LMDZ tutorial: XIOS

LMDZ team

December 10-12, 2024

This tutorial focuses on setting up, compiling and running LMDZ with XIOS.

This document can be downloaded as a pdf file:

```
wget http://lmdz.lmd.jussieu.fr/pub/Training/Tutorials/Tutorial_XIOS.pdf
```

which should ease any copy/paste of command lines to issue.

This tutorial is for users who want to learn the basic steps needed to be able to run LMDZ with the XIOS input-output library on their computer. Note that this implies the prerequisite that you can run in parallel on your machine (i.e. that you have already completed the tutorial on running LMDZ in parallel).

1 Running the install_lmdz.sh script

The **install_lmdz.sh** script can download and compile the necessary libraries (NetCDF, IOIPSL, XIOS) and programs (ORCHIDEE and LMDZ), and runs a test simulation. All that is required is to specify the `-xios` and `-parallel mpi_omp` (XIOS is designed to be used in parallel) options:

```
wget http://lmdz.lmd.jussieu.fr/pub/install_lmdz.sh
chmod +x install_lmdz.sh
./install_lmdz.sh -name LMDZORXIOS -parallel mpi_omp -xios -veget CMIP6 -rad oldrad
```

Note that, just as in the other tutorials, if you are using a not too recent gfortran compiler (e.g. version 9 or less) then you should add `-arch local-gfortran9-parallel -arch_dir $PWD/arch` to the `install_lmdz.sh` script flags.

As with previous automated installations, you are encouraged to browse through the contents of sub-directory **LMDZ** (e.g. the **compile.sh** script) and **BENCH32x32x39**.

Note that to compile LMDZ with XIOS, one must use the `-io xios` flag, e.g.:

```
./makelmdz_fcm -arch local-gfortran-parallel -parallel mpi_omp -io xios -d 32x32x39 \
-j 8 gcm
```

As can be seen in the **compile.sh** script in the **LMDZ** directory.

Take the time to browse through the ***.xml** and **histday.nc** files[1] in **BENCH32x32x39** to identify definitions and settings that were used. Note that the automated processing in the install script has renamed the executable **gcm.e**, but you may still find it in **LMDZ/bin**.

2 Running a first simulation with XIOS

Make a new simulation directory (e.g. by copying over all input **.def** and **.nc** files from the **BENCH32x32x39** directory[2]), along with the executable `gcm_32x32x39_phylmd_oldrad_para_mem_orch.e` from **LMDZ/bin**. In addition you will need the input **.xml** files to manage XIOS outputs, which you should also copy over from **BENCH32x32x39**.

[1] Note that there is also a **histmth.nc** file that is generated but that it is empty, as the run is too short compared to the 1 month output rate for that output file

[2] As in this example ORCHIDEE will not used, the flag **VEGET** in **config.def** should be set to **n**

For this first simulation, we will use XIOS in attached mode (i.e. embedded in LMDZ), so the **using_server** variable in **iodef.xml** must be set to **false** (which should already be the case in the current setup).
Moreover, to enable outputs via XIOS in LMDZ, note that the following flag:

```
ok_all_xml = .true.
```

(or equivalently `ok_all_xml = y`) must be set in the **run.def** or **config.def** file.
As we will not be using ORCHIDEE (which also uses XIOS for its outputs) you should also make sure you have

```
VEGET = n
```

in **config.def**. Moreover you can also remove (or simply ignore) references to ***_orchidee.xml** files, e.g. by commenting out (or deleting) line

```
<context id="orchidee" src="./context_orchidee.xml"/>
```

from the **iodef.xml** file.
As can be seen in the **context_lmdz.xml** file, many predefined output files (mimicking what is done via the output.def file when using IOIPSL) are defined via the inclusion of `file_def_*_lmdz.xml` files. By default a few output files are enabled. You can check this by checking the value of the **enabled** field of the various `file_def_*_lmdz.xml` files. For instance by default **histday.nc** is enabled since in file **file_def_histday_lmdz.xml** we have:

```
<file id="histday" name="histday" output_freq="1d" output_level="5"
      type="one_file" enabled=".TRUE." compression_level="0" sync_freq="1d">
```

Let's imagine that instead we want output file **histhf.nc** to be outputed. Modify file **file_def_histhf_lmdz.xml** by setting

```
<file id="histhf" name="histhf" output_freq="3h" output_level="5"
type="one_file" enabled=".true." compression_level="0" sync_freq="3h" >
```

so that the **histhf.nc** file will also be generated (as a single file over the entire domain) when the model is run.

Then run the model "as usual", e.g. in MPI mode using 4 processes[3](without OpenMP here):

```
mpirun -np 4 gcm_32x32x39_phylmd_oldrad_para_mem_orch.e > listing 2>&1
```

And check the contents of the generated **histday.nc** and **histhf.nc** files. Note the use of `operation="average"`, `operation="instant"`, `operation="once"` , etc. attributes in the xml files to trigger designated operations on the outputted fields.

3 Exercise: Changing XIOS output files and verbosity level

Parameters specific to dictate XIOS behavior are defined in the **iodef.xml** file within the XIOS context:

```
<context  id="xios">
            <variable_definition>
                <!-- set print_file to true to have XIOS write its
                     outputs in separate files -->
                <variable id="print_file" type="bool"> false </variable>
....
                <variable_group id="parameters" >
                    <!-- set using_server to true to run with XIOS
                         in detached server/client mode -->
                    <variable id="using_server" type="bool">false</variable>
                    <!-- info_level is the verbosity level of XIOS
                         0 for no messages; 100 for lots -->
                    <variable id="info_level" type="int">100</variable>
                </variable_group>
            </variable_definition>
</context>
```

[3]Or using a script such as **bench.sh** or **bench_parallel.sh**, which you might need to adapt to your settings

The **info_level** variable sets the degree of verbosity of XIOS (0: low, 100: high). There are other additionnal variables than the one present in the current **iodef.xml**, for instance **print_file**, which can be set to true to have XIOS issue its output and error messages to dedicated text files (one of each for each MPI process), or false (default behavious) to send these messages to the same standard output and error streams as LMDZ. When debugging, another useful variable is **xios_stack** (true/false; default value is false) to have a more complete traceback or encountered errors.

Assume you want to rerun the previous experiment with higher XIOS verbosity and dedicated output files; adapt the **iodef.xml** and rerun to check the generated **xios_client_*.err** and **xios_client_*.out** files.

4 Exercise: Running in client-server mode

When running on a small number of cores, it is advised to use XIOS in "attached" mode, as done so far. In multicore environments (i.e. more than 32), when outputing a lot of data, it can be more efficient to run in client-server mode and dedicate some cores to the XIOS server.

To test this setup, make a new directory where to run and copy over input files from previous simulation. Start by copying over the XIOS server **xios_server.exe** from **XIOS/bin**. Then adapt the **iodef.xml** to switch to client-server mode by setting the **using_server** variable to **true**. The executables may now be run, where the number of processes allocated to each is set via the **mpirun** command, for instance to run LMDZ on 3 processes and XIOS on 1:

```
mpirun -np 3 gcm_32x32x39_phylmd_oldrad_para_mem_orch.e > listing 2>&1 : -np 1 xios_server.exe
```

And check that you get the same output files as before.

5 Exercise: Defining an additional output domain and grid

One can output only a selected subset of the global domain[4] by specifying the appropriate **domain** attributes in the **context_lmdz.xml** file. For example to ontput a 2x3 subdomain starting at (rectilinear) grid indexes $i = 20$,$j = 15$ (C convention: index begining at 0):

```
<domain_definition>
    <domain id="dom_glo" data_dim="1" />
    <domain id="dom_glo_rect" ni_glo="32" nj_glo="33" type="rectilinear" >
      <generate_rectilinear_domain lat_start="90" lat_end="-90" lon_start="-180"/>
      <interpolate_domain order="1"/>
    </domain>
    <domain id="domain_zoom" domain_ref="dom_glo_rect">
      <zoom_domain ni="2" ibegin="20" nj="3" jbegin="15" />
    </domain>
....
</domain_definition>
```

And also, still in the **context_lmdz.xml** file, define the corresponding grid in the `<grid_definition>` section:

```
  <grid_definition>
....
     <grid id="grid_glo">
        <domain domain_ref="dom_glo" />
     </grid>
     <grid id="grid_out">
        <domain domain_ref="dom_out" />
     </grid>

    <grid id="grid_zoom" >
      <domain domain_ref="domain_zoom" />
```

[4]Some XIOS terminology: a "domain" refers to a horizontal (2D) domain, an "axis" is 1D (e.g. altitude, or number of tracers) and a "grid" is a (0D, 1D, 2D or 3D) geometry along which outputs will be made.

```
    </grid>
     <grid id="grid_zoom_presnivs" >
        <domain domain_ref="domain_zoom" />
        <axis axis_ref="presnivs" />
     </grid>
 ....
```

To test implementing this setup, let's assume you want to output at only one grid point, corresponding to Paris to compare model ouput to station records.

The first thing to do is to identify the grid coordinates that will have to be specified in the **domain** attributes. This can be done by inspecting the **lat** and **lon** values in the **histday.nc** file from the previous run, either via your favorite visualization software, or simply using the **ncdump** utility:

```
ncdump -fc -v lon histday.nc
ncdump -fc -v lat histday.nc
```

And adapt the **context_lmdz.xml** file accordingly.
Since we are interested in instantaneous values of for instance the **t2m** (temperature at 2m), **precip** (precipitation rates), **psol** (surface pressure) and **temp** (temperature profile) in the zoomed grid, it makes sense to define a new output file. One could either adapt the current **histins** file, or define a new one e.g. a **file_def_histinsParis_lmdz.xml** file:

```
<file_definition>
    <file_group id="defile">
        <file id="histinsParis" name="histinsParis"
               output_freq="1ts" output_level="5"
               type="one_file" enabled=".TRUE.">

            <!-- VARS 2D -->
            <field_group operation="instant" freq_op="1ts"
                         grid_ref="grid_zoom" >
                <field field_ref="t2m" level="5" />
                <field field_ref="t2m_min" level="5" operation="minimum" />
                <field field_ref="t2m_max" level="5" operation="maximum" />
                <field field_ref="precip" level="1" />
                <field field_ref="psol" level="5" />
            </field_group>

            <!-- VARS 3D -->
            <field_group operation="instant" freq_op="1ts"
                         grid_ref="grid_zoom_presnivs">
                <field field_ref="temp" level="4" />
            </field_group>
        </file>
    </file_group>
</file_definition>
```

And add this new definition file to the others specified in **context_lmdz.xml**:

```
    <file_definition src="./file_def_histinsParis_lmdz.xml"/>
```

Run the model and check the produced **histinsParis.nc** file.