Tutorial: Compiling and running LMD in parallel

The LMDZ team

December 7, 2015

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

This document can be downloaded as a pdf file:

```
wget http://www.lmd.jussieu.fr/~lmdz/Distrib/TD_para.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 OpenMPI library is installed).

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[1]:

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

Depending on your environment variables and settings[2], 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.sh script

You should have already installed the model, using the install.sh script, as instructed in the first LMDZ tutorial.

[1] 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.

[2] 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.

2 Setting up 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 you must create **arch-para.fcm** and **arch-para.path** files (in the **arch** sudirectory of the **LMDZ5** directory).
The **arch-para.path** should simply be a copy of the **arch-local.path** file that was generated by the install.sh script, and the **arch-para.fcm** should be something like[3]:

```
%COMPILER            mpif90
%LINK                mpif90
%AR                  ar
%MAKE                make
%FPP_FLAGS           -P -traditional
%FPP_DEF             NC_DOUBLE
%BASE_FFLAGS         -fdefault-real-8 -fcray-pointer
%PROD_FFLAGS         -O3
%DEV_FFLAGS          -g -O1 -Wall
%DEBUG_FFLAGS        -g -O0 -Wall -fbounds-check -ffpe-trap=invalid,zero,overflow
%MPI_FFLAGS          -I/usr/lib/openmpi/include
%OMP_FFLAGS          -fopenmp
%BASE_LD
%MPI_LD              -L/usr/lib/openmpi/lib -lmpi
%OMP_LD              -fopenmp
```

Where additions concern the **BASE_FLAG** line for general purpose, the **MPI_FFLAGS** and **MPI_LD** lines for MPI, and the **OMP_FFLAGS** and **OMP_LD** lines for OpenMP.

3 Compiling and running LMDZ in MPI

Once the arch files are correctly set, from the **LMDZ5** directory, compile the model in MPI mode:

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

The executable, **gcm_32x32x39_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 **LMDZ5** and copy boundary conditions, initial conditions and parameter files (**limit.nc, star*nc, *.def**) over from directory **BENCH32x32x39**, along with the newly created excutable.

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_para_mem.e > listing 2>&1
```

The run should be much shorter than the equivalent serial run (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,

[3]Check out the files in the **arch** directory for examples relative to other compilers; e.g. ifort in the Ada arch files.

a modipsl directory was created and the rebuild script installed in **modipsl/bin**[4].
For each type of file, histday, histmth, etc., simply issue a command of the likes of:

```
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 serial run. Likewise for the **restart.nc** and **restartfi.nc** files. Note also that the **listing** file for the MPI run is larger than in the 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 para -mem -parallel omp -d 32x32x39 -j 8 gcm
```

The executable, **gcm_32x32x39_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 **LMDZ5**, e.g. **BENCH32x32x39_omp** and copy over boundary conditions etc. from **BENCH32x32x39**, along with the newly created excutable.

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_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 para -mem -parallel mpi_omp -d 32x32x39 -j 8 gcm
```

Again, create a subdirectory in which to run the model. All that was mentionned 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[5]:

```
export OMP_STACKSIZE=200M
export OMP_NUM_THREADS=2
mpirun -np 3 gcm_32x32x39_phylmd_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.

[4]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

[5]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 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 where handled by each MPI process[6].

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 automaticaly 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 you 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_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.

Note: At low resolution, and on a machine shared by many users, it is quite possible to obtain surprising results with the automated load adjustment scheme. In principle the adjustment should be done in a "controlled" environment to be meaningful.

[6]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.