TITANI CLUSTER MANUAL

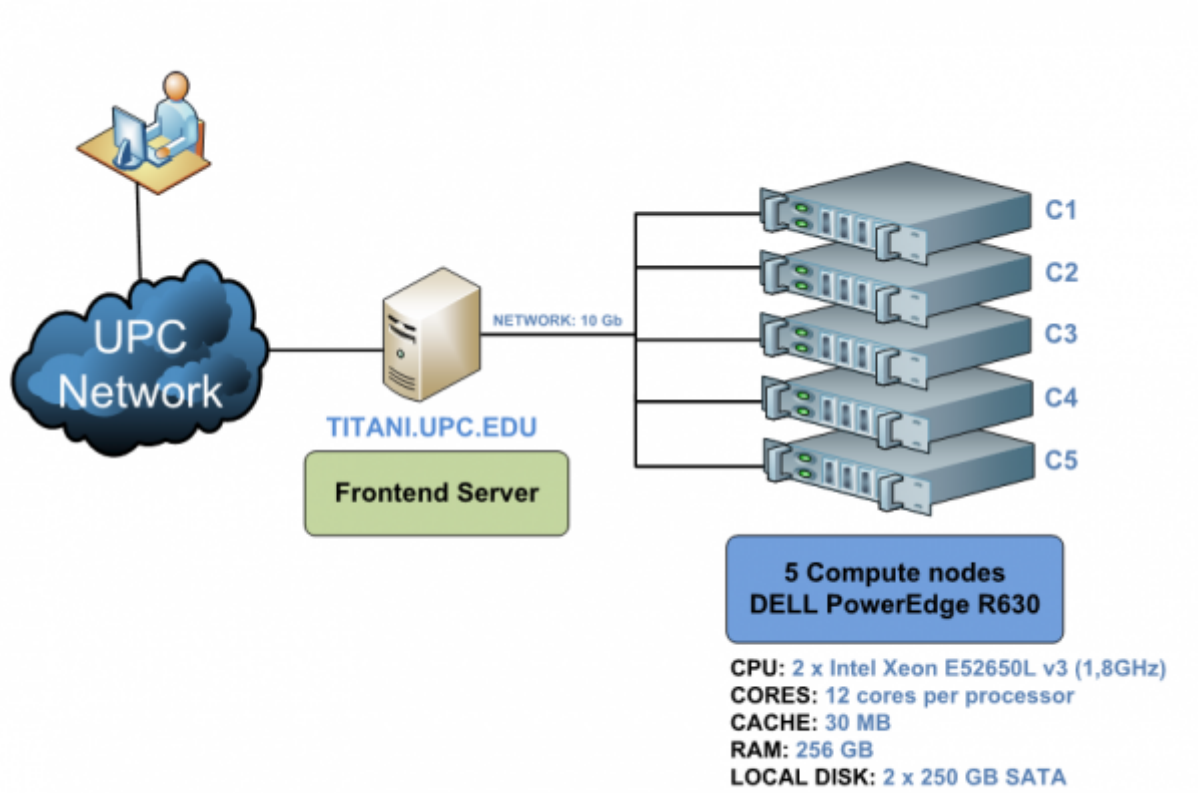
This document is intended to give some basic notes in order to work with the TITANI High Performance Green Computing Cluster of the Civil Engineering School (ETSECCPB) located in the CaminsTECH facilities.

1.- ARCHITECTURE

TITANI is a **high performance green computer cluster** formed by a set of Dell servers. It is a multitasking and multiuser environment configured with **CentOS 7** Linux operating system and uses the **SLURM** open-source workload manager (slurm.schedmd.com). TITANI is oriented towards users that execute programs with high computing performance requirements of memory and processing time. The cluster has two main parts:

- A **frontend/access server**:
  - This is the server where users can launch their simulations (jobs), but they actually run on the computing servers. Every user has his personal workspace and useful tools such as compilers, editors, etc.
  - Name: **titani-nx**
- **Five compute servers/nodes (24 cores per server)**:
  - Every compute server executes submitted jobs from the fronted server.
  - Name: **cN**, where N is the number of the compute server (e.g. c1, c2...)

As a **Green Computing Cluster**, the compute servers are powered on while jobs are running (when a node is idle for 30 minutes it will be powered off). An overview of the architecture of TITANI and its characteristics is shown below:



2.- ACCES MODES

In order to work with TITANI cluster, it is required that the user asks for an account with the UTGAC IT Services (https://caminstech.upc.edu/es/calculintensiu). Once the account is created, UPC users will be able to gain access to the service with their UPC username and password, otherwise a local username and password will be given. Connection to the server is only allowed from computers in Camins networks (buildings B1, B2, C1, C2, D1, D2 and the computer rooms of the Civil Engineering School) or from anywhere using the UPCLink VPN client (VPN UPC Manuals).

2.1.- Terminal Access

You can use any tool that supports **secure shell (ssh)** to connect to the service.

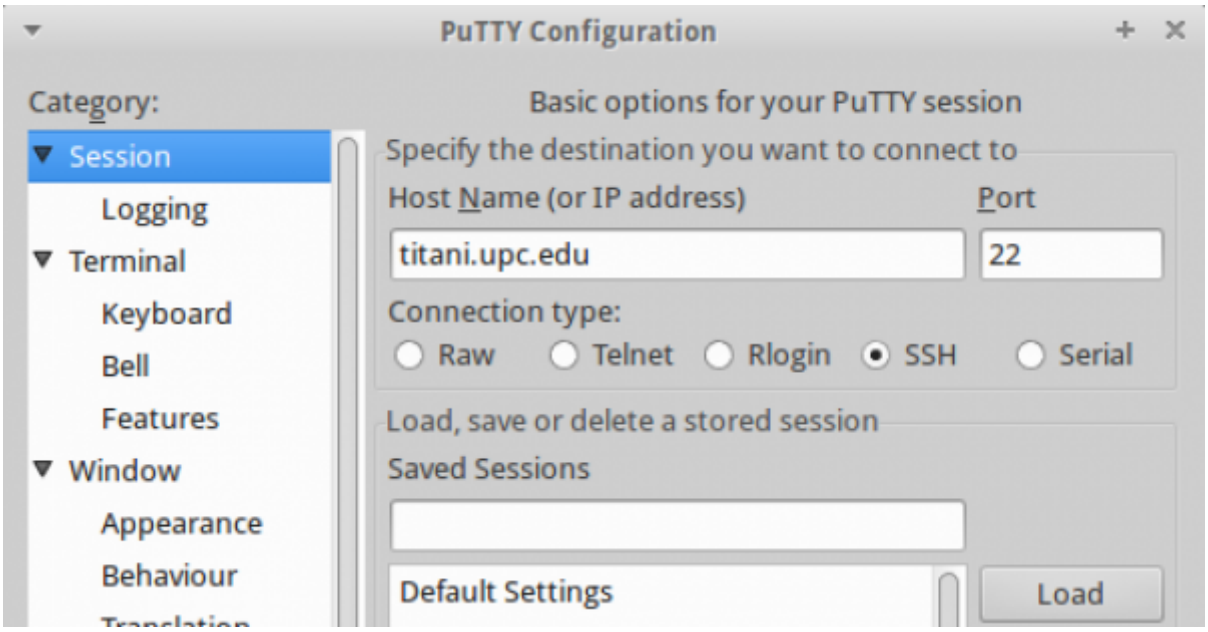
- In **Linux** almost all distributions have a preinstalled ssh client.
- In **MacOSX** there is a ssh client in Terminal.
- In **Windows** for example, use Putty (ssh client) or Mobaxterm (cygwin terminal with a SSH and SFTP client that includes Graphical Access).

Putty

Configure connection as follows:

Host name	titani.upc.edu
Port number	22

User name	UPC_username
Password	UPC_password



2.2.- Graphical Access

To run programs using graphical output, such as Abaqus CAE or Matlab, use:

- Mobaxterm (cygwin terminal with a SSH and SFTP client)
  - http://mobaxterm.mobatek.net/
- x2go client
  - http://wiki.x2go.org/doku.php/doc:installation:x2goclient

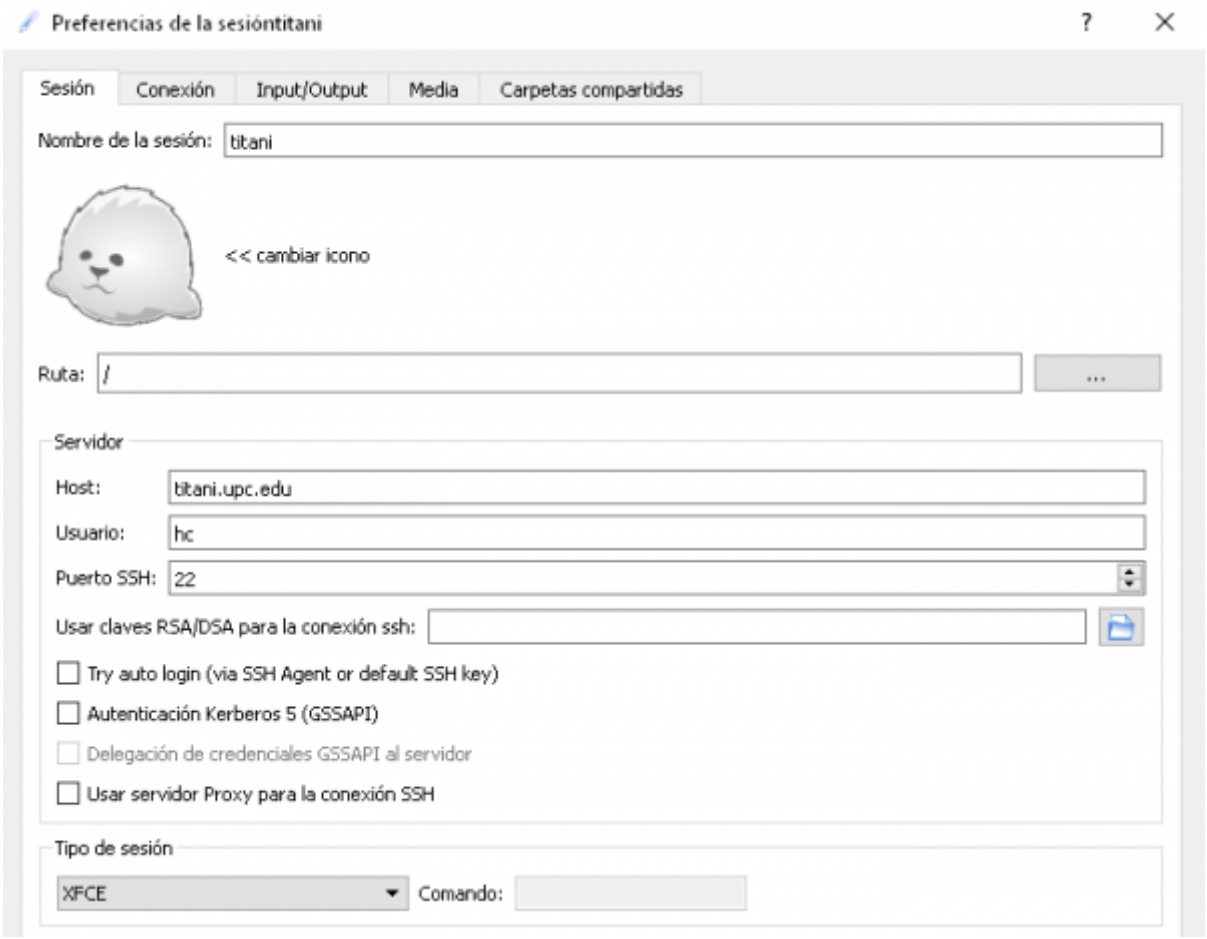
Both options (terminal and graphical access) use the SSH protocol for secure connections.

X2go:

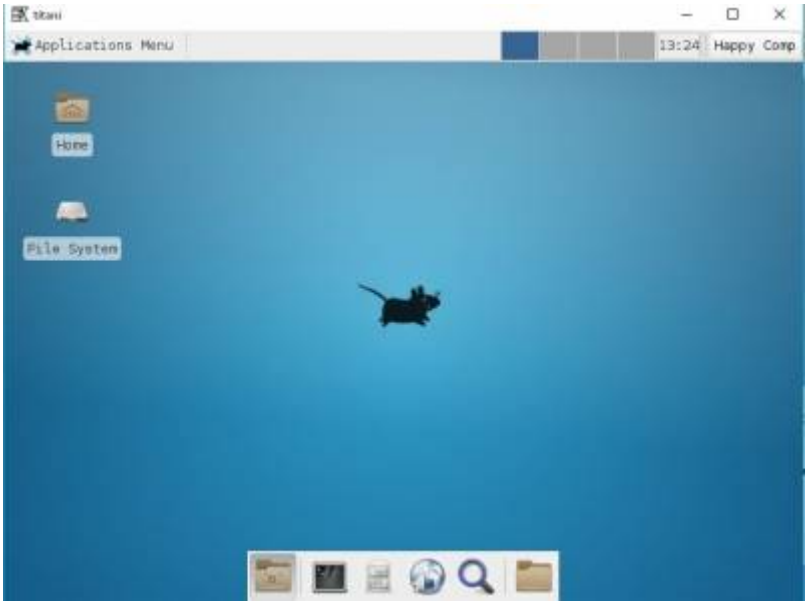
Add a New Session with the parameters in the table below.

Host / IP	titani.upc.edu / 147.83.87.15
Port (TCP)	22
Services	SSH, SFTP and x2go
Authentication method	username/password
Desktop	XFCE

Your preferences screen should look like this:

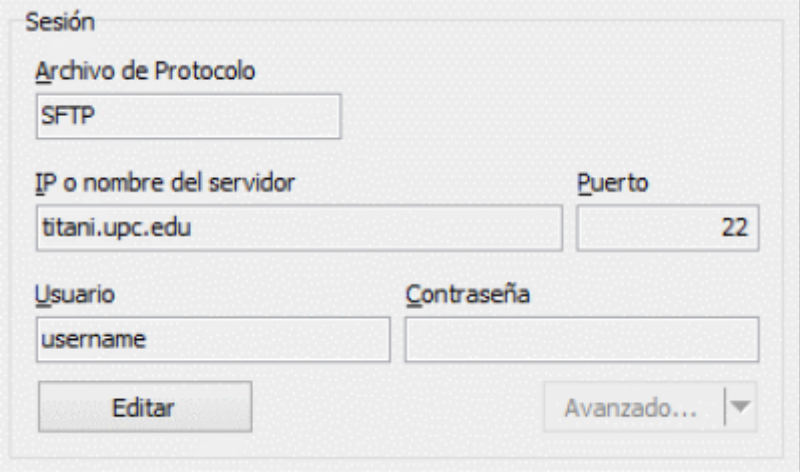


Once you insert your username and password you will see your personal desktop:



2.3.- File Transfer

In order to download/transfer files from/to TITANI, it is recommended to use a tool such as winSCP configuring it as follows.



3.- LINUX BASIC COMMANDS

TITANI Cluster is a CentOS 7 Linux operating system. These are some basic commands in a Linux environment (shell) that you will find useful when working on it.

Command	Description	Example
pwd	shows the current working directory	`$ pwd`
mkdir	creates a new directory	`$ mkdir directory`
cd	changes from one directory to the one specified	`$ cd directory`
ls	lists the files and directories under the current directory	`$ ls`
mv	moves files or subdirectories from one directory to any other specified or renames the file or directory	`$ mv test.txt /home/directory`
cp	copies files into the specified directory	`$ cp test.txt /home/user`
rm	deletes a file NOTE: once deleted it cannot be recovered	`$ rm test.txt`
rmdir	deletes the specified directory which must be empty	`$ rmdir directory`
more, cat	allows file visualization, not modifications allowed	`$ more test.txt`, `$ cat test.txt`

For more detailed information about Linux Shell commands take a look at http://linuxcommand.org/lc3_learning_the_shell.php

4.- DISK SPACE

Each user has a **personal disk space** to store files in TITANI. The user's home directory has the following path:

```
/users/USERNAME
```

It is recommended to use this space to store code, executable and data files. The initial disk quota assigned is 10 GB per user; however should you need to increase the amount of space, it is possible to request this from UTGAC IT Services (https://caminstech.upc.edu/es/on-trobar-nos). To check the

available quota use the next command:

```
> quota -s
```

Additionally, every user has **temporary disk space (scratch)** available.

4.1.- NFS scratch personal directory

```
/scratch/USERNAME
```

This directory does not have any quota restrictions, for this reason, it is suggested to use this temporary scratch directory to run jobs sent to the batch queues and to store execution results. All compute nodes have access to this NFS mounted scratch directory. The amount of space available is 4.4 TB and it is shared by all users in the cluster, therefore it is requested that users consider others by not overloading the disk. In the case that a user occupies too much space and affects the jobs of others, **the system administrator could delete files without notice. Please note: files in the scratch not changed in more than 32 days will be removed every day at 11 PM.**

4.2.- Local scratch node directory

```
/local_scratch
```

When a job is sent to a batch queue it will run in one of the cluster nodes (c1-c5). Users can use the /local_scratch directory (450 GB of space physically allocated in every node disk). The local scratch space is a storage space unique to each individual node where a job is running and therefore **disk access is faster than in home or scratch directories** (NFS network disks). Using /local_scratch improves programs performance when they have a lot of input/output operations (a large amount of reading/writing files). To use this temporary directory it will be necessary for every user to create a folder inside /local_scratch directory and copy the files to run a job. Once the job finishes it is required to copy the result files back to the user home or scratch personal directory, as the local scratch directory will no longer be accessible. Don’t forget to include these steps in the shell script that is sent to the batch queue.

1. Data files should be copied from the home directory or personal scratch to the local scratch space.
2. Run the program.
3. Copy the output files to the original directory.

It is important to keep in mind that all users must remove excess files by themselves. Preferably this should be done within the user's batch job when the computation has finished. Leaving files in the local scratch directory could be an impediment to other users who are trying to run their jobs. Simply delete all files when the job ends. Example: The following script creates/deletes a temporary directory in */local_scratch* using the username and the SLURM JOBID as the directory name, in this way it is possible to send more than one job using */local_scratch* at the same time.

```
#!/bin/bash
MYSCRATCHDIR=/local_scratch/$USER/$SLURM_JOB_ID
mkdir -p $MYSCRATCHDIR
```

```
cp -r /users/$USER/sourcedir/* $MYSCRATCHDIR
cd $MYSCRATCHDIR
$MYSCRATCHDIR/program
cp -r $MYSCRATCHDIR /users/$USER/resultsdir
rm -rf $MYSCRATCHDIR
```

5.- SOFTWARE

5.1.- Compilers

Fortran compilers installed:

COMPILER	COMMAND
Intel Fortran Compiler (14.0.0)	ifort
GNU Fortran 95 compiler (4.8.5 -default-, 6.2.0, 7.4.0)	f95, gfortran

C/C++ compilers installed:

COMPILER	COMMAND
Intel C++ Compiler (14.0.0)	icc
GNU project C and C++ compiler (4.8.5 -default-, 6.2.0, 7.4.0)	cc, gcc, g++

Additionally installed, is the Intel Cluster Toolkit for Linux (version 3.2) which includes the following tools:

- Intel MPI Library 4.1
- Intel Math Kernel Library 11.1
- Intel Trace Analyzer and Collector
- Intel MPI Benchmarks (IMB)
- Intel Debugger for Linux

5.2.- Parallel Computing

The Intel and GNU the MPI libraries (Message Passing Interface) are installed in TITANI through its OpenMPI implementation (Version 1.10). These libraries are interfaces for parallel computing designed for programs that take advantage of multiple cores. The following link provides a user's guide for MPI programming in Fortran.
https://caminstech.upc.edu/sites/default/files/ManualMPIFortran.pdf

Compilation Commands using OpenMP (only shared memory):

Use the following commands to compile a program prepared for parallelization with OpenMP:

GNU Compilers:

- **GNU Fortran:**

```
f95 –o program_name –fopenmp program_code.f

gfortran –o program_name –fopenmp program_code.f
```

- **GNU C/C++:**

```
gcc –o program_name –fopenmp program_code.c

g++ –o program_name –fopenmp program_code.cpp
```

Intel Compilers:

- **Intel Fortran:**

```
 ifort –o program_name –openmp program_code.f
```

- **Intel C/C++:**

```
icc –o program_name –openmp program_code.c

icpc –o program_name –openmp program_code.cpp
```

Compilation Commands using MPI (shared and distributed memory):

Use the following commands to compile a program prepared for parallelization with MPI:

GNU Compilers:

- **GNU Fortran MPI:**

```
mpif90 –o program_name program_code.f

mpifc –o program_name program_code.f
```

- **GNU C/C++ MPI:**

```
mpicc –o program_name program_code.c

mpicxx –o program_name program_code.cpp
```

Intel Compilers:

- **Intel Fortran:**

```
 mpiifort –o program_name program_code.f
```

- **Intel C/C++:**

```
mpiicc –o program_name program_code.c

mpiicpc –o program_name program_code.cpp
```

Execute the following command to display more help regarding this commands:

```
$ command -help
```

5.3.- Running Jobs: batch queues (partitions)

TITANI uses the SLURM open-source workload manager to manage cluster resources (nodes, processors, cores, memory). It provides a queue management system oriented to send jobs for their execution and uses a job scheduler to allocate the simulations to the compute nodes. The job scheduler sends the job from the TITANI frontend server to any compute node depending on the instant cluster resource usage. When all resources are busy, newer jobs will wait until older jobs finish their processes. Therefore, to execute a job it is required to send it to a predefined batch queue according the program specific needs. The batch queues (called partitions in SLURM) are described below in detail as well as its maximum CPU run time, CPU maximum wall time, maximum number of cores allowed, maximum number of simultaneous jobs allowed to be executed or queued, memory limitations and other restrictions. Consider the following recommendations to send jobs to the queues:

- Each queue imposes restrictions per user for the limit of executing and pending jobs.
- It is forbidden to link jobs execution that means the same shell script executes one job after another executed before.
- Contact the UTG IT Services to ask for access to restricted queues in the particular case you need to increase the limits imposed for each queue.

Queues' characteristics could be modified regarding its maximum number of jobs and the memory limits according the users' needs, the computers' workload, among other circumstances, therefore it is recommended to consult the following link to have the most up to date characteristics:

http://caminstech.upc.edu/faq/titani

There are two queues (partitions) defined in TITANI: serial and parallel.

- SERIAL queue: This is the default queue and it only allows to submit serial jobs, namely that only use a core in their execution. Every user can run 8 jobs simultaneously and has 4 pending jobs.
- PARALLEL queue: This queue only allows to submit parallel jobs that is to say that use more than a core in their execution. Every user can run 2 jobs simultaneously and has 2 pending jobs. It is allowed to use 48 cores per user, in a job or in total with all his jobs.
- EXPRESS queue: This queue only allows to submit fast parallel jobs (maximum 24 hours in the queue) that is to say that use more than a core in their execution. Every user can run 6 jobs simultaneously and has 6 pending jobs. It is allowed to use 48 cores per user, in a job or in total with all his jobs.

The Serial queue uses C3, C4 and C5 computing nodes (72 cores) and a maximum of 24 cores to execute their jobs. The Parallel queue uses C2, C3, C4 and C5 computing nodes (96 cores) and a maximum of 84 cores can be used. The Express queue uses C1, C2, C4 and C5 computing nodes (96 cores) and a maximum of 96 cores can be used.

The following table reflects a summary of the characteristics of each queue. This information has been retrieved by the time this tutorial has been created, consult the most up to date information at the link mentioned before.

QUEUE	USER RESTRICTIONS		JOB LIMITS	
	Max Number of running (pending) jobs	Max Number of cores	Max CPU Run TimeMinutes, (Hours), [Days]	Memory limit
serial	8 (4)	-	20160, (336), [14]	16 GB/job
parallel	2 (2)	48	60480, (1008), [42]	256 GB/job, 128 GB/node
			Wall Time 10080, (168), [7]	
express	6 (6)	48	8640, (144), [6]	256 GB/job, 128 GB/node
			Wall Time 1440, (24), [1]	

5.3.1.- Sending jobs

As mentioned before to run simulations at TITANI is necessary to send jobs to the computing queues of the cluster to use the compute nodes. In fact, it is not allowed to execute high CPU usage on the frontend server because it has not enough resources to do theses taks. The **sbatch** command is used to send jobs to a defined queue of TITANI. It is required to make a *shell script*. This script would look like the following:

```
#!/bin/bash
<path>/program_to_execute
```

Where <**path**> is the full name of the directory where the program to run is. Use the following syntax:

```
 > sbatch –p queue_name <shell_script_name>
```

In the directory where the **sbatch** command was launched, SLURM creates one output file called `slurm-<JOBID>.out` containing all output and errors of the execution. <JOBID> is the own ID of each sent job or job identifier.

By **default**, the **sbatch** command sends a job to the serial queue and asks to the job scheduler for 2048MB of RAM memory. If you need more memory resources you can use this flag:

```
  --mem <amount_of_RAM(MB)>
```

Please, do not ask for more memory than needed.

Example: submit a job myscript.sh asking for a machine 10GB of RAM.

```
 sbatch [-p serial] –-mem 10000 myscript.sh
```

It is also possible and very useful to specify the job parameters inside the shell script. These are the most used:

```
#!/bin/bash

#SBATCH --partition=<name_of_the_queue>
#SBATCH --job-name=<name_for_the_job>
#SBATCH --output=<output_filename.out>
#SBATCH --error=<error_filename.err>
```

```
#SBATCH --mem=<amount_of_RAM(MB)>
#SBATCH --cpus-per-task=<num_of_threads> # OMP programs
#SBATCH --nodes=<num_of_nodes> # MPI programs #SBATCH --ntasks-per-
node=<num_of_cores> # MPI programs
```

5.3.2.- Sending parallel jobs

In order to send jobs for parallel execution, namely using more than one core, it must be used the **parallel** or **express** queue.

The command used would be the following syntax:

```
> sbatch –p parallel <parallel_script_name>
```

By **default**, the **sbatch** command asks to the job scheduler for 2048MB of RAM memory per node. If you need more memory resources you can use this flag:

```
--mem <amount_of_RAM_per_node(MB)>
```

Using OpenMP (only shared memory):

```
> sbatch –p parallel –c <num_threads> omp_script
> sbatch –p parallel –-cpus-per-task=<num_threads> omp_script
```

A *omp_script* file would be for instance:

```
#!/bin/bash
<path>/omp_program_to_execute
```

Example: OMP program using 4 threads and 12 GB of RAM in the parallel queue:

```
> sbatch –p parallel –c 4 –-mem=12000 omp_script
```

Example: OMP program using 4 threads to the parallel queue specifying the parameters in the shell script *omp_script*:

```
#!/bin/bash
#SBATCH --partition=parallel
#SBATCH --mem=8000
#SBATCH --cpus-per-task=4
<path>/omp_program_to_execute
```

Shell command:

```
 > sbatch omp_script
```

Using MPI (shared and distributed memory):

```
> sbatch –p parallel –-ntasks-per-node=<num_of_cores> mpi_script
> sbatch –p parallel -–nodes=<num_of_nodes>
                      –-ntasks-per-node=<num_of_cores> mpi_script
```

A *mpi_script* file would be for instance:

```
#!/bin/bash
mpirun <path>/mpi_program_to_execute
```

Example: MPI program using 8 cores in the parallel queue:

```
> sbatch –p parallel –-ntasks-per-node=8 mpi_script
```

Example: MPI program using 8 cores in the parallel queue specifying the parameters in the shell script *mpi_script*:

```
#!/bin/bash
#SBATCH --partition=parallel
#SBATCH –-ntasks-per-node=8
mpirun <path>/mpi_program_to_execute
```

Shell command:

```
> sbatch mpi_script
```

Example: MPI program using 2 nodes and 4 cores per node in the parallel queue:

```
> sbatch –p parallel –-nodes=2 –-ntasks-per-node=4 mpi_script
```

Example: MPI program using 2 nodes: 4 cores and 4 GB per node in the parallel queue specifying the parameters in the shell script:

```
#!/bin/bash
#SBATCH --partition=parallel
#SBATCH --nodes=2
#SBATCH --mem=4000 #SBATCH –-ntasks-per-node=4

mpirun <path>/mpi_program_to_execute
```

```
> sbatch mpi_script
```

5.3.3.- Consulting service and jobs in the queues

The **status of the service** is shown by main commands **sinfo** and **squeue**:

```
> sinfo
```

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
serial*    up     infinite    3    idle~ c[3-5]
serial*    up     infinite    1    mix      c2
serial*    up     infinite    1    alloc c1
parallel   up     infinite    3       idle~ c[3-5]
parallel   up     infinite    1    mix      c2
parallel   up     infinite    1    alloc c1
```

This command shows the state of the compute nodes, in this example:

Node c1 is powered on and has been fully allocated. Node c2 is getting started and has been allocated by one or more jobs (but not fulfilled). Nodes c3, c4 and c5 are idle and in power saving mode. Note that "~" means that is powered off (energy saving mode) and "#" that is powering on. To show the current job list use the command **squeue**:

```
> squeue
JOBID PARTITION     NAME       USER       ST   TIME     NODES
NODELIST(REASON)
621   parallel    bucleINT    mike.smith  R    18:28:08      1    c1
622   parallel    IntelMPI    david.garc  R    18:25:48      1    c1
625   serial      bucleCua    jack.sparr  R    18:20:26      1     c2
```

State "R" means that the job is running. Sometimes powered off nodes must to be started in order to process a job, the average up time is about 5 minutes so the state of the job is "CF" (configuring). Please be patient.

To show complete information of your jobs (running or not) use the following command:

```
> scontrol show job <JOBID>
```

Example:

```
> scontrol show job 622
JobId=622 JobName=IntelMPICua.sh
   UserId=david.garcia(10104) GroupId=users(1000)
   Priority=4294901708 Nice=0 Account=users QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=18:34:02 TimeLimit=3-12:00:00 TimeMin=N/A
   SubmitTime=2016-05-18T13:58:36
   EligibleTime=2016-05-18T13:58:36
   StartTime=2016-05-18T13:58:37 EndTime=2016-05-22T01:58:37
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=parallel AllocNode:Sid=titani-nx:29074
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=c1
   BatchHost=c1
   NumNodes=1 NumCPUs=12 CPUs/Task=12 ReqB:S:C:T=0:0:*:*
   TRES=cpu=12,mem=24576,node=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=12 MinMemoryCPU=2G MinTmpDiskNode=0
```

```
   Features=(null) Gres=(null) Reservation=(null)
   Shared=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/users/david.garcia/programs/IntelMPICua.sh
   WorkDir=/users/david.garcia/programs
   StdErr=/users/david.garcia//programs/slurm-622.out
   StdOut=/users/david.garcia//programs/slurm-622.out
```

5.3.4.- Deleting jobs from the queues

The command to cancel a job is **scancel**:

```
> scancel <JOBID>
```

5.4.- Changing Software Environment

As seen, different applications and compilers are installed into the TITANI cluster. Initially, when a user logs in the system Intel Compilers and MPI environment configurations are loaded and ready to be used by default.

In the future if new software versions that require new environment variables are installed it may be necessary to load new configuration profiles.

You can list the available configurations with this command:

```
> module avail
---------------------------------------- /etc/modulefiles ----------------
-------------------------------------
cmake/3.2.1   cmake/3.6.2   gcc/6.2.0   gcc/7.4.0   intel/Composer-2013
mpi/openmpi-2.0.1   mpi/openmpi-x86_64   openfoam/v1812
```

You can load one of them:

```
> module load gcc/6.2.0
```

You can back to the initial environment:

```
> module purge
```

5.5. Scientific Software

Listed below are the scientific programs installed in TITANI and the basic user manual to send jobs to the queues.

5.5.1.- ABAQUS

Abaqus is a calculation program for finite elements analysis. It is oriented to solve problems such as solid mechanics, linear and nonlinear problems in static and dynamic ranges.

Abaqus/Standard and Abaqus/Explicit are available for the analysis and Abaqus/CAE for data pre-processing and post-processing. Abaqus licenses are limited, for that reason we encourage to make responsible and moderate use of them. It is recommended **not to send more than two jobs per user using Abaqus/Standard and Abaqus/Explicit**, and just **one job for Abaqus/CAE**. If a user exceeds these limits, the system administrator is allowed to delete one of them without any previous warning.

To execute **Abaqus/CAE** and have a visualization or post processing of the obtained data, it is required a Graphical Acces (see section 2.2) to get the visualization window. Abaqus/CAE command:

```
> abaqus cae -mesa
```

To check the available licenses at any time you can use the following command:

```
> abaqus-licenses
```

To send an Abaqus job to the **serial** queue is necessary to use a shell script like this:

```
#!/bin/bash
unset SLURM_GTIDS
abaqus job=job_name input=input_filename.inp interactive
```

Shell command:

```
> sbatch [-p serial] <abaqus_script_name>
```

LAUNCHING ABAQUS PARALLEL JOB

To send to the **parallel** queue an Abaqus job that uses more than one core is necessary to use a shell script like this:

```
#!/bin/bash
#SBATCH --cpus-per-task=<NumThreads>
unset SLURM_GTIDS
abaqus job=name input=filename.inp cpus=<NumThreads> interactive
```

Shell command:

```
> sbatch -p parallel <abaqus_script_name>
```

5.5.2.- MATLAB

MATLAB is a mathematic software that offers its own programming language, M language, integrated with its development environment. A shell script is required to send jobs to the batch queues. This shell script must have the following format, where *program.m* would be the filename of the Matlab file

to execute:

```
#!/bin/bash
echo "Running on host: " `hostname`
echo "Starting MATLAB at `date`"

matlab -nodesktop -nodisplay -nosplash < program.m > program.out
echo "MATLAB run completed at `date`"
```

Shell command:

```
> sbatch [-p serial] <matlab_script_name>
```

LAUNCHING MATLAB PARALLEL JOB WITH MULTITHREADS

To send to the **parallel** queue a Matlab job that uses multithreads is necessary to create a shell script specifying the number of threads:

```
#!/bin/bash
#SBATCH --cpus-per-task=<number_of_threads>
echo "Running on host: " `hostname`
echo "Starting MATLAB at `date`"

matlab -nodesktop -nodisplay -nosplash < program.m > program.out
echo "MATLAB run completed at `date`"
```

Shell command:

```
> sbatch -p parallel <matlab_script_name>
```

LAUNCHING MATLAB EXPLICIT PARALLEL JOB

The Parallel Computing Toolbox is available on TITANI. It is possible to use up to 24 workers for shared parallel operations on a single node in the current MATLAB version. Our license does not include MATLAB Distributed Computing Server. Therefore, multi-node parallel operations are not supported.

To send to the **parallel** queue a Matlab job that uses workers (**parpool** function) is necessary to create a shell script as shown below:

```
#!/bin/bash
#SBATCH --ntasks-per-node=6 =<number_of_cores>
echo "Running on host: " `hostname`
echo "Starting MATLAB at `date`"

matlab -nodesktop -nodisplay -nosplash < program.m > program.out
echo "MATLAB run completed at `date`"
```

Shell command:

```
> sbatch -p parallel <matlab_script_name>
```

5.5.3.- OpenFOAM

OpenFOAM is the free, open source CFD software released and developed primarily by OpenCFD Ltd since 2004. It has a large user base across most areas of engineering and science, from both commercial and academic organisations. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to acoustics, solid mechanics and electromagnetics.

We need to prepare OpenFOAM environment before to use it. It is necessary to load a module and to source a script to set the needed variables:

```
$ module load openfoam/v1812
$ source $OPENFOAM_SETUP
```

It is possible to configure an alias in your `.bashrc` to do it in a single command:

```
echo "alias of1812='module load openfoam/v1812; source \$OPENFOAM_SETUP'"
>>$HOME/.bashrc
```

In this case, just login again and type `of1812` to be ready to use OpenFOAM.

Sending jobs to the queue

A shell script is required to send jobs to the batch queues. This shell script must have the following format:

```
#!/bin/bash
echo "Running on host: " `hostname`
echo "Starting OpanFOAM at `date`"

module load openfoam/v1812
source $OPENFOAM_SETUP

cd <openfoam-working-directory i.e: /users/username/OpenFOAM/username-
v1812/run/pitzDaily/>
blockMesh > mesh.out #Use here your OpenFOAM needed commands
simpleFoam > output.out
echo "OpenFOAM run completed at `date`"
```

Shell command:

```
> sbatch -p serial <openfoam_script_name>
```

OpenFOAM can also be run in parallel, but needs further configuration. This is an example to send a job to the parallel queue:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=5
echo "Running on host: " `hostname`
echo "Starting OpanFOAM at `date`"

module load openfoam/v1812
source $OPENFOAM_SETUP

cd /users/username/OpenFOAM/username-v1812/run/mesh/parallel/cavity
mpirun -np 5 Allrun > output.out
echo "OpenFOAM run completed at `date`"
```

Shell command:

```
> sbatch -p parallel <openfoam_script_name>
```

manual, titani, calculintensiu