LluisVives v2

User’s Guide

1 Introduction

This user's guide for the LluisVives v2 (LV2) 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 Tirant v3 user's guide, because they are very similar.

2 System Overview

LV2 is a supercomputer based on Intel Xeon Skylake processors. It is a ATOS system composed of rear door water cooled racks, a Mellanox InfiniBand EDR high performance network interconnect and running CentOS 7.5 as operating system. Its current Linpack Rpeak performarce is 158 Teraflops.

LV2 is a hybrid supercomputer. It has two different partitions: thin and fat. The hardware in each partition is different, allowing users to execute a great variety of simulations.

The THIN block consists of 20 nodes with a total of 720 processor cores and ~ 3.84 Terabytes of main memory. Compute nodes are equipped with:

- 2 sockets Intel Xeon Skylake Gold 6154 CPU @ 3.00GHz with 18 cores each, for a total of 32 cores per node
- 192 GB of main memory DDR4
- 100 Gbit/s InfiniBand EDR PCI card
- 1 Gbit/s Ethernet
- 2 TB local SATA HD, available as temporary storage during jobs ($TMPDIR=/scratch/tmp/[jobid])

The FAT block consists of 2 nodes with a total of 144 processor cores and ~ 3.00 Terabytes of main memory. Compute nodes are equipped with:

- 4 sockets Intel Xeon Skylake Gold 6154 CPU @ 3.00GHz with 18 cores each, for a total of 72 cores per node
- 1584 GB of main memory DDR4
- 100 Gbit/s InfiniBand EDR PCI card
- 1 Gbit/s Ethernet
- 2 TB local SATA HD, available as temporary storage during jobs ($TMPDIR=/scratch/tmp/[jobid])

The processors support well-known vectorization instructions such as SSE4 and AVX512.

2.1 Login Node

You can connect to LV2 supercomputer using one public login nodes, named *vives*:

vives.uv.es (`lluisvives.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 be changed it again.

2.3 Transferring files

The only way to transfer files to LluisVives is using scp or sftp to login nodes. You can execute the scp command *from* or *to* login nodes. 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 SSE4 and AVX-512 instruction sets you will have to use the corresponding compile 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 is the 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.

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 vives.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 so 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. Currently the latest version available in LluisVives is gcc 8.2. *Gcc* and *Gfortran* compilers can be used loading the corresponding module:

```
module load gcc/8.2
```

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 **ONLY** provide a daily backup of /home directories on LluisVives.

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 policy of usage of each of these file systems. There are 4 different types of storage available inside a node:

- Root file system: is the file system where the operating system resides
- NFS filesystem: is the file system where the /home of all users 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 on RAM, as nothing is installed on each node.

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 could read just ahead.

3.2 NFS File-system

/home: this file system is NFS mounted from central storage of SIUV. Every user will have their own home directory to store own developed sources and their personal data. There is a daily backup of this mount point. A default quota will be enforced on all users to limit the amount of data stored there (10 GB). Also, it is highly discouraged to run jobs from this file system, because of the low performance of NFS protocol. Please run your jobs on your group's /storage/projects or /storage/scratch instead

3.3 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.
- /storage/projects: In addition to the home directory, there is a directory in /storage/projects for each group of users. For instance, the group "lv00" will have a "/storage/projects/lv00" 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 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. No quotas are enforced in this area.

3.4 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 1800 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.

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. The queue thin is the default and only public queue, so it is not necessary to specify it.

Queue	Max cores per job	Cores per project	Maximum wallclock
interactive	16 (vives)	16	30 h
thin	180	240	96 h
fat	72	72	60 d

Please remember: access to the fat queue on fat partition is not allowed to all users, as the resources available on this partition are reserved to execute programs with special memory needs. If you have one of this problems, please contact us and provide as much information as you can

about yours needs and a way to test the program and check the use of memory.

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 `lv_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 sh
```

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 either the sbatch syntax.

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 LluisVives 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 LluisVives nodes using a pure MPI job

```
#!/bin/bash
#SBATCH --job-name=MPI
#SBATCH --output=mpi_%j.out
#SBATCH --error=mpi_%j.err
#SBATCH --ntasks=72
srun ./mpi_binary
```

- Running a hybrid MPI+OpenMP job on one LluisVives node with 18 MPI tasks, 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=36
#SBATCH --cpus-per-task=2
#SBATCH --tasks-per-node=18
#SBATCH --time=00:02:00
srun ./parallel_binary> parallel.output
```

4.6 Resource usage and job priorities

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. LluisVives 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. All users are classified in two groups. One group has a higher priority than the other. Between the members of each group, resources are shared equally.

As this system was initially 50% financed between a group of users and the University of Valencia, it was decided that the members of the group of users who signed the application for funding and supported it in a coordinated manner would form part of that group with the highest priority.

The rest of UV users would have access to the system, but in the lowest priority group. The fundamental difference between both groups occurs in situations of competition for resources and translates into a longer queue time.

A mechanism has been provided to be able to move the high priority group. It is based on making some type of investment in the cluster, increasing its power for the entire UV. However, since no one has shown interest, the regulations for this mechanism have not yet been established. Do not hesitate to getting in contact if this is your case.

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: **calculo@uv.es** or **calcul@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 a ssh client in your local machine. SSH is available without charge for almost all versions of UNIX (including Linux and MacOS X). For UNIX and derivatives, we recommend using the OpenSSH client, downloadable from: http://www.openssh.org, and for Windows users we recommend using Putty, a free SSH client that can be downloaded from: http://www.putty.org. Otherwise, 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 of 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)

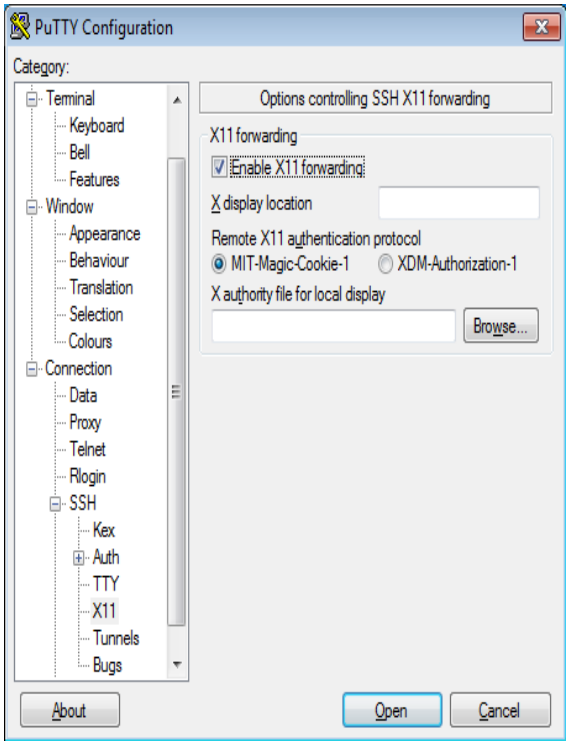
7.3 Using X11

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
- X-Win32: http://www.starnet.com
- WinaXe: http://labf.com
- XconnectPro: http://www.labtam-inc.com
- Exceed: http://www.hummingbird.com

The only Open Source X-server listed here is Cygwin/X, 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 full features required to purchase a licence.