

# LMDZ tutorial: Parallelism

LMDZ team

January 9-11, 2024

This tutorial focuses on setting up, compiling and running LMDZ in a parallel environment.

This document can be downloaded as a pdf file:

```
wget http://lmdz.lmd.jussieu.fr/pub/Training/Tutorials/Tutorial_Parallel.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 parallel versions of LMDZ on their computer. Note that this implies the prerequisite that there is a working MPI library already installed on the machine, which is the case for the laptops provided for this training session (on which the MPICH library is installed; the OpenMPI library is also known to work).

Throughout this tutorial we will assume that you are working on the provided laptops, for which an MPI library and utilities (using the gfortran compiler) are installed in the following directories<sup>1</sup>:

```
/usr/bin /usr/lib /usr/lib/openmpi
```

Depending on your environment variables and settings<sup>2</sup>, it is recommended (often mandatory!) to set available stacksize (which is roughly the amount of memory your program is allowed to request from the system) to maximum and add the MPI library path to **LD\_LIBRARY\_PATH** (this variable tells programs where to look for needed dynamical libraries at run time):

```
ulimit -s unlimited
export LD_LIBRARY_PATH=/where/your/mpi/lib/is/lib:$LD_LIBRARY_PATH
```

Note that for the provided laptops there is no need to modify **LD\_LIBRARY\_PATH** since the library is in a common standard location.

## 1 Running the `install_lmdz.sh` script

The `install_lmdz.sh` script can do the necessary to install the model; all that is required is to use the **-parallel** option, which may be set to **none** (default), **mpi** to compile and run in MPI only, **omp** to compile and run in OpenMP only, or **mpi\_omp** to compile and run the hybrid (mixed MPI and Openmp) parallel version of LMDZ.

Create a separate directory to install the model (assuming you want to keep previous work done with the serial version used in tutorial #1) and run<sup>3</sup>:

```
wget http://lmdz.lmd.jussieu.fr/pub/install_lmdz.sh
chmod +x install_lmdz.sh
./install_lmdz.sh -name LMDZORpar -parallel mpi_omp -veget CMIP6 -rad oldrad \
-d 32x32x39 -netcdf 'pwd'/LMDZTraining
```

Just as in tutorial #1, the script will end after a short run of the 32×32-L39 test case has run in parallel.

---

<sup>1</sup>If running on your own machines, you will of course have to adapt MPI paths to point to the appropriate location. There are various available MPI libraries downloadable from the Internet, such as OpenMPI or MPICH. What is important to know is that the MPI library must have been compiled with the same compiler than the one used to compile LMDZ.

<sup>2</sup>Ideally setting stacksize and **LD\_LIBRARY\_PATH** should be set in your `~/.bashrc` to avoid having issue these commands in each terminal and in every session.

<sup>3</sup>An alternative to working in a different directory than previously would be to use the **-name** option of `install_lmdz.sh` to download, install and run in a different target directory than the default.

## 2 About the arch files

In order to enable MPI and OpenMP, one has to set the corresponding options in arch files prior to compilation. In practice, this means one must create (the `install_lmdz.sh` script takes care of it) an `arch-local.fcm` files (in the `arch` subdirectory of the `LMDZ` directory).

This `arch-local.fcm` should be something like<sup>4</sup>:

```
%COMPILER          /usr/bin/mpif90
%LINK              /usr/bin/mpif90
%FPP                /usr/bin/mpif90 -E
%AR                 ar
%ARFLAGS            rU
%MAKE               make
%FPP_FLAGS          -P -traditional
%FPP_DEF            NC_DOUBLE
%BASE_FFLAGS        -cpp -ffree-line-length-0 -fdefault-real-8 -DNC_DOUBLE
%PROD_FFLAGS        -O3
%DEV_FFLAGS         -Wall -fbounds-check
%DEBUG_FFLAGS       -g3 -Wall -fbounds-check -ffpe-trap=invalid,zero,overflow -O0 -fstack-protector
%MPI_FFLAGS         -fcray-pointer
%OMP_FFLAGS         -fopenmp -fcray-pointer
%BASE_LD            -Wl,-rpath:/usr/lib
%MPI_LD             -fopenmp
%OMP_LD             -fopenmp
```

Where additions mainly concern the `COMPILER`, `LINK`, `MPI.FFLAGS` and `MPI_LD` lines for MPI, and the `OMP.FFLAGS` and `OMP_LD` lines for OpenMP.

Having correctly set an arch file, one can then compile and run in the various parallel modes, or even just in serial. In practice, it is convenient to put these instructions in scripts (see for instance the `compile.sh` in the `LMDZ` directory and the `bench.sh` and `bench_parallel.sh` scripts in the `BENCH32x32x39`), but in what follows the minimal necessary sequence of commands are given and explained one at a time.

## 3 Compiling and running LMDZ in MPI

In the `LMDZ` directory, compile the model in MPI mode:

```
./makelmdz_fcm -arch local -mem -parallel mpi -d 32x32x39 -j 8 -rad oldrad gcm
```

Note that here we compile without Orchidee (`-v false`).

The executable, `gcm_32x32x39_phylmd_oldrad_para_mem.e`, is generated in the `bin` subdirectory.

Once the model has been successfully recompiled, run a simulation. To do so, create a new subdirectory, e.g. `BENCH32x32x39_mpi`, in `LMDZ` and copy boundary conditions, initial conditions and parameter files (`limit.nc`, `star*nc`, `*.def`) over from directory `BENCH32x32x39`, along with the newly created executable. Since this case is without ORCHIDEE you need to adapt `config.def` and set

```
VEGET=n
```

To run, you will need to use the `mpirun` utility (from the MPI library) and specify the number of processes to run on (using the `-np` option), e.g. 4:

```
mpirun -np 4 ./gcm_32x32x39_phylmd_oldrad_para_mem.e > listing 2>&1
```

The run should be much shorter than if using only one process (verify this!).

When running in parallel, each MPI process creates its own `hist*` files. You will thus obtain for instance a files `histday.0001.nc`, `histday.0001.nc`, `histday.0002.nc` and `histday.0003.nc` which contains data relative to the sets of atmospheric columns managed by each process. To combine the output files from different processes into a single file containing the full dataset, use the IOIPSL `rebuild` script. The `rebuild` script is generated when installing IOIPSL. When installing IOIPSL, a `modipsl` directory was

---

<sup>4</sup>Check out the files in the `arch` directory for examples relative to other compilers; e.g. `ifort` in the `JeanZay` arch files.

created and the rebuild script installed in **modipsl/bin**<sup>5</sup>.

For each type of file, histday, histmth, etc., simply issue a command of the likes of<sup>6</sup> :

```
rebuild -o histday.nc histday_000*
```

to generate the recombined output file.

You can check that the output files thus generated are identical to the ones generated by the **BENCH32x32x39** run. Likewise for the **restart.nc** and **restartfi.nc** files. Note also that the **listing** file for the MPI run is larger than in a serial case, as most output messages are redundantly written by all processes.

## 4 Compiling and running LMDZ in OpenMP

Compile LMDZ in OpenMP mode:

```
./makelmdz_fcm -arch local -parallel omp -d 32x32x39 -j 8 -rad oldrad gcm
```

The executable, **gcm\_32x32x39\_phylmd\_oldrad\_para\_mem.e**, is generated in the **bin** subdirectory.

Once the model has been successfully recompiled, run a new simulation in a new subdirectory of **LMDZ**, e.g. **BENCH32x32x39\_omp** and copy over boundary conditions etc. from **BENCH32x32x39**, along with the newly created executable. Again, In this test you will run without ORCHIDEE you need to adapt **config.def** and set

```
VEGET=n
```

Before launching the run, some OpenMP environment variables must be set to specify the (maximum) amount of memory privately allocated to each thread, and the number of OpenMP threads to run with, e.g. 4:

```
export OMP_STACKSIZE=200M
export OMP_NUM_THREADS=4
./gcm_32x32x39_phylmd_oldrad_para_mem.e > listing 2>&1
```

Note that this time the generated hist\*nc files are unique, but suffixed 0000.nc. Again you can check that the model outputs are the same than those obtained in serial and pure MPI runs.

## 5 Compiling and running LMDZ in mixed MPI/OpenMP

Compile LMDZ in mixed MPI/OpenMP mode:

```
./makelmdz_fcm -arch local -parallel mpi_omp -d 32x32x39 -j 8 -rad oldrad gcm
```

Again, create a subdirectory in which to run the model. All that was mentioned in the previous sections on pure MPI and OpenMP runs combine when using the mixed mode. So assuming you want to run using 3 MPI processes, each using 2 OpenMP threads<sup>7</sup>:

```
export OMP_STACKSIZE=200M
export OMP_NUM_THREADS=2
mpirun -np 3 ./gcm_32x32x39_phylmd_oldrad_para_mem.e > listing 2>&1
```

Since you have used 3 MPI processes, output hist files will be split in 3 (i.e. histday\_0000.nc, histday\_0001.nc and histday\_0002.nc) and should be recombined using the **rebuild** tool. Once again, results should match those obtained with the serial, pure MPI, and pure OpenMP runs.

---

<sup>5</sup>Adding this directory to your PATH, e.g. in your `~/bashrc` to avoid having to type the full path each time you want to use rebuild is advised

<sup>6</sup>This is what the provided **reb.sh** script does. You need only run it and specify the file to rebuild as an argument; e.g. **reb.sh histday**

<sup>7</sup>Note that you can request using more cores than available on a given machine. This is of course extremely inefficient and one should strive to use at most the total number of available cores.

## 6 Exercice: Adjusting and setting the workload between MPI tasks

In the directories where you did your MPI (or mixed MPI/OpenMP) run, you'll find a **bands** file, e.g. **Bands\_32x32x39\_4prc.dat**, which contains information on how many columns were handled by each MPI process<sup>8</sup>.

The default behaviour, for LMDZ, is to load and follow the instructions of the bands file present in the directory where it runs. If it cannot find such a file, it then automatically generates one (which was the case in the test runs you've done so far) which simply considers splitting evenly the work between all available tasks. But this is rarely optimal.

There is an automated way of (iteratively) adjusting the workshare in LMDZ which can be triggered by setting the **adjust** parameter in file **run.def** to **y**.

**Important:** The **adjust** option should only be used in pure MPI mode, and is intended to be used to tune the bands file, and not to be used for production runs. Once a suitable **bands** file is obtained (which typically requires a month, i.e. 30 days long run), one should revert the **adjust** option to **n** and run with the resulting **bands** file.

Set up your experiment in a new directory. Import files (and MPI executable) there and set **adjust=y** in the **run.def** file, as well as **nday=30**. Run using 4 processes:

```
mpirun -np 4 ./gcm_32x32x39_phylmd_olddrad_para_mem.e > listing 2>&1
```

Check out the produced **Bands\_32x32x39\_4prc.dat** file and compare to the more naïve one that was previously generated.

To evaluate if this **bands** file is "converged", copy it aside for future reference and re-run LMDZ.

---

<sup>8</sup>The **bands** file contains information on how many columns are handled by each MPI process, but also for each of the four main "code steps": dynamics, tracer advection, dissipation, and physics. A **bands** file for N processes thus contains 4N lines. Each line contains two elements: the process number and number of atmospheric columns assigned to it.