Tirant v3

User’s Guide

Universitat de València

TIRANT supercomputador

RES

RED ESPAÑOLA DE SUPERCOMPUTACIÓN

Last update: 04/27/20

1 Introduction

This user's guide for the Tirant v3 cluster is intended to provide the minimum amount of information needed by a new user of this system. As such, it assumes that the user is familiar with many of the standard features of supercomputing as the Linux operating system.

Here you can find most of the information you need to use our computing resources and the technical documentation about the machine. Please read carefully this document and if any doubt arises do not hesitate to contact us (see section Getting Help).

This user's guide is based on the MareNostrum v4 user's guide, the main RES node. All RES nodes are similar, so users will find that it is easy to use any of them if they have prior experience with any of the machines.

2 System Overview

Tirant v3 is a supercomputer based on Intel Xeon (v1, Sandy Bridge) processors. It is an IBM system composed of rear door water cooled iDataPlex racks, a Mellanox InfiniBand FDR10 high performance network interconnect and runs OpenSuSE Leap 42.3 as operating system. Its current Linpack Rpeak performarce is 111,8 Teraflops.

This general-purpose block consists of 4 racks housing 336 nodes with a total of 5376 processor cores and ~ 10 Terabytes of main memory. Compute nodes are equipped with:

- 2 sockets Intel Xeon SandyBridge E5-2670 with 8 cores each @ 2.6GHz for a total of 16 cores per node
- 32 GB of DDR3 main memory
- 40 Gbit/s InfiniBand FDR10 PCIE card
- 1 Gbit/s Ethernet
- 500 GB local SATA HD, available as temporary storage during jobs ($TMPDIR=/scratch/tmp/[jobid])

The processors support well-known vectorization instructions such as SSE 4.2 and AVX.

2.1 Login Nodes

You can connect to the Tirant supercomputer using two public login nodes. The logins are:

```
tirant.uv.es (tirant1.uv.es or login1.uv.es)
tirant2.uv.es (login2.uv.es)
```

2.2 Password Management

The first time you log in the system, you will be **forced** to change the password. However, you can change the password at any time using the command "passwd". If you forget your password, please contact us to have it reset. You password will revert to the original password that was sent to you in the welcome email and you will need to change it again.

2.3 Transferring files

The only way to transfer files to Tirant is using *scp* or *sftp* to login nodes. You can execute the scp command *from* or *to* login nodes. It is possible to use *rsync* also. If you think that you need another way to do it, please contact us.

On a Windows system, most of the secure shell clients come with a tool to make secure copies. There are several tools that accomplish the requirements, please refer to the Appendix, where you will find the most common ones and examples of use.

2.4 Compilation for the architecture

To generate code that is optimized for the target architecture and the supported features such as SSE, MMX and AVX instruction sets you will have to use the corresponding compiler flags. For compilations of MPI applications, an MPI installation needs to be loaded in your session as well. For example Intel MPI via 'module load impi/2018.3.222'.

The latest Intel compilers provide the best possible optimizations for the Xeon architecture. That includes the main compilers (intel/2018.3.222), the Intel MPI software stack (impi/2018.3.222) and the math kernel libraries MKL (mkl/2018.3.222) in their latest versions. We highly recommend linking against MKL where supported to achieve the best performance results. An older version (2017.7.259) of the Intel compilers and libraries is available in case of problems with recent ones. With time there will be newer Intel versions available but we will default to the 2018.3.222 version.

To separately load the Intel compilers please use:

```
module load intel/2018.3.222
```

The corresponding optimization flags for icc are: CFLAGS="-O3 -xHost". As the login nodes are of the exact same architecture as the compute nodes, the -xHost flag enables all possible optimizations available on the execution hosts.

2.5 Intel Compiler Licenses

We recommend compiling using either login1 (tirant1.uv.es), login2 (tirant2.uv.es) or the partition interactive if your compilation exceeds 20 minutes. For reasons of licensing it is only possible to execute two simultaneous compilations. Should all of them be in use when trying to compile, you will experience a delay when the compiler starts and tries to checkout a license.

In this case an error message like the one below will appear. If that is the case, please wait a couple of minutes and try again.

```
fort: error #10052: could not checkout FLEXlm license
Error: A license for Comp-CL is not available (-9,57)
```

2.6 GNU compilers

The GNU compiler provided by the system is version 4.8.5. For better support of new hardware features we recommend to use the latest version that can be loaded via the modules system. Currently the latest version available in Tirant is gcc 8.1.0. Gcc and Gfortran compilers can be used loading the corresponding module:

```
module load gcc/8.1.0
```

3 File Systems

IMPORTANT: It is your responsibility as a user of our facilities to backup all your critical data. In this moment we **CAN'T** provide any backup of any file system on Tirant.

Each user has several areas of disk space for storing files. These areas may have size or time limits, please read carefully all this section to know about the usage policy of each of these file systems. There are 3 different types of storage available inside a node:

- Root file system: is the file system where the operating system resides
- LUSTRE filesystem: LUSTRE is a distributed networked filesystem which can be accessed from all the nodes
- Local hard drive: Every node has an internal hard drive

3.1 Root File system

The root file system, where the operating system is stored doesn't reside in the node, this is a NFS file system mounted from one of the servers.

As this is a remote file system only data from the operating system has to reside in this file system. It is NOT permitted the use of /tmp for temporary user data. The local hard drive can be used for this purpose as you can read just ahead.

3.2 LUSTRE File system

The LUSTRE file system is a high-performance shared-disk file system providing fast, reliable data access from all nodes of the cluster to a global file system. LUSTRE allows parallel applications simultaneous access to a set of files (even a single file) from any node that has the LUSTRE file system mounted while providing a high level of control over all file system operations.

These are the partitions of the LUSTRE file systems available in the machine from all nodes:

- /apps (or /storage/apps): Over this file system will reside the applications and libraries that have already been installed on the machine. Take a look at the directories to know the applications available for general use.

- /home (or /storage/home): This file system has the home directories of all the users, and when you log in you start in your home directory by default. Every user will have their own home directory to store their own source code and their personal data. A default quota will be enforced on all users to limit the amount of data stored there. Also, it is highly discouraged to run jobs from this file system unless you need to access little amounts of data (less than 1GB). Please run your jobs on your group's /storage/projects or /storage/scratch instead.
- /storage/projects: In addition to the home directory, there is a directory in /storage/projects for each group of users. For instance, the group "vlc00" will have a "/storage/projects/vlc00" directory ready to use. This space is intended to store data that needs to be shared between the users of the same group or project. A quota per group will be enforced depending on the space assigned by the Access Committee. It is the project's manager responsibility to determine and coordinate the better use of this space, and how it is distributed or shared between their users.
- /storage/scratch: Each user will have a directory over /storage/scratch. Its intended use is to store temporary files of your jobs during their execution. A quota per group will be enforced depending on the space assigned by the Access Committee.

3.3 Local Hard Drive

Every node has a local SATA hard drive (HDD) that can be used as a local scratch space to store temporary files during executions of one of your jobs. This space is mounted over /scratch/tmp/$JOBID directory and pointed out by $TMPDIR environment variable (and $SCRDIR in Gaussian scripts). The amount of space within the /scratch file system is about 480 GB. All data stored in these local hard drives at the compute nodes will not be available from the login nodes. You should use the directory referred to by $TMPDIR to save your temporary files during job executions. This directory will automatically be cleaned after the job finishes.

4 Running Jobs

Slurm is the utility used for batch processing support, so all jobs must be run through it. This section provides information for getting started with job execution at the cluster.

Important notice: All jobs requesting 16 or more cores will automatically use all requested nodes in exclusive mode. For example if you request 17 cores you will receive two complete nodes (16 cores * 2 = 32 cores) and the consumed runtime of these 32 cores will be reflected in your budget.

4.1 Queues

There are several queues present in the machine and different users may access different queues. All queues have different limits in amount of cores for the jobs and duration.

The standard configuration and limits of the queues are the following

Queue	Max cores per job	Cores per project	Maximum wallclock
Interactive	16 (login2)	16	12 h
UV class_t	1024	3072	72 h
UV class_u	512	2048	24h
RES class_a	1024	3072	72 h
RES class_b	1024	2048	48 h
RES class_c	512	2048	24 h

We provide a command that allows users to check the queues they have available at a given time and the limits associated with them. This command is `tirant_showq.sh` and you can use it in any of the login machines. It will give you a list of what queues can you use and which one is your default (the one that you don't need to specify when you launch your scripts). The script is useful because you can learn about the limits that each queue imposes to your jobs. There are general queue limits that apply to any jobs and specific limits to your project account or user. Usually the limits involve maximum number of CPUs or nodes that a single job can use. There are also limits about the maximum number of CPUs a project account can use. You will also be able to see how many jobs you have submitted and are running and also the total amount of jobs in the queue (including jobs from other project accounts). You can use this to have a rough idea of the general utilization that the machine has at a given time.

4.2 Submitting jobs

The method for submitting jobs is to use the SLURM **sbatch** directives directly. A job is the execution unit for SLURM. A job is defined by a text file containing a set of directives describing the job's requirements, and the commands to execute.

In order to ensure the proper scheduling of jobs, there are execution limitations in the number of nodes and cores that can be used at the same time by a group.

These are the basic directives to submit jobs with **sbatch**:

```
sbatch <job_script>
```

Submits a "job script" to the queue system (see Job directives).

```
squeue
```

Shows all the submitted jobs.

```
scancel <job_id>
```

Remove the job from the queue system, canceling the execution of the processes, if they were still running.

4.3 Interactive Sessions

Allocation of an interactive session in the interactive partition has to be done through SLURM:

```
salloc -partition=interactive -qos=interactive
```

This will create a job on the interactive qos just executing a shell. This allows the user to execute small processes that need more than 10 minutes of CPU, like long compilations or file transfers.

4.4 Job directives

A job must contain a series of directives to inform the batch system about the characteristics of the job. These directives appear as comments in the job script and have to conform to the sbatch syntax (you can also pass them directly in the sbatch command).

Sbatch syntax is of the form:

```
#SBATCH --directive=value
```

Additionally, the job script may contain a set of commands to execute. If not, an external script may be provided with the 'executable' directive. Here you may find the most common directives:

- `#SBATCH --qos=QUEUE_NAME`

To request the queue for the job. If it is not specified (preferred way), Slurm will use the user's default queue.

- `#SBATCH --time=DD-HH:MM:SS`

The limit of wall clock time. This is a mandatory field and you must set it to a value greater than real execution time for your application and smaller than the time limits granted to the user by the queue. Notice that your job will be killed after the time has passed.

- `#SBATCH --workdir=PATHNAME`

The working directory of your job (i.e. where the job will run). If not specified, it is the current working directory at the time the job was submitted.

- `#SBATCH --error=FILE`

The name of the file to collect the standard error output (stderr) of the job.

- `#SBATCH --output=FILE`

The name of the file to collect the standard output (stdout) of the job.

- `#SBATCH --ntasks=NUMBER`

The number of processes to start. Optionally, you can specify how many threads each process would open with the directive.

- `#SBATCH --cpus-per-task=NUMBER`

The number of cores assigned to the job will be the total_tasks number * cpus_per_task number.

- `#SBATCH --tasks-per-node=NUMBER`

The number of tasks assigned to a node.

- `#SBATCH --array=<indexes>`

Submit a job array, multiple jobs to be executed with identical parameters. The indexes specification identifies what array index values should be used.

Multiple values may be specified using a comma separated list and/or a range of values with a "-" separator. Job arrays will have two additional environment variable set. 'SLURM_ARRAY_JOB_ID' will be set to the first job ID of the array. 'SLURM_ARRAY_TASK_ID' will be set to the job array index value. For example:

```
sbatch --array=1-3 job.cmd
Submitted batch job 36
```

Will generate a job array containing three jobs and then the environment variables will be set as follows:

```
# Job 1
SLURM_JOB_ID=36
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=1

# Job 2
SLURM_JOB_ID=37
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=2

# Job 3
SLURM_JOB_ID=38
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=3
```

Here you will find some interesting environmental variables:

Variable	Meaning
SLURM_JOBID	Specifies the job ID of the executing job
SLURM_NPROCS	Specifies the total number of processes in the job
SLURM_NNODES	Is the actual number of nodes assigned to run your job
SLURM_PROCID	Specifies the MPI rank (or relative process ID) for the current process. The range is from 0-(SLURM_NPROCS–1)
SLURM_NODEID	Specifies relative node ID of the current job. The range is from 0-(SLURM_NNODES–1)
SLURM_LOCALID	Specifies the node-local task ID for the process within a job

For more information:

```
man sbatch
man srun
man salloc
```

4.5 Examples

You will find this examples on /storage/apps/SLURM_EXAMPLES, ready to copy, modify and use.

Example for a sequential job:

```
#!/bin/bash
#SBATCH --job-name="test_serial"
#SBATCH --workdir=.
#SBATCH --output=serial_%j.out
#SBATCH --error=serial_%j.err
#SBATCH --ntasks=1
#SBATCH --time=00:02:00
./serial_binary> serial.out
```

Examples for some parallel jobs:

- Running a pure OpenMP job on one Tirant node using 16 cores on the interactive queue:

```
#!/bin/bash
#SBATCH --job-name=OpenMP
#SBATCH --workdir=.
#SBATCH --output=omp_%j.out
#SBATCH --error=omp_%j.err
#SBATCH --cpus-per-task=16
#SBATCH --ntasks=1
#SBATCH –time=00:10:00
#SBATCH --partition=interactive
#SBATCH --qos=interactive
./openmp_binary
```

- Running on two Tirant nodes using a pure MPI job

```
#!/bin/bash
#SBATCH --job-name=MPI
#SBATCH --output=mpi_%j.out
#SBATCH --error=mpi_%j.err
#SBATCH --ntasks=32
srun ./mpi_binary
```

- Running a hybrid MPI+OpenMP job on two Tirant nodes with 16 MPI tasks (8 per node), each using 2 cores via OpenMP:

```
#!/bin/bash
#SBATCH --job-name=Hybrid
#SBATCH --workdir=.
#SBATCH --output=mpi_%j.out
#SBATCH --error=mpi_%j.err
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=2
```

```
#SBATCH --tasks-per-node=8
#SBATCH --time=00:02:00
srun ./parallel_binary> parallel.output
```

4.6 Resource usage and job priorities

Projects will have assigned a certain amount of compute hours or core hours that are available to use. One core hour is the computing time of one core during the time of one hour. That is a full node with 16 cores running a job for one hour will use up 16 core hours from the assigned budget. The accounting is solely based in the amount of compute hours used.

The priority of a job and therefore its scheduling in the queues is being determined by a multitude of factors. The most important and influential ones are the fairshare in between groups, waiting time in queues and job size. Tirant is a system meant for and favoring large executions so that jobs using more cores have a higher priority. The time while waiting in queues for execution is being taken into account as well and jobs gain more and more priority the longer they are waiting. Finally our queue system implements a fairshare policy between groups. Users who did not run many jobs and consumed compute hours will get a higher priority for their jobs than groups that have a high usage. This is to allow everyone their fair share of compute time and the option to run jobs without one group or another being favored. You can review your current fair share score using the command

```
sshare -la
```

5 Software Environment

All software and numerical libraries available at the cluster can be found at /apps/. If you need something that is not there please contact us to get it installed.

5.1 Modules Environment

The **lmod** environment modules package provides a dynamic modification of a user's environment via modulefiles. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically, in a clean fashion. All popular shells are supported, including bash, ksh, zsh, sh, csh, tcsh, as well as some scripting languages such as perl.

Installed software packages are divided into five categories:

- Environment: modulefiles dedicated to prepare the environment, for example, get all necessary variables to use openmpi to compile or run programs
- Tools: useful tools which can be used at any time (php, perl, …)
- Applications: High Performance Computers programs (GROMACS, …)
- Libraries: Those are tipycally loaded at a compilation time, they load into the environment the correct compiler and linker flags (FFTW, LAPACK, …)
- Compilers: Compiler suites available for the system (intel, gcc, …)

Modules can be invoked in two ways: by name alone or by name and version. Invoking them by name implies loading the *default* module version. This is usually the most recent version that has been tested to be stable (recommended) or the only version available.

```
% module load intel
```

Invoking by version loads the version specified of the application. As of this writing, the previous command and the following one load the same module.

```
% module load intel/2018.3.222
```

The most important commands for modules are these:

- `module list`: shows all the loaded modules
- `module avail`: shows all the modules the user is able to load
- `module purge`: removes all the loaded modules
- `module load <modulename>`: loads the necessary environment variables for the selected modulefile (PATH, MANPATH, LD_LIBRARY_PATH…)
- `module unload <modulename>`: removes all environment changes made by module load command
- `module switch <oldmodule> <newmodule>`: unloads the first module (oldmodule) and loads the second module (newmodule)

You can run "`module help`" any time to check the command's usage and options or check the module(1) manpage for further information.

5.2 C Compilers

In the cluster you can find these C/C++ compilers:

icc / icpc -> Intel C/C++ Compilers

```
% man icc
% man icpc
```

gcc /g++ -> GNU Compilers for C/C++

```
% man gcc
% man g++
```

All invocations of the C or C++ compilers follow these suffix conventions for input files:

.C, .cc, .cpp, or .cxx -> C++ source file.
.c -> C source file
.i -> preprocessed C source file
.so -> shared object file
.o -> object file for ld command
.s -> assembler source file

By default, the pre-processor is run on both C and C++ source files. These are the default sizes of the standard C/C++ datatypes on the machine

Default datatype sizes on the machine

Type	**Length (bytes)**
bool (c++ only)	1
char	1
wchar_t	4
short	2
int	4
long	8
float	4
double	8
long double	16

To compile **MPI** programs it is recommended to use the following handy wrappers: mpicc, mpicxx for C and C++ source code. You need to choose the Parallel environment first: `module load openmpi / module load impi`. These wrappers will include all the necessary libraries to build MPI applications without having to specify all the details by hand.

```
% mpicc a.c -o a.exe
% mpicxx a.C -o a.exe
```

OpenMP directives are fully supported by the Intel C and C++ compilers. To use it, the flag -qopenmp must be added to the compile command line.

```
% icc -qopenmp -o exename filename.c
% icpc -qopenmp -o exename filename.C
```

You can also mix **MPI + OPENMP** code using -openmp with the mpi wrappers mentioned above.

5.3 FORTRAN Compilers

In the cluster you can find these compilers:

ifort -> Intel Fortran Compilers

```
% man ifort
```

gfortran -> GNU Compilers for FORTRAN

```
% man gfortran
```

By default, the compilers expect all FORTRAN source files to have the extension ".f", and all FORTRAN source files that require pre-processing to have the extension ".F". The same applies to FORTRAN 90 source files with extensions ".f90" and ".F90".

In order to use **MPI**, again you can use the wrappers mpif77 or mpif90 depending on the source code type. You can always `man mpif77` to see a detailed list of options to configure the wrappers, ie: change the default compiler.

```
% mpif77 a.f -o a.exe
```

OpenMP directives are fully supported by the Intel Fortran compiler when the option "-qopenmp" is set:

```
% ifort -qopenmp
```

6 Getting Help

UV provides users with consulting assistance. User support consultants are available during normal business hours, Monday to Friday, 09 a.m. to 15 p.m. (CEST time).

User questions and support are handled at: **res_support@uv.es**

If you need assistance, please supply us with the nature of the problem, the date and time that the problem occurred, and the location of any other relevant information, such as output files. Please contact us if you have any questions or comments regarding policies or procedures or you think you need help with your project.

7 Appendices

7.1 SSH

SSH is a program that enables secure logins over an insecure network. It encrypts all the data passing both ways, so that if it is intercepted it cannot be read. It also replaces the old an insecure tools like telnet, rlogin, rcp, ftp, etc. SSH is a client-server software. Both machines must have ssh installed for it to work.

We have already installed a ssh server in our machines. You must have installed an ssh client in your local machine. SSH is available without charge for almost all versions of UNIX (including Linux and MacOS X) and recent versions of Windows 10. For UNIX and derivatives, we recommend using the OpenSSH client, downloadable from: http://www.openssh.org, and for Windows users we recommend the built-in client in recent versions or using free SSH clients like Putty (can be downloaded from: http://www.putty.org). Any client compatible with SSH version 2 can be used. MobaXterm is also a great option.

No matter your client, you will need to specify the following information:

- Select SSH as default protocol
- Select port 22
- Specify the remote machine and username

7.2 Transferring files

To transfer files to or from the cluster you need a secure copy (scp) client. There are several different clients, but as previously mentioned, we recommend using Putty clients for transferring files: pscp. You can find it at the same web page as Putty (http://www.putty.org).

Some other possible tools for users requiring graphical file transfers could be:

- WinSCP: Freeware Sftp and Scp client for Windows (http://www.winscp.net)
- MobaXterm: available personal license (https://mobaxterm.mobatek.net)

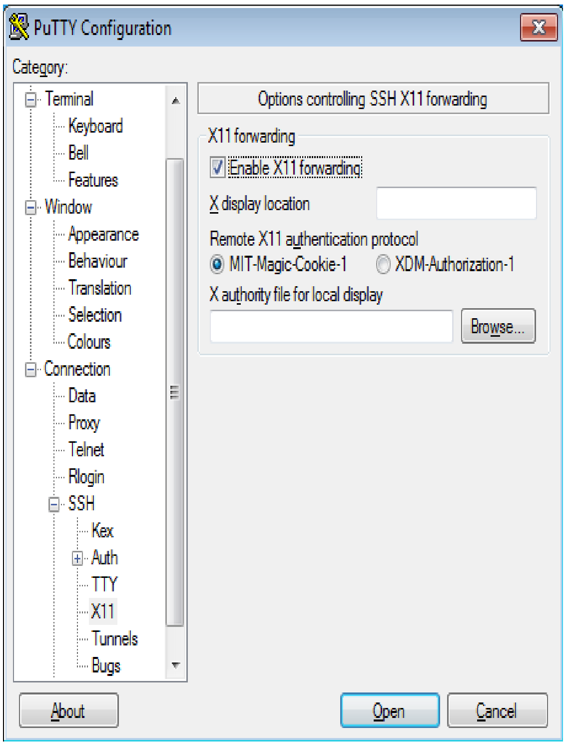
7.3 Using X11

In order to start remote X applications you need and X-Server running in your local machine. Here is a list of most common X-servers for windows:

- Cygwin/X: http://x.cygwin.com
- VcXsrv: https://sourceforge.net/projects/vcxsrv/ (similar to XQuartz in Mac OS X). Can be used in conjunction with WSL linux distros running in Windows (you can get them in the Microsoft Store).
- X-Win32: http://www.starnet.com
- WinaXe: http://labf.com
- XconnectPro: http://www.labtam-inc.com
- Exceed: http://www.hummingbird.com

Both Cygwin/X and VcXsrv are open source and free X servers, you need to pay for the others.

Once the X-Server is running run putty with X11 forwarding enabled:



MobaXterm (https://mobaxterm.mobatek.net) allows to use X11. It is a great tool, but using full features require to purchase a license.