Installing LMGC90 on ComputeCanada servers

David Cantor *Tristan* Vuilloz

December 25, 2020

ComputeCanada is the national network of servers for high-performance computing. It is composed of 4 main servers:

- Cedar
- Graham
- Béluga
- Niagara

These servers are maintained and located at different universities across the country. In particular, the server Béluga is located at the University of Montréal and managed by the regional computing service CalculQuébec. It makes the access and support to this server faster when working in Montréal. However, each cluster has particular advantages in performance and jobs submission policies one should study to choose the servers which would be the more suitable for one's works.

The procedures to access, upload/download, install software, and run scripts are similar in all of them. Please be careful when using the clusters. These are systems shared by many people and a wrong set of parameters can cause other simulations to stop or dramatically slow down. Although there's a large storage space in these servers, please download or backup your data on local storage if possible. If, for any reason, the storage memory is full, everybody's simulations will stop.

Each one of these servers is provided with a set of structural modules giving you access to different versions of software packages you may want to use. This root set is called a "standard environment" and is named `StdEnv`. At the present time it exists three competing versions of these `2016.4`, `2018.3` and `2020`. For those of you who might want to investigate what truly lies behind, learn that detailed information on the major additions developed between each version can be find here.

How to access the servers

You should have an account on ComputeCanada supported by a senior researcher. In our case, it is Carlos' account. You will receive a username and password that you should keep secret. Many attacks on servers occur because of a bad use of identifiers.

The access to the server uses `ssh` protocol. In a terminal, with your username (I'm going to make the examples with mine `-dacantor-`) write the line:

```
davidcantor@MacBook-Pro-de-David ~ % ssh dacantor@beluga.computecanada.ca
```

You will be asked to write your password. When it's done, press enter.

Then, you should see the following:

```
dacantor@beluga.computecanada.ca's password:
Last login: Wed May 20 19:05:18 2020 from 38.87.8.149
################################################################################
 _          __ _
| |__   ___/_/| |_   _  __ _  __ _    Bienvenue sur Béluga / Welcome to Béluga
| '_ \ / _ \ | | | | |/ _` |/ _` |
| |_) |  __/ | | |_| | (_| | (_| |   Aide/Support:    support@calculcanada.ca
|_.__/ \___|_|\__,_|\__, |\__,_|    Globus endpoint: computecanada#beluga-dtn
                    |___/           Documentation:   docs.calculcanada.ca


2019-04-18    Sauvegarde de l'espace project / Backup of project space
              =========================================================

Étant donné les transferts massifs de données vers Béluga, le système de
sauvegarde n'est pas nécessairement à jour à tous les jours.

Because of massive data transfers to Béluga, the backup system is not
necessarily up to date everyday.

################################################################################
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory
[dacantor@beluga4 ~]$
```

TIP: You may log into the server very frequently, so it may be annoying to write down each time the `ssh` line and your password. Install and use `sshpass` that will allow you to create a `bash` script that you can run each time you want to log-in.
No need to write down the `ssh` line or your password. Your script may look something like this:

```
 GNU nano 2.9.3                    r_beluga.sh                         Modified

#!/bin/bash

export SSHPASS='XXXXXXXXX'

sshpass -e ssh dacantor@beluga.computecanada.ca

echo "Logged out of Beluga." # Message displayed by the terminal after you disconnect from the server.
```

Then, in your terminal you only need to run the line below to get logged in the server.

```
davidcantor@MacBook-Pro-de-David ~ % bash r_beluga.sh
```

Once logged in, you can see `ls` the structure of the folder you have access to by using the command:

```
[dacantor@beluga4 ~]$ ls
nearline  projects  scratch
[dacantor@beluga4 ~]$
```

Structures of the folders

1) `nearline`: Tape-like storage that does not consume our quotas of maximum storage. This folder allows you to store large files but it is quite slow compared to the others. Reading information from `nearline` can be time consuming as well (in the order of some minutes).

2) `project`: Fast and daily backup storage of files. Files stored here are permanent and can be shared between the members of the group. The limit storage is 1 TB per group.

In this folder you should upload and install your version of LMGC90 or any other compiled code. However, you should not run your simulations here.

3) `scratch`: Large space for temporary storage of files during computations. The limit storage is 20 TB per user. Files of more than 60 days old are automatically deleted. So please, periodically check and download your files in scratch or you risk losing them permanently.

Your scripts for simulation (`gen_sample`, `command`, and others) should be uploaded to your scratch and ran from there.

Uploading and downloading data

Among different methods, we recommend using `rsync` for uploading and downloading files to the server.

Example: uploading LMGC90 to your `project` folder.

- My `project` folder has the following path:

```
[dacantor@beluga4 dacantor]$ pwd
/home/dacantor/projects/def-covalle/dacantor
[dacantor@beluga4 dacantor]$
```

In that location, create a new folder called `lmgc90_v202005` (you can use another name if you want) using :

```
[name@server name]$ mkdir lmgc90_202005
```

Then, you can get inside using :

```
[name@server name]$ cd lmgc90_202005
```

Using `pwd`, you will get the path to that location:

```
[dacantor@beluga4 lmgc90_v202005]$ pwd
/home/dacantor/projects/def-covalle/dacantor/lmgc90_v202005
[dacantor@beluga4 lmgc90_v202005]$
```

Now, on your computer, using another window of terminal, go find the sources of LMGC90.

NOTE: Use the `version2019.rc1` available in the website.

Upload your sources using the following command:

```
davidcantor@MacBook-Pro-de-David ~ % rsync -r -vP -z . dacantor@beluga.computecanada.ca:/home/dacantor/projects/def_cova
lle/dacantor/lmgc90_v202005
```

To download files, for instance from a folder in your simulation folder `scratch`, you should refer to the following procedure.

First, go with a new window of terminal to the location you want ti download the files.

To know the location of your files in the server you can use `pwd`.

```
[dacantor@beluga4 test_run]$ pwd
/home/dacantor/scratch/test_run
[dacantor@beluga4 test_run]$
```

In order to download the files in the folder `test_run` use:

```
davidcantor@MacBook-Pro-de-David ~ % rsync -r -vP -z . dacantor@beluga.computecanada.ca:/home/dacantor/scratch/test_run
```

You will to type down your password and the files will start to be downloaded.

Modules in the server

The servers have a vast collection of softwares available. However, you have to explicitly define which are the modules you want to use. For instance, during the compilation of LMGC90 we need fortran, vtk, C++, and others.

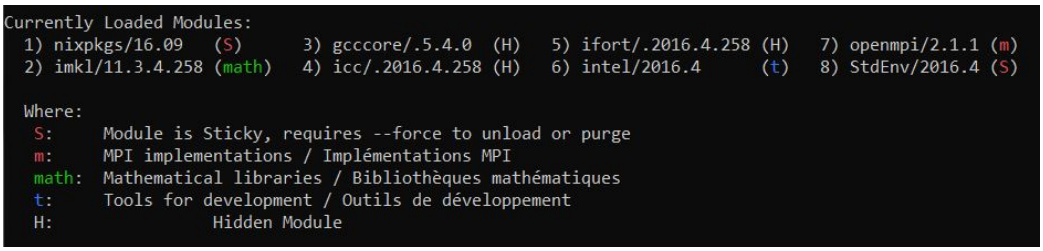
The collection originally implemented on each cluster is defined by their standard environment. For a given `StdEnv` version, the implementation of LMGC90 needs a specific handling procedure.

- `StdEnv/2018.3` is the standard environment originally implemented on both Béluga and Niagara.
- `StdEnv/2016.4` is the standard environment originally implemented on both Cedar and Graham.

First and foremost, we remind you that at any time and in any repertory of your server share you can access to the list of modules loaded by using the command:

```
[name@server ~]$ module list
```

By doing so you must obtain an initial structure similar to the one displayed below. In particular, you must also be able to check, among a set of modules, which version of `StdEnv` is implicitly provided on the cluster you have chosen.



For instance, Béluga actually owns the `StdEnv/2018.3`.

At this point, you need to have uploaded LMGC90 data in a definite folder in such location:

```
[name@server ~]$ cd projects/account_name/name/lmgc90_
202005/
```

To load the modules that allow us to compile LMGC90, write the following line:

```
[name@server lmgc90_202005]$ module purge A
```

If you are using **Cedar** or **Graham**

As previously stated, the compilation of LMGC90 requires specific modules but the root set `StdEnv/2016.4` is unable to give you access to the the needed packages. We are actually looking for its next update named `StdEnv/2018.3` (originally present on Béluga and Niagara) which you can load by overwriting the former environment:

```
[name@server lmgc90_202005]$ module load StdEnv/2018.3 *
```

NOTE: If the one above does not work, you may try:

```
[name@server lmgc90_202005]$ module switch StdEnv/2016.4
StdEnv/2018.3 *
```

```
The following have been reloaded with a version change:
  1) StdEnv/2016.4 => StdEnv/2018.3            5) imkl/11.3.4.258 => imkl/2018.3.222
  2) gcccore/.5.4.0 => gcccore/.7.3.0          6) intel/2016.4 => intel/2018.3
  3) icc/.2016.4.258 => icc/.2018.3.222        7) openmpi/2.1.1 => openmpi/3.1.2
  4) ifort/.2016.4.258 => ifort/.2018.3.222
```

Also, add:

```
[name@server lmgc90_202005]$ module load vtk nixpkgs/16.09
gcc/7.3.0 hdf5 B
```

```
Lmod is automatically replacing "intel/2018.3" with "gcc/7.3.0".


------------------------------------------------------------------------------------------------------
The following dependent module(s) are not currently loaded: hdf5/1.10.3 (required by: netcdf/4.6.1, vtk/
8.2.0), netcdf/4.6.1 (required by: vtk/8.2.0)
------------------------------------------------------------------------------------------------------


Due to MODULEPATH changes, the following have been reloaded:
  1) openmpi/3.1.2
```

NOTE: The previous lines imply that we will be using Python 3 instead of Python 2.7. If you find some errors in your scripts make sure everything is not related to the version of Python

Finally, check another time the list of modules loaded.

```
Currently Loaded Modules:
  1) StdEnv/2018.3    (S)      6) ifort/.2018.3.222 (H)     11) openmpi/3.1.2 (m)
  2) nixpkgs/16.09    (S)      7) python/3.7.4      (t)     12) mpi4py/3.0.3  (t)
  3) imkl/2018.3.222 (math)    8) scipy-stack/2019a (math)  13) vtk/8.2.0     (vis)
  4) gcccore/.7.3.0   (H)      9) gcc/7.3.0         (t)     14) hdf5/1.10.3   (io)
  5) icc/.2018.3.222  (H)     10) netcdf/4.6.1      (io)

  Where:
   S:     Module is Sticky, requires --force to unload or purge
   m:     MPI implementations / Implémentations MPI
   math:  Mathematical libraries / Bibliothèques mathématiques
   io:    Input/output software / Logiciel d'écriture/lecture
   t:     Tools for development / Outils de développement
   vis:   Visualisation software / Logiciels de visualisation
   H:               Hidden Module
```

Hopefully, you obtain the type of response displayed abovc.

You will need to run the same two or three lines (`A`, `B` and if needed `*`) before running your simulations. In fact, you will systematically add them in each `bash` script from which we describe the structure later.

Installing LMGC90

Adequate modules are now loaded. The compilation is done as explained in the procedure given by the LMGC90_wiki page, although some flags should be added as written below.

Create a `build` folder with:

```
[name@server lmgc90_202005]$ mkdir build
```

Access to the `build`:

```
[name@server lmgc90_202005]$ cd build
```

And compile with:

```
[name@server build]$ cmake ..  -DMATLIB_VERSION=none
-DSPARSE_LIBRARY=none
```

Finally:

```
[name@server build]$ make
```

The compilation went probably well, without raising any issue.

Running your simulations with the queuing assistant: SLURM

To run your simulations, you must use a queuing assistant called Slurm. You should not run the simulations directly typing `python gen_sample.py` or

`python command.py`. The queuing assistant is in charge of allocating space and sequentially let people run their simulations. The Slurm assistant can work using a `bash` script. This same script will have to be in the folder of your simulation located somewhere in your `scratch`:

```
[name@server ~]$ cd scratch/path/test_run/
[name@server test_run]$ pwd
/home/name/scratch/path/test_run
```

After you have uploaded the files referring to your simulation in its folder, create the `bash` script with the command:

```
[name@server test_run]$ nano script.sh
```

In the nano editor you are then able to write a simple `bash` script from which we give you an example of structure below.

```
#!/bin/sh
#SBATCH -N 1
#SBATCH --account=account_name
#SBATCH --time=1-10:10        # This will run for 1DAY, 10HOURS and 10MINUTES
#SBATCH --mail-user=your@mail # Send email updates to you or someone else
#SBATCH --mail-type=ALL       # Send an email in all cases (job started, job ended, job aborted)

module purge
module load StdEnv/2018.3 #This line is obviously only needed if you are using Cedar or Graham!
module load vtk nixpkgs/16.09 gcc/7.3.0 hdf5

export PYTHONPATH=$PYTHONPATH:/home/trvui/projects/def-covalle/trvui/lmgc90_v202005/build

python command.py
```

- The first line says that this is a `bash` script.
- Second line states that this belongs to the group `account_name`.
- Third line states the duration of the simulation. In Béluga and Niagara the limit is set to 7 days, 28 days in Cedar and Graham.
- Fourth and fifth lines are optional but helpful. They will keep you updated of your simulation's state by email so you do not have to keep an eye on it constantly.
- Then, the modules are purged and loaded as required.

- Next line states where the sources of LMGC90 are located.
- And finally, we run the simulation.

Then, run it using:

```
[name@server test_run]$ sbatch script.sh
```

To see all the jobs running at the same time use `squeue` (there are hundreds!). To check the status of your jobs using `squeue` and your username as follows:

```
[name@server ~]$ squeue -u name
```

To cancel a job use `scancel` and the number `id` of your job. The outputs of your simulation will be located in a file called `slurm-XXX` with `XXX` the `id` of your job.

NOTE: Remember you should run your simulation in the `scratch` folder!

More options with SLURM

Slurm gives you many options for running your simulations. You can run simulations in parallel if you compile the corresponding version of the code. When running simulations in parallel you should also explicitly state some variables in the Slurm script. We strongly recommend reading the options of running simulations in array or in parallel if necessary.

Follow the instructions here.

Important: When running simulations in parallel remember to set in the Slurm script the following variables (and those mentioned in the link above):

```
export OMP_SCHEDULE=STATIC
export OMP_NUM_THREADS=n
export OPENBLAS_NUM_THREADS=1
```

where `n` is the number of threads you want to use (usually no more than the number of core available).