 for example), and converted to int(frac_hour*3600).

5.3.4. Compilation through the job manager

It is now possible to submit the compilation job through the job manager. This assumes bash is available on your machine, and requires few keys in your machine rcfile. The feature was required on cca @ECMWF. So, if you need something similar, look into its corresponding machine rcfile.

5.3.5. New macros (cpp)

- tropomi : to use different format/conventions for timeseries output in the modified CB05 chemistry project.
- with_parallel_io_meteo : use to read netCDF meteo in parallel
- with grib_api : to use grib_api instead of gribex. Used only in pre-processing of the original meteo fields from MARS archive.
- with_udunits1 & with_udunits2 : replace with_udunits and provide choice for udunits lib.

5.3.6. Lightning NOx emissions

Total has been added to the budget file, and is not printed anymore into the log file.

5.3.7. New libraries

Switch to udunits v2 and grib_api in place of udunits v1 and gribex. See new macros.

6. Compile and/or run: the Pycasso wrapper

6.1. Basics

The pycasso script for TM5 is linked into the top directory as setup_tm5. It lets you:

1. gather and compile the code source,
2. setup a run, and
3. submit a run.

The call is:

```
./setup_tm5 [options] my.rc
```

You can see the list of available options with -h:

```
./setup_tm5 -h
```

6.2. Gather and compile code

TM5 is modular. The main consequence of this modularity is that you cannot compile the code in place. The first step of the setup script consists in gathering the code from the */base* and (optionally) additional projects, which are below the */proj* dir, into one location, the "my.project.dir/build" dir. The later is created if needed, and "my.project.dir" is defined in your rc file. Gathering the code and compiling it are done by default.

Note that nothing prevents you to use the checkout dir as "my.project.dir".

There are three rules when building the code source:

1. The files in each project dir listed in the rcfile key **proj** overwrite those from the *base* (or a previously added project) if they happen to have the same name.
2. The string that follows a double underscore in the file name are removed. This is not mandatory when naming files, but a way to help identifying files provenance. For example, ebischeme__dummy.F90 becomes ebischeme.F90 before being copied to the build directory.
3. Defined in the expert rcfile, some macros remove unneeded files.

Finally, for sake of completion, there are a couple of files that are generated by the script. They are the *.h include files and the *dims_grid.F90*.

6.3. Setup and submit run

The run is also setup by default. It consists in creating a run dir and a runtime rc file, and in copying some scripts. Automatic submission of your run is determined by the value of *submit.auto* key in your rc file. It can also be triggered by the *-s* option at the command line.

A run can be submitted to the foreground, background or through a job manager. The default is set in the *expert.rc* file, and can be overwritten with command line options.

6.4. Examples

Most used commands are:

- compile and setup run (will also submit run, with default submit method, if *submit.auto* is T)

```
setup_tm5 file.rc
```

- compile, setup and submit (-s) run with the default submit method

```
setup_tm5 -s file.rc
```

- compile, setup and run through job scheduler ("submit in the queue", -sq)

```
setup_tm5 -sq file.rc
```

- recompile everything (will also submit run if *submit.auto* is T)

```
setup_tm5 -n file.rc
```

Less frequently used command:

- submit run only (without gather & compile & setup run)

```
cd rundir
submit_tm5 file.rc
```

- setup and submit run in the queue (**without** gather & compile)

```
setup_tm5 -rsq file.rc
```

- gather and compile in the queue **without** setting-up/submitting the run

```
setup_tm5 -mq file.rc
```

The last two options (-r for run, -m for make) are useful when the code is compiled through a job manager ("in the queue"), since pycasso has then no way for checking when compilation is done and successful. In other words, -r is truly a "skip compilation" option, while -m is a "skip run setup and submission".

Note that "using -r without -s" lets you setup the rundir without running, if submit.auto is F. This feature may turn out useful for debugging, since we have a way of doing "setup rundir without compiling and without running". However, since you most likely would like to have the run submitted when using -r, just set auto.submit to True in your main rc to not have to type -s all the time.

7. Developing TM

Once you have created and checked out your branch, you can start modifying and/or adding code. Such activity is part of a working cycle, which is presented first, followed by a description of each steps involved.

7.1. Working cycle

The repository consists of a *trunk*, several *branches* and several *tags*. Below each of those, you will find a "copy" of the code.

The *trunk* is the main line of development and, unlike *branches*, should be stable. It is used to gather developments from various branches, when they are deemed mature and stable and of interest for others. Only core developers maintain the trunk.

For a branch, a typical development cycle is:

- synchronize: get update from the *trunk* and/or your branch
- develop: write new code, modify old code, test, validate,…
- commit

These steps should be repeated as often as needed, and are describe in more details in the following sections. Once the steering committee agrees, a branch can be merged into the trunk. This last step is done by the core developers.

7.2. Synchronize

If there is more than one developers working with the same branch, the best way to collaborate is to commit and update frequently (and do not forget to communicate!).

```
# it is better and easier to work from the top dir, but if you only want part
# of a commit from a colleague you could navigate the tree directory. So:

cd MYNAME
svn up

# That's it. Let's finish with a tip. You can check if a colleague has
committed
# something by issuing:
svn status -u
```

You should also keep up-to-date with the *trunk*. Here are the steps to follow to sync your branch.

```
# check that you have no edit:
cd MYNAME   # it is better and easier to work from the top dir
svn status

# if you have some changes (typically indicated by a M, A, or D for Modified,
# Added, Deleted - ignore all unknown files indicated with a ?), then commit
# them before going further:
svn ci -m "message"

# always update:
svn up

# check if merging will create any conflict:
svn merge ^/trunk --dry-run

# and merge
svn merge ^/trunk

# The last commands may not work if your version of svn is somewhat old. Then
# you need to use the full repository path:
svn merge ${repo}/trunk

# You end up with uncommitted changes in your code. Test them, and commit if
# you have no problem with them (see 'Commit' section hereafter).

# Again, let's finish with a couple of tips. You can check if there is
# something in the /trunk/ that could be merged into your branch:
svn mergeinfo ^/trunk --show-revs eligible

# and try:
svn mergeinfo ^/trunk
```

7.3. Develop

So, you have checked out your branch, and wonder where you should start coding? Typically, you will work in an existing project or create a new one. Projects are sub-directories below the *proj* directory. You include them to the final code sources by listing them in your main rcfile (look for the *my.source.proj* key).

Few things to keep in mind when writing new code:

- follow the TM5 coding standards. See: TM5 coding standards
- **Remember** that the *base* is used by everybody. To keep it usable for everybody, modifications to the base should be of interest for every users. In other words, it should not break existing functionalities. If it does, which can be totally justified, it is likely that the branch will not be merged back to the *trunk* to be shared with other projects.

7.3.1. Common task #1 - traceback

- Error handling through a TRACEBACK macro: Each subroutine declares a parameter holding its name:

```
character(len=*), parameter  :: rname ='abc'
```

  for the TRACEBACK macro to work. You must declare *rname* if your routine uses IF_NOT_OK_RETURN(), defined at the top of the module. See code for examples. **Note**: the use of these macros is not possible with pure subroutine.

- For increase verbosity when setting *okdedug:T* in your main rcfile, you can add:

```
call goLabel(rname) ! at begin of subroutine
...
call goLabel()      ! on exit
```

  This code will print *<rname>* when entering a subroutine, and *(rname)* when exiting, in the log file.

7.3.2. Common task #2 - reading and scattering a global data set

The strategy is to read on one core (root) and scatter the data. The method works with any file format, and there is no penalty in using it as long as you do not read the file too often.

The following example read the 3D field 'z0' from a netcdf4 file. The data are on a 1x1 lon-lat grid, which is identified as *iglbsfc*, and *nlon360*, *nlat180* in TM5.

```
! get local bounds of 1x1
CALL get_distgrid( dgrid(iglbsfc), I_STRT=i1, I_STOP=i2, J_STRT=j1, J_STOP=j2 )

! allocate target array
ALLOCATE( z0( i1:i2, j1:j2, nz0 ) )

! open file
CALL MDF_Open( TRIM(fileName), MDF_NETCDF4, MDF_READ, FileID, status )
IF_NOTOK_RETURN(status=1)

! allocate array to read
IF (isRoot) THEN
    ALLOCATE( global_3d( nlon360, nlat180, nz0) )
ELSE
    ALLOCATE( global_3d(1,1,1) )
```

```
ENDIF

! read the data
IF (isRoot) THEN
    CALL MDF_Inq_VarID ( FileID,   'z0',    var_id, status ) ; IF_NOTOK_MDF()
    CALL MDF_Get_Var   ( FileID, var_id, global_3d, status ) ; IF_NOTOK_MDF()
ENDIF

! distribute the data
CALL SCATTER ( dgrid(iglbsfc), z0, global_3d, 0, status)
IF_NOTOK_RETURN(status=1)

! clean
DEALLOCATE( global_3d )
IF (isRoot) THEN
    CALL MDF_Close( dust_FileID, status )
    IF_NOTOK_RETURN(status=1)
ENDIF
```

Note that often there is an intermediate step, when user want the data on a different grid. Something along those lines:

```
! get local bounds of 1x1
CALL get_distgrid( dgrid(iglbsfc), I_STRT=i01, I_STOP=i02, J_STRT=j01,
J_STOP=j02 )

! get local bounds of TM5 run resolution
CALL get_distgrid( dgrid(1), I_STRT=i1, I_STOP=i2, J_STRT=j1, J_STOP=j2 )

! allocate target array
ALLOCATE( z0( i1:i2, j1:j2, nz0 ) )

! open file
CALL MDF_Open( TRIM(fileName), MDF_NETCDF4, MDF_READ, FileID, status )
IF_NOTOK_RETURN(status=1)

! allocate array to read
IF (isRoot) THEN
    ALLOCATE( global_3d_src( nlon360, nlat180, nz0) )
    ALLOCATE( global_3d_tgt( nlon, nlat, nz0) )
ELSE
    ALLOCATE( global_3d_tgt(1,1,1) )
ENDIF

! read and remap the data
IF (isRoot) THEN
    CALL MDF_Inq_VarID ( FileID,   'z0',        var_id, status )
    IF_NOTOK_MDF()
    CALL MDF_Get_Var   ( FileID, var_id, global_3d_src, status )
    IF_NOTOK_MDF()

    ! ADD REMAP FROM global_3d_src TO global_3d_tgt
    <ADD CODE HERE>
ENDIF

! distribute the regridded data
CALL SCATTER ( dgrid(1), z0, global_3d_tgt, 0, status)
IF_NOTOK_RETURN(status=1)

! clean
DEALLOCATE( global_3d_tgt )
```

```
IF (isRoot) THEN
    DEALLOCATE( global_3d_src )
    CALL MDF_Close( FileID, status )
    IF_NOTOK_RETURN(status=1)
ENDIF
```

7.3.3. Common task #3 - remapping data without gathering/scattering

This is about involving every cores into remapping a dataset. It is discussed in details in this section.

7.4. Commit

This is the easy part. Just do:

```
# it is better and easier to work from the top dir
cd MYNAME
svn ci -m "message"
```

You can do this as often as you want, since it is not required to have a stable branch. Actually it is recommended to do it often, since it makes it easier to find when bugs have been introduced, and to undo changes.

7.5. Debug

7.5.1. rc file keys

The following keys can help debugging:

- Your machine rcfile (or its compiler rcfile) should provide a debug option for the **my.build.configure.flags** key. Just use it, along the *-n* option at the command line.
- And the **okdebug**, and **go.print** keys let you increase the verbosity of the program.

7.5.2. How do you follow a box?

In tm5_tracer_data.F90, set the *ii,jj,ll,kk* at the beginning of the TRACER_PRINT subroutine to the indices of the box and tracer you want to follow. Every call to TRACER_PRINT will print the tracer mass in that box. Some of these call are already in the code within if-okdebug-then statements.

7.5.3. About bit reproducibility

The *mmix*, *restart*, and (from chemistry only) *j_stat* output should be bit-reproducible, when switching MPI layout. Budgets are not, because they are aggregated.

Currently, one subroutine cannot guarantee bit reproducibility between different MPI layout: *boundary.F90* (in *proj/cbm4* and *proj/cb05*). This is due to the way nudging factors are calculated. The order of sum will change with *par.nx*. If your test required bit reproducibility, you need to use the **without_boundary** flag.

Also, be sure to use the correct compiler flags. For example, for the CRAY compiler, "-hflex_mp=conservative" flag should be replaced with "-hflex_mp=intolerant".

7.5.4. Non-existing MPI wrapper

If you have an error message like:

```
'No specific match can be found for the generic subprogram call
"PAR_BROADCAST".'
```

It means that the interface does not cover your input type/size. You need to add a subroutine. See the many interfaces examples in *base/tm5_partools.F90*. For example:

```
INTERFACE Par_Broadcast
   MODULE PROCEDURE Par_Broadcast_i
   MODULE PROCEDURE Par_Broadcast_s
   MODULE PROCEDURE Par_Broadcast_l0
   MODULE PROCEDURE Par_Broadcast_l1
   MODULE PROCEDURE Par_Broadcast_r0
   MODULE PROCEDURE Par_Broadcast_r1
   MODULE PROCEDURE Par_Broadcast_r2
   MODULE PROCEDURE Par_Broadcast_r3
END INTERFACE
```

does not cover vector or arrays of integer, only the case of a single integer.

8. Miscellaneous

8.1. Performance

8.1.1. Lapack

It is possible to use LAPACK in TM5. Just specify the location and the library flags for linking in your machine rc file (see for an example, */rc/machine-ecmwf-cca-ifort.rc* file), and add *with_lapack* to the *my.defs_misc* key in your main rc file. LAPACK is used in the convection scheme, and can speed up that part of the code by 10%, although overall speed up is a lot less.

8.2. Output size

- W/o the restart files, @3x2-34L, with the current timeseries required for the benchmarking of CB05, we got around 50G of data for one year of output.
- Restart files are 820M, so 12 of them gives: 9.6G.
- total=60G/y
- At 1x1: 6 times more, ie 360G

8.3. Remapping and number of cores

Remapping data between different grids is done in several places in the code. It is usually performed on a single core to remap an input file, like emissions or a restart file with *istart=32*. Once remapped to the good resolution, the dataset is scattered. However, it is possible that the data are already scattered on the source grid, and you do not want to gather-remap-scatter but instead would like to just remap. It would save a lot of MPI communication, and make the code more scalable, particularly if you have to often repeat the operation during a run. This so-called on-core remapping can be easily implemented. The method has one (small) drawback: it limits the number of cores you can use, although not the maximum (see below). Note that the run will stop if the restrictions on the core number are not fulfilled. You will get an error message like:

```
ERROR - do not know how to relate east bounds:
ERROR -   lli  bound, spacing, number : -126.0000    3.0000    18
ERROR -   lliX bound, spacing, number : -128.0000    1.0000    52
ERROR in grid_type_ll/Relate
ERROR in grid_type_ll/FillGrid
```

The TM5-MP code already does on-core remapping when reading netCDF meteo in parallel at a different-from-simulation resolution. It is examined first to show what are the limitations. Directions to apply something similar in your code is then presented. We finish with the specific core limitation for reading restart files with istart=32, which are due to the reading itself and not the remapping.

8.3.1. Reading netCDF meteo in parallel

When reading the netCDF met fields in parallel, and the meteo and the run are at different resolutions, a remapping is performed on each core assuming that the data on the other cores are not needed. In other words, the MPI decomposition for both resolutions must share boundaries. This makes the remapping efficient, but the number-of-boxes-per-core (which is different for each resolution) should be multiple of each other, and implies (by design) that the grid boxes be evenly distributed in both resolutions.

1. Example: reading meteo at 1x1 for runs at 3x2.

   For a run at 3x2, you can use up to 45 cores in the latitudinal direction. Normally any number between 2 and 45 would work, but if you read the meteo at 1x1, you must evenly distribute the grid boxes. Evenly decompositions at 3x2 can be obtained from prime numbers:

   ```
   120 long. boxes = 2x2x2x3x5
   90 lat. boxes = 2x3x3x5
   ```

   we are then limited to these numbers of core in each direction:

   ```
   NX = 2, 3, 4, 5, 6,  8, 10, 12, 15, 20, 24, 30, 40 or 60
   NY = 2, 3, 5, 6, 9, 10, 15, 18, 30, or 45
   ```

   which also divide 360 and 180, the number of boxes at 1x1 in each direction. Since the model does not scale very well in the NX direction, the limit is on NY: you cannot use any number between 31 and 44 for example. But you can still use 45 cores!

8.3.2. Implementing on-core remapping

The method is straightforward, and build around these few calls for a remapping between the run resolution (region=1) and 1x1 (region=iglbsfc):

```
USE METEODATA, ONLY : lli

! get local bounds of 1x1
CALL get_distgrid( dgrid(iglbsfc), I_STRT=i1, I_STOP=i2, J_STRT=j1, J_STOP=j2 )
allocate( Arr1x1( i1:i2, j1:j2 ) )  ! no halo

! get local bounds of TM5 run resolution
CALL get_distgrid( dgrid(1), I_STRT=i1, I_STOP=i2, J_STRT=j1, J_STOP=j2 )
allocate( Arr3x2( i1:i2, j1:j2 ) )  ! no halo

! ...do some work with Arr1x1...

! remap Arr1x1 into Arr3x2
! Replace 'sum' with 'area-average' or 'average', depending on the nature
! of the data
call fillgrid( lli(1), 'n', Arr3x2, lli(igblsfc), 'n', Arr1x1, 'sum', rc)
IF_ERROR_RETURN(status=1)

! check rc
if (status==-1)then
    write(gol,*) "Only part of target array was filled!"; call goErr
    write(gol,*) "Make sure that the number of processors"; call goErr
    write(gol,*) "divides the number of grid boxes."; call goErr
    TRACEBACK; status=1; return
endif
```

Ideally you want to check if the run is not done at 1x1 to avoid unneeded work. This can be achieved with:

```
if (igblsfc==1) then ...
```

8.3.3. Restart with istart=32

When reading a restart, all cores are involved. Since the maximum number of cores you can use depends on the resolution, you are limited by the lower resolution: restart or run. More specifically, the input data resolution (number of boxes) and the number of cores should fulfill the following requirement, in both longitude and latitude directions:

```
nb-of-boxes/nb-of-cores > max(1,halo)
```

When remapping a restart file at a resolution coarser than the model resolution, this constraint decreases the maximum number of cores you can use. For example, reading a 3x2 or 6x4 restart file for a 1x1 run, you have:

- run at 1x1: max number of cores is 180x90 (halo=2)
- restart at 3x2: max number of cores is 120x90 (halo=0)
- restart at 6x4: max number of cores is 60x45 (halo=0)

If these limits are a problem, you can make a very short run (3hr) that writes an extra restart at time step 0.

9. Frequently Asked Questions

9.1. How do I run my script before or after a run?

There are pre- and post-processing sections in the main rc file:

```
01  !=====================================================!
02  ! PRE-Processing
03  !=====================================================!
04  ...
05  ! II - User scripts to setup run
06  input.user.scripts   : <bindir>/tm5-tmm-setup <rcfile>
07
08  !=====================================================!
09  ! POST-Processing
10  !=====================================================!
11  ...
12  ! III - Run user scripts
13  output.user.scripts.condition : True
14  output.user.scripts           : /path/to/my/script
```

Line 6 is an example of a pre-prosssing script to submit before the run. It refers to two special variables <*bindir*> and <*rcfile*>, that are expanded as *../bin* and the current rc file name by the python script. See next Frequently Asked Questions for how useful that can be.

Line 13 is a conditional test to run the script(s) on line 14. Set to *True*, the script is run at the end of ever leg. But to run you script only at the end of the last leg, you would use the following condition (two lines for readability, but should be on one line):

```
output.user.scripts.condition : \
        "%{jobstep.timerange.end}" == "%{timerange.end}"
```

There is no conditional test for pre-processing, since you can run you script outside the TM5 workflow if needed only once.

Note that these scripts are submitted to the queue requesting a single core, i.e. they are serial jobs. It is possible:

- to pass the rc file as an argument to the script, so all keys are available in your script
- to use intermediate key for readability, if the path to your script is quite long for example
- to submit more than one script (use ";")

Some of these features are illustrated in this example:

```
your.script        : /scratch/rundir/get_some_data
my.script          : /scratch/rundir/get_my_data
input.user.scripts : ${my.script} <rcfile> ; ${your.script}
```

9.2. How do I read one of the rc file key in my python or bash script?

9.2.1. case of sh or bash script

You can read any of your rcfile keys value by calling the *go_readrc* script available in the *bin* subdirectory. You can see example calls in the *tm5-tmm-store* script. You just need to pass the rc file path to your script. If your script is triggered through the rc file in post- or pre-processing you can refer to it with <*rcfile*>, and to the *bin* subdirectory with <*bin*>.

9.2.2. case of python script

With a python script you can simple use the *bin/rc.py* module. See its header doc string for how-to.

9.3. What is the content of the MMIX output files exactly?

Ans.: The *mmix* files contains monthly averages of volume mixing ratio, temperature and pressure. In the global attributes, you will find:

- names : names of the species in the order they are in the *fscale* and *ra* arrays
- fscale : scaling factor for conversion of mixing ratios in kg of tracer per kg of air to practical mixing ratio units (e.g. ppm), the later being saved in the mmix file. Fscale is just the ratio of molecular weight of air to that of tracer: mw(air)/mw(tracer)
- ra : molecular weight of species (See *chem_param.inc* and *chem_param.F90* files for details)
- at, bt : A's and B's hybrid coefficients to compute level pressures.

So the following relation hold:

```
mass_tracer * fscale = mixing_ratio * mass_of_air
```

9.4. How do I make a meteo field available on 1x1?

Q1: I want to emit my biological particles depending on specific humidity on the 1x1 degree grid. However, *humid_dat* seems to live only on the coarser 3x2 grid. Is there a way to have this data on the 1x1 grid?

A1: Yes. You just need to tell TM5 to used it with a call like this one:

```
call Set( u10m_dat(iglbsfc), status, used=.true. )
```

The iglbsfc is the 1x1 grid. This example for u10 is in several places, you do the the same with *humid_dat* in one of the init routine (normally the one in the module where you will use the data) and you are all set.

9.5. Which part of the code ensures global mass conservation?

Q1: It is my impression that when TM5 reads in era interim wind fields and produces the mfu/mfv/mfw fields, it ensures global airmass conservation. Could you point me to the part of the code that does that?

A1: the mass balancing is done in:

```
TM5/base/trunk/src/meteo.F90
```

in routine:

```
Meteo_Setup_Mass
```

by a call to:

```
BalanceMassFluxes
```

The input (unbalanced) mass fluxes are: *mfu, mfv, mfw* which are copied to new arrays: *pu, pv, pw* from these, *pu/pv* are then changed to ensure mass conservation.

Q2: Is this the routine that makes sure the global air mass is constant?

A2: Global mass conservation is ensured by scaling the global surface pressure fields (*sp1_dat* and *sp2_dat*) with a single-cell global surface pressure field *sp_region0*. Search for the later in meteo.F90

10. Known Issues

We try to keep track of issues in the development portal here. Additionally here are some known problems.

10.1. Interpolating model at observation location

Let say the domain on one processor is indicated by the blue area. When needed, information from the neighbouring processors (in yellow) is fetched. This is typically done in the code with call to *update_halo* functions, and is useful for transport and some interpolations. However, if your interpolation scheme requires one of the "corner neighbors" *C*, it will not work, and you must either change interpolation method, or interpolate off-line in a post-processing step.