Tutorial: Using LMD with XIOS

The LMDZ team

December 6, 2015

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

This document can be downloaded as a pdf file:

```
wget http://www.lmd.jussieu.fr/~lmdz/Distrib/TD_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 Installing the required HDF5 and NetCDF4 libraries

The XIOS library relies on HDF5 and NetCDF4 libraries (which also rely on curl and zlib libraries) that have to be compiled with the same compiler than the MPI library that will be used.

We have a script that combines all the download and install instructions, which you can download using:

```
wget http://www.lmd.jussieu.fr/~lmdz/Distrib/install_netcdf4_hdf5_mpi.bash
```

You should then make that script executable and run it

```
chmod u=rwx install_netcdf4_hdf5_mpi.bash
./install_netcdf4_hdf5_mpi.bash > install_netcdf4_hdf5_mpi.out 2>&1
```

If all goes well, then all the required libraries will be generated under **$HOME/local** (on the provided laptops this will be **/home/util1/local**). This default is set in the script as **INSTALL_PATH** and can be changed; in all that follows it is assumed that the libraries have been installed under **$HOME/local**

2 Initial setup and recompiling the IOIPSL library

Because of the change in used NetCDF libraries, it is recommended that you work on a copy of the modipsl directory you have used so far. Copy over the entire **LMDZtrunk** directory and remove subdirectories **LMDZtrunk/netcdf-4.0.1**, as well as the **ORCHIDEE** and **ORCHIDEE_beton** from the **modeles** subdirectory (this is only to keep things simple in this tutorial and avoid having to also recompile Orchidee).

To rebuild the IOIPSL library, you must first go to the **modipsl/util** directory and adapt the **AA_make.gdef** file to point NetCDF library. Replace lines

```
#-Q- g95      NCDF_INC= ...
#-Q- g95      NCDF_LIB= ...
```

With (again we assume here the NetCDF library is installed in **/home/util1/local**; adapt accordingly if it is not the case):

```
#-Q- g95      NCDF_INC= /home/util1/local/include
#-Q- g95      NCDF_LIB= -L/home/util1/local/lib -lnetcdff
```

And then issue the following command:

```
./ins_make -t g95 -p I4R8
```

This will generate adequate Makefile files in **IOIPSL/tools** and **IOIPSL/src**.

To recompile the IOISPL library, go to the **IOIPSL/tools** subdirectory and modify the **Makefile** to set the **SHELL** variable to:

```
SHELL = /bin/bash
```

Note that this is mandatory, unless the (default) ksh shell is installed on your system. And then build the library using the **make** command

```
make
```

This will compile the sources and put the generated library **libioipsl.a** in directory **../../../lib**, i.e. in the **lib** directory under **modipsl**. You have to also manualy move all the generated module files to that same directory:

```
mv *.mod ../../../lib
```

3 Installing the XIOS library

First you must download the XIOS library sources, using svn, in your **modipsl/modeles** directory:

```
svn co http://forge.ipsl.jussieu.fr/ioserver/svn/XIOS/branchs/xios-1.0 XIOS
```

Then go to the **XIOS/arch** directory and create the following files:
arch-local.env:

```
export MPI_LIB="-L/usr/lib/openmpi/lib -lmpi"
export NETCDF="$HOME/local"
export NETCDFLIB=$NETCDF/lib
export NETCDFINC=$NETCDF/include
```

arch-local.path:

```
NETCDF_INCDIR="-I${NETCDFINC} "
NETCDF_LIBDIR="-L${NETCDFLIB} "
NETCDF_LIB="-lnetcdf -lnetcdff -lnetcdf_c++4"
HDF5_INCDIR="-I${NETCDFINC} "
HDF5_LIBDIR="-L${NETCDFLIB} "
```

arch-local.fcm:

```
%CCOMPILER      mpicc
%FCOMPILER      mpif90
%LINKER         mpif90
%BASE_CFLAGS    -w
%PROD_CFLAGS    -O3 -D BOOST_DISABLE_ASSERTS
%DEV_CFLAGS     -g -O2
%DEBUG_CFLAGS   -g -DBZ_DEBUG
%BASE_FFLAGS    -D__NONE__
%PROD_FFLAGS    -O3
%DEV_FFLAGS     -g -O2
%DEBUG_FFLAGS   -g
%BASE_INC       -D__NONE__
%BASE_LD        -lstdc++
%CPP            mpicc -EP
%FPP            cpp -P
%MAKE           make
```

Once these three files have been created, you can build the XIOS library. In directory **XIOS**, run the **make_xios** script as follows:

```
./make_xios --debug --arch local --job 8 > make_xios.out 2>&1
```

If all went well, the **XIOS/lib** directory should contain the XIOS library, **libxios.a**, and the **XIOS/bin** directory should contain (among other test programs) the XIOS server **xios_server.exe**.

4 Compiling LMDZ with XIOS

Return to the **LMDZ5** directory and add the following two lines to **arch/arch-para.path**:

```
XIOS_INCDIR=$LMDGCM/../XIOS/inc
XIOS_LIBDIR=$LMDGCM/../XIOS/lib
```

You also have to adapt the **NETCDF_LIBDIR** and **NETCDF_INCDIR** in **arch/arch-para.path**:

```
NETCDF_LIBDIR="-L/home/util1/local/lib -lnetcdff -lnetcdf -lnetcdfc++4 -lhdf5hl_fortran
-l hdf5_hl -lhdf5_fortran -lhdf5 -lz -lstdc++"
NETCDF_INCDIR="-I/home/util1/local/include"
```

You can now compile LMDZ5 as usual, but adding the "-io xios" flag, e.g.

```
./makelmdz_fcm -arch para -mem -parallel mpi -io xios -d 32x32x39 -j 8 gcm
```

5 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[1]). In addition you will need the input **.xml** files to manage XIOS outputs. Examples of these can be found in the **LMDZ5/Deflists** directory. Copy over files **iodef.xml**, **context_lmdz.xml**, **field_def_lmdz.xml** and **file_def_hist*.xml**.

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**.
Moreover, to enable outputs via XIOS in LMDZ, the following flag:

```
ok_all_xml = y
```

[1]As in this example Orchidee is not used, the flag **VEGET** in **config.def** should be set to **n**

must be set in the **run.def** file (or equivalently in the **config.def**.

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. By default none are enabled. Modify file **file_def_histday_lmdz.xml** by setting

```
<file id="histday" name="histday" output_freq="1d" output_level="2"
type="one_file" enabled=".true.">
```

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

The model can now be run. Note that because the NetCDF libraries are dynamically loaded at run time, the path to these must be included in the **LD_LIBRARY_PATH** environment variable[2]:

```
export LD_LIBRARY_PATH=\home\util1\local\lib:$LD_LIBRARY_PATH
```

Then run the model "as usual", e.g. in MPI mode using 4 processes:

```
mpirun -np 4 gcm_32x32x39_phylmd_para_mem.e > listing 2>&1
```

And check the contents of the generated **histday.nc** file.

6 Defining the output domain

One can ouput only a selected subset of the global domain by specifying the appropriate **domain** attributes in the **context_lmdz.xml** file. For example to ontput a 2x3 subomain starting at grid indexes $i = 14$,$j = 5$:

```
<domain id="dom_glo" data_dim="2" zoom_ni="2" zoom_ibegin="14"
zoom_nj="3" zoom_jbegin="5" />
```

To test implementing this setup, let's assume you want to output at only one grid point, corresponding to Paris (longitude 48N ,latitude 2E) to compare model ouput to station records.

The first thing to do is to identify the global 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 -ff -v histday.nc
```

And adapt the **context_lmdz.xml** file accordingly.
Since we are interested in instantaneous values, it makes sense to enable the **histins** file, so adapt the **file_def_histins_lmdz.xml** so that an **histins_Paris.nc** file (i.e.: add the **enabled=.true.** and **name="histins_Paris.nc"** in the **file** attributes) will be generated.
Also adapt general **output_level** and/or individual **level** in **file_def_histins_lmdz.xml** so that **histins_Paris.nc** will contain **t2m** (temperature at 2m), **precip** (precipitation rates), **psol** (surface pressure) and **temp** (temperature profile).
Run the model and check the produced files.

[2]Rather than having to have to add the path to **LD_LIBRARY_PATH** in each new shell before running the model, it makes sense to add that command in your **$HOME/.bashrc** file so that it is always automatically done.

7 Running in client-server mode

When running on a small number of cores, it is advised to use XIOS in "attached" mode. In multicore environments (i.e. more than 32) 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_para_mem.e > listing 2>&1 : -np 1 xios_server.exe
```

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