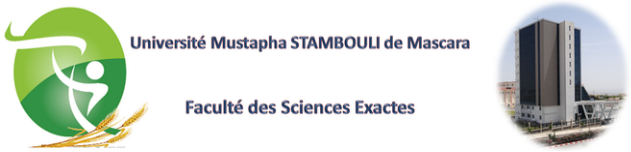


People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University Mustapha Stambouli of Mascara
Faculty of Exact Sciences

HPC Cluster Emir

A short introduction to high performance computing

Presented by

Benaoumeur Bakhti

Supercomputing Center: University of Mascara

Mascara 2023

Contents

1 Introduction

This document is intended for anyone who is new to using the High Performance Computing (HPC) Cluster or Supercomputer and wants to work with the HPC Cluster **Emir** of the University of Mascara. It is also intended for students and researchers of other universities in Algeria as all the existing Clusters in Algeria (belonging to the CERIST) share the same architecture and all are using SLURM (see sections below) for cluster resource management. The document can be helpful and may also contain important information for experienced users who are already familiar with the subject. This guide does not claim to be complete, but rather should, on the contrary, enable the simplest possible introduction to work with Clusters. Further extension of this guide is a work in progress and will be shared in the near future.

The cluster showcased here is the HPC Cluster Emir of the University of Mascara. An important thing to note here is that an HPC may not be faster than one small single computer only if cluster parallelism is exploited. The latter becomes a hot topic in modern computing and groundbreaking scientific discoveries have been made based on using the HPC. Thus, people are strongly encouraged to take part in this development. The strength of the HPC cluster lies in the large number of tasks that can be performed simultaneously. Parallelism can be done either via programming many processors (CPU: Central Processing Unit) using MPI (Message Passing Interface) or OpenMP (Open Multi-Processing) or via parallel programming of the GPU (Graphics Processing Unit) of the graphic card using either CUDA (Compute Unified Device Architecture) or OpenCL (Open Computing Language). CUDA works only with Nvidia graphic cards, however, programming with OpenCL has been implemented on a vast array of graphic cards including AMD, Intel, and Nvidia GPUs. The programming language itself is not the aim of this tutorial but the reader is advised to check the large amount of information, examples, and tutorials on the different programming languages on the web.

Due to the necessity of High Performance Computing, almost all software in physics, chemistry, biology or engineering have been extended to work on HPC clusters and parallel computers. Thus, this tutorial is intended to provide the teachers/researchers and students of the University of Mascara with

the necessary tools to use correctly the HPC Cluster Emir.

Finally, if you have any questions or suggestions for improvement or if you find errors in the text, please do not hesitate to send me an email at benaoumeur.bakhti@univ-mascara.dz or at hpc.emir@univ-mascara.dz.

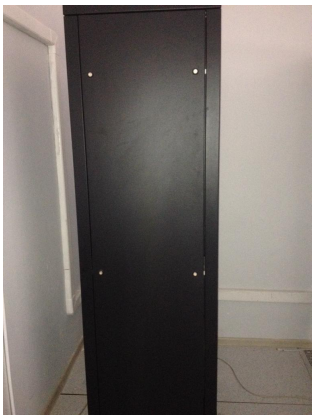
Have fun and success working with the Cluster

Mascara 2023 B. Bakhti

HPC Cluster

Voltage stabilizer

UPS



Cooling system

2 What is an HPC Cluster?

2.1 Cluster architecture

An HPC cluster or a Supercomputer is a set of computers (or servers), called nodes, connected to each other via a fast interconnection network and share a common storage unit. A simple schematic of an HPC cluster is shown in Fig. (1). In real clusters, the architecture is optimized in terms of communication speeds between nodes, storage capacity, processor and graphics card performance, and space occupied by the cluster.

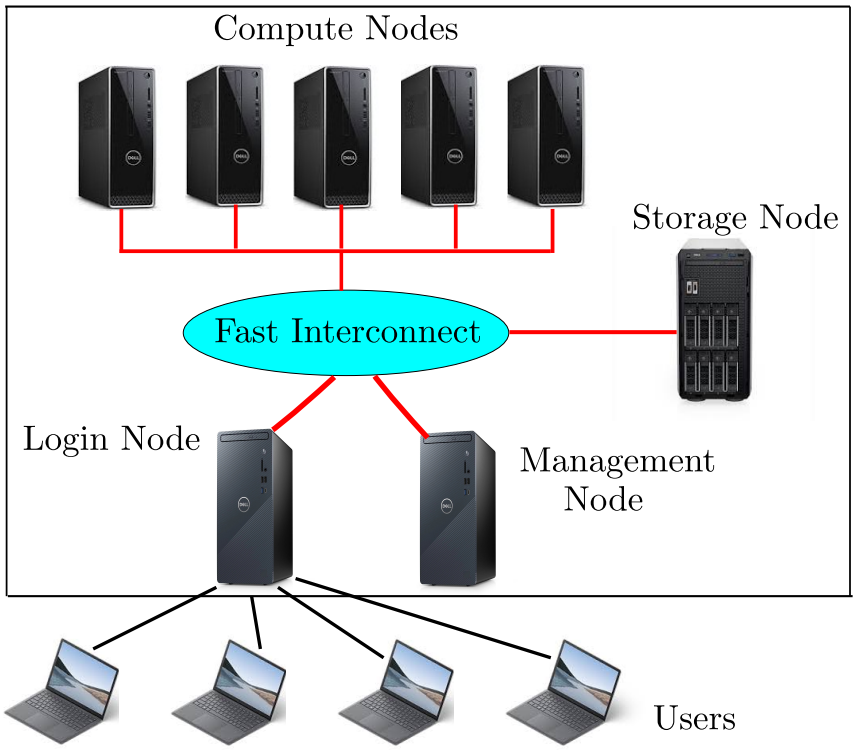


Figure 1: A simple architecture of an HPC Cluster.

2.2 Cluster nodes

Depends on the type of tasks to be done on the HPC cluster, the later can be configured with different types of nodes such as:

- Login node: it provides an external interface to the HPC cluster. Users wishing to run their jobs can access the Cluster only via the login node. The latter is intended for basic tasks such as uploading or retrieving data to or from the cluster. It can also be used for editing scripts, submitting, checking, and monitoring jobs, and so on. The login node can be used (though it's not recommended) to run small jobs that do not require a lot of computational resources.

- Administration or management node: it controls and performs the housekeeping for the cluster and in many cases, it serves also as the login node.

- Storage node: Most of the data in an HPC cluster is stored in a separate unit called the storage node which contains many drives configured using RAID technology (Redundant Array of Independent Disks). The storage space of the storage node is shared with all other nodes in the cluster. The data is stored or retrieved from the storage node across networks using the network file-sharing protocol NFS.

- Compute nodes: There is a large number of compute nodes in a typical cluster. Compute nodes provide computational resources such as processors cores, storage, RAM, and GPUs required for computations and they are the nodes on which work runs.

- Visualization node: it allows processing, monitoring, and visualizing data using 2D and 3D graphical applications.

- Virtualization node: it allows using virtualization technology such as containers in which user environments can be separated from the provided machine hardware, operating system, or software setup on a cluster.

2.3 Compute power

The performance or Computing power of an HPC cluster is usually expressed in Flop/second (or Flops) number of floating point operations per second. A simple example of a floating point operation is the multiplication (or addition) between two floating numbers 2.278*5.106. In the HPC world, the most used multiples of the flop are gigaFlops ($1GFlops = 10^9 Flops$), teraFlops

($1TFlops = 10^{12}Flops$), petaFlops ($1PFlops = 10^{15}Flops$), and exaFlops ($1EFlops = 10^{18}Flops$). Fig. (2) shows the Fugaku supercomputer which was the most powerful cluster in 2020/2021. It has been built by Fujitsu at the Riken Center for Computational Science in Kobe, Japan and it has a speed of 1.42 exaFlops.

Figure 2: Fujitsu Fugako cluster with 1.42 exaFlops computing power was selected as the most powerful HPC cluster in the world in 2020/2021 (Image from Fugaku).

Starting from May 2022, the most powerful supercomputer in the world is the Frontier which is hosted at the Oak Ridge Leadership Computing Facility (OLCF) in the USA and has a speed of 1.685 exaFlops.

2.4 Fast Interconnect Network

The communication between nodes is done via a hardware technology called the interconnect network. The communication strategy impact directly the performance of the cluster on one hand in terms of the speed at which the nodes process data between them and on the other hand on how fast they reach the shared storage space. Interconnect performance is measured with two factors: bandwidth and latency. Bandwidth is the rate at which data can be moved between nodes and is measured in megabytes (MB) per second. Latency is defined as the time spent in setting up access to a remote node so that communications can occur. There are many interconnect technologies used in the HPC cluster such as the Ethernet and the InfiniBand. While

the former is widely used, it is well known that its underlying protocol has inherent limitations preventing low-latency deployments in HPC clusters. A better solution is the Infiniband technology which features very high throughput and very low latency and it allows for very fast (today Nvidia Mellanox NDR Fig. (3) and can deliver up to $400GB/s$ throughput) communications between the nodes.

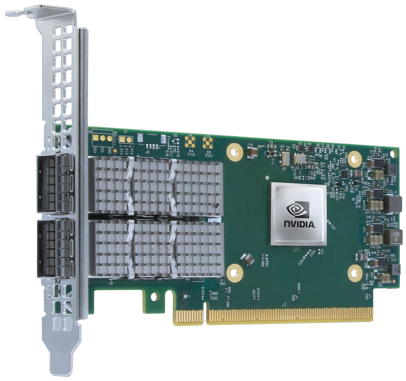


Figure 3: Nvidia Mellanox NDR InfiniBand Adapter (Image from Nvidia Mellanox).

2.5 External infrastructures

High availability of an HPC cluster requires running the nodes at 100But working continuously and effectively creates two issues. On one hand, HPC clusters consume a large amount of power densities of up to 100kW per rack, and sometimes higher. On the other hand part of the consumed power is transformed into heat energy that can harm the system. The solution for the first issue is to use UPS systems and power generators while overheating requires a specialized and highly efficient cooling system.

2.5.1 Cooling systems

Depends on the density of the HPC configration, there are four main technologies available for cooling, and selecting the right option is as important as choices around hardware and software. These technologies air cooling, water cooling, rear door heat exchangers (RDHX), and finaly Immersion cooling.
Air cooling is the most simple, cheaper and it relies on fans to carry heat away from components. But it can't keep up with the growing density of

computing environments. In addition, cooling an HPC system requires high-speed fans which generate significant noise.
Because of this, liquid or water cooling becomes the standard and most popular technology used in HPC today. Here, water is pumped through a closed loop passing through the HPC cluster and as the liquid heats up, it circulates and moves away from the hot components, creating a fow that keeps the component cool. Water cooling technology provides a good balance between performance and cost of set-up and infrastructure.
RDHX is a mixture of air and water cooling device that is attached to the back of the racks of the cluster. The HPC cluster exhausts hot air through the heat exchanger of the RDHX. The heated liquid passes through a cooling towers connected to an external chilled water system. This will cool the air back to the HPC cluster.
Immersion cooling directly immerses electronic parts in a non-conductive liquid (typically oil) without risk of electrical conductance across the computer circuits. This technology support increasingly dense HPC configurations but on the other hand it requires huge modifications in the HPC installations and also it can make maintenance of changing components very tricky. An example of air and water cooling device is shown in Fig. (4)

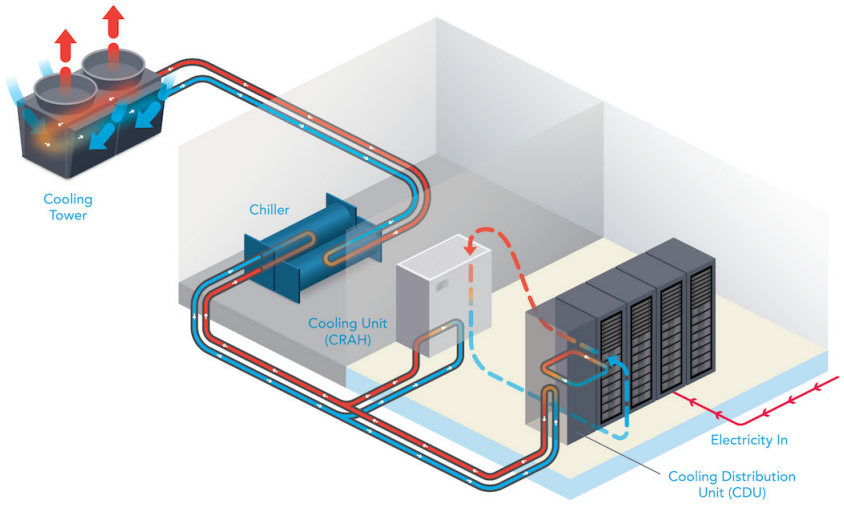


Figure 4: RDHX cooling system which combines air and liquid cooling devices (Image from upsite).

2.5.2 UPS Sysetms

A UPS (Uninterruptible Power Supply) is a battery backup that supplies power to the HPC cluster in case of power outage or voltage drop. The UPS system must provide enough time so that work and data can be saved, and then all applications and operating systems cleanly and securely closed. This prevents equipment failures that sometimes happen when the power is rapidly lost.

A UPS unit alone will only keep the nodes powered for a limited amount of time, usually less than 30 minutes. To extend this period, the UPS system can be supported by a power generator (such as a diesel generator) that can kick on in case the power grid goes offline. The UPS backup time must be large enough for the power generator to start and reach its steady-state regime.

3 A brief overview of parallel programming

In this section, we give a very short introduction to parallel programming models and the different parallel architectures. However, the section is not aimed at providing a comprehensive presentation of parallel programming or how to do parallel programming using different programming languages. Instead, we refer to some useful books that should be read (with practice) to build up effective programming skills

3.1 What is parallel programming?

Parallel programming or similarly parallel computing or parallel processing Fig. (5) is a process in which a large problem is decomposed into smaller independent tasks that can be executed simultaneously on different processors. This is to be distinguished from the standard serial or sequential programming Fig. (6) in which the problem is divided into a discrete series of instructions executed sequentially on a single processor.

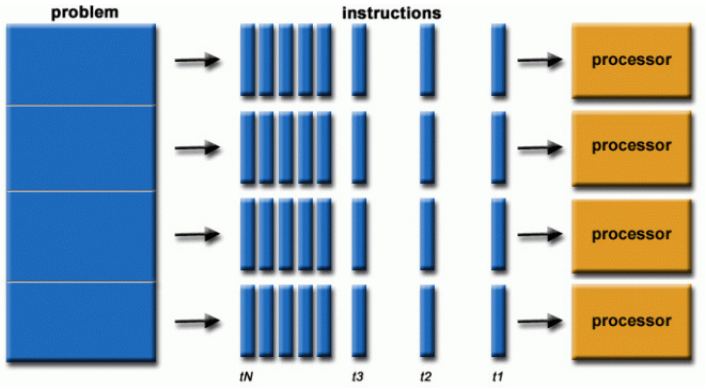


Figure 5: Parallel Programming: A large problem is subdivided into four parts and each part is executed on a single processor (Image from Lawrence Livermore National Laboratory).

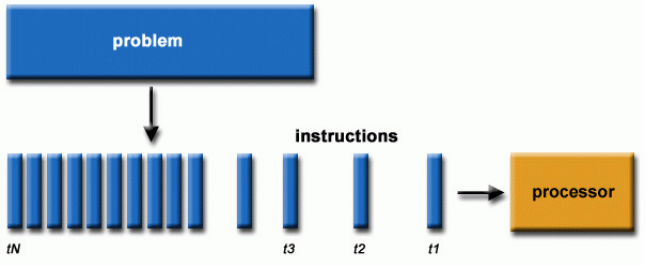


Figure 6: Serial Programming: The problem is divided into instructions executed sequentially by a single processor(Image from Lawrence Livermore National Laboratory).

3.2 Why parallel programming?

High performance parallel computers have played crucial roles basically in all research areas both in academia and industry. HPC becomes indispensable if the problem at hand is very complex, expensive, or dangerous for example in the case of weather forecasting, drug discovery, and airplane incidents simulations respectively.

Today, real-world problems are massively large and complex and this stems from the fact that for real-life problems, the number of particles involved in computer simulations is very large and the simulation requires

a very long time (months or even years). Three examples are shown in Fig. (29).

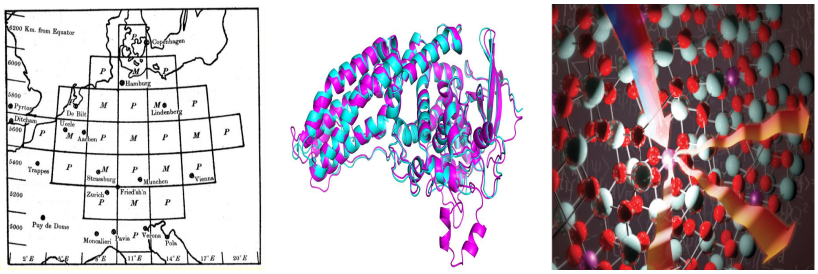


Figure 7: Three examples where parallel programming is highly required.

The first example on the left is weather forecasting. The simulation starts by discretizing the space into the three-dimensional grid. Each point on the grid stores the current atmospheric conditions (of the molecule of the air) such as temperature, pressure, wind speed and direction, humidity, and so on. The dynamic at each point is described by equations from physics and fluid dynamics. Two factors make the problem of forecasting very complex. On one hand, to increase the accuracy of the simulation, we need to increase the number of grid points and this results in an astronomical number of equations. On the other hand, because the evolution at each point depends on the states of neighboring points, we end up with very complicated nonlinear partial differential equations that require supercomputers to solve them.

The second example is the protein folding and protein-ligands banding processes that are very useful for enzyme regulation and drug discovery in general. For this system, both the number of constituents (amino acids) and a number of microscopic configurations (folded states of the protein) is very large which makes the use of massively parallel programming indispensable.

The last example is a simulation in physics or chemistry where in sequential programming the number of atoms or molecules is taken from tens to hundreds but in a massively parallel simulation the number of particles reaches hundreds of billions (for example in Monte Carlo or Molecular dynamics simulations).

3.3 Paralle Programming: Models and Languages

It is well known that for sequential programming, the von Neumann model is the predominant programming model. however, in parallel programming, there are many parallel models out there each associated with a different hardware architecture. We briefly review here four different models (that can be used in the HPC-Cluster EMIR): Shared Memory Programming, Distributed Memory Programming, GPU Programming, and finally Hybrid Programming.

3.3.1 Shared Memory Programming

In the shared-memory architecture, all the CPU cores are connected to the same piece of memory Fig (8) via a fast interconnection network called a memory bus. This allows any processor to access any part of the memory in parallel. For example, in a system with a quad-core processor and 4 GBytes of RAM, each of the 4 CPU cores will be connected to the same 4 Gbytes of RAM.
There are many programming models to program the shared-memory systems, the most used are:

- OpenMP which stands for Open Multi-Processing,
- PThreads which stands for POSIX Threads,
- Cilk, Cilk++ and OpenCilk,,
- Linda
- Split-C

OpenMP becomes the de facto standard of parallel programming for shared-memory systems.

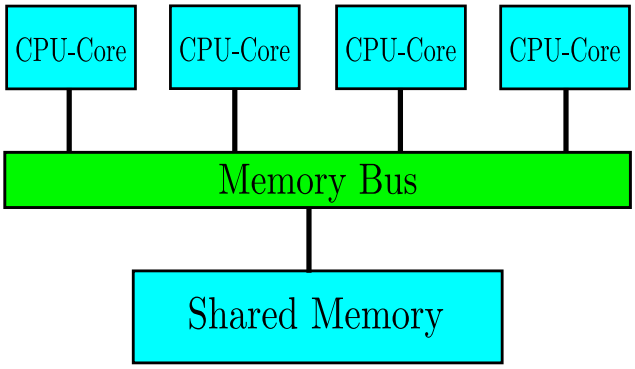


Figure 8: Shared memory architecture.

3.3.2 Distributed Memory Programming

The shared memory model is powerful and largely used but its main drawback is that it is very difficult to have very large numbers of CPU cores in a single shared-memory computer. In modern supercomputers, another highly used architecture is the distributed memory architecture in which each processor has its own private memory Fig. (10).

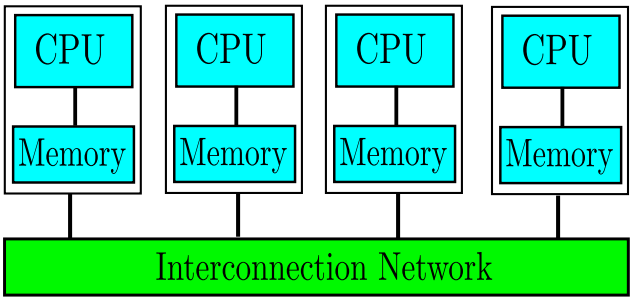


Figure 9: Distributed memory architecture.

Processors cannot access the memory of each other directly. But as shown in Fig. (10) they are able to talk to each other via the interconnection network (Memory Bus). All communication and synchronization between processors happen via messages passing through this interconnection network. In another word, if a processor needs to update the data stored in the memory of another processor, then the former sends a message to the latter asking for the data. As opposed to the shared memory model, the distributed memory

architecture allows to scale the supercomputers to a larger number of processors. In addition, because of the use of local memory for each processor, the memory access can achieve a lower access time. The main model for the parallel processing of distributed memory systems is the Message Passing Interface (MPI) and it becomes the paradigm model for parallel programming using message passing. The most popular message passing models are:

- MPI for C, C++ and Fortran,
- Akka exists for both Java and Scala,
- Actor,
- Occam,
- Fortran M.

3.3.3 GPU Programming

Modern GPU architecture Fig. (10) makes it suitable for parallel processing. In modern supercomputers, GPU have become an extremely powerful tool for high-performance computing applications. Today, GPU programming is used in all industries such as gaming, medical imaging, image processing, audio signal processing, aerospace, etc. It has also boosted the development and applications of Artificial Intelligence in industry.
Different parallel programming languages have been developed to program the GPU such as CUDA (available for Nvidia hardware only), OpenACC (for Cray hardware only), and OpenCL (hardware independent). As Nvidia GPU becomes the most popular and commonly used chipset (especially in HPC clusters and workstations), Nvidia CUDA becomes the dominent language for programming the GPU.

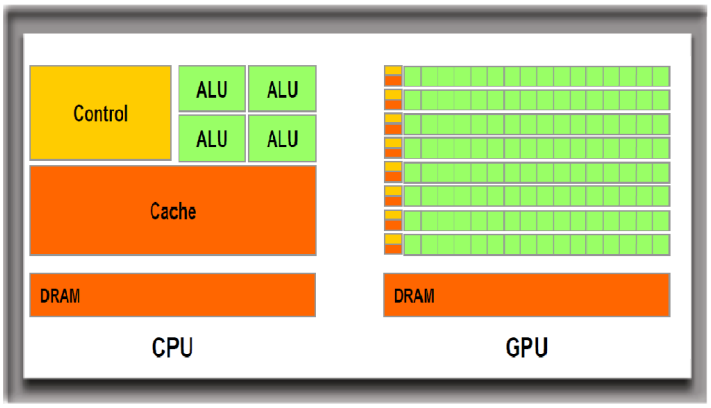


Figure 10: Comparison between the CPU and GPU architectures (Image from Nvidia).

The most used programming languages for the GPU are:

- CUDA: Compute Unified Device Architecture,
- OpenCL: Open Computing Language,
- OpenACC: Open Accelerators,
- C++ AMP: C++ Accelerated Massive Parallelism,
- Halide,
- HIP: Heterogeneous-Computing Interface for Portability (for AMD GPU),
- SYCL: can be compiled and executed on different GPUs.

In addition to the programming languages, there are many parallel libraries and APIs (such as OpenGL, DirectX, Hadoop, etc.) that facilitate the use of parallel processing.

3.3.4 Hybrid Programming

Despite that both the CPU and the GPU are indispensable computing engines, they both have advantages and disadvantages. The CPU on one hand is flexible and can provide more speed for computations (performs sequential tasks quicker) compared to the GPU, but it cannot carry a heavy load.

On the other hand, because the GPU consists of hundreds (or thousands) of cores, it can accelerate the workloads and hence processes data several orders of magnitude faster than a CPU. But the GPU cores are less powerful than their CPU counterparts in terms of clock speed and it has less memory. As a result, GPUs can struggle with processing tasks that are not well-structured [1]. Not to forget that the GPU is quite expensive and it consumes a lot of power (it has its own power source). A novel approach to leverage the resource for computation is to use **hybrid** or **heterogeneous** programming. The latter refers to combining multiple hardware architectures in one architecture. Hybrid programming (like OpenMP-MPI programming) uses shared–distributed memory architecture. Heterogeneous programming (sometimes also called hybrid programming) is designed for multi/many-core systems such as CPU-GPU architecture. Examples of hybrid and heterogeneous programming are given below:

- MPI+OpenMP
- MPI+CUDA
- OpenMP+CUDA
- MPI+OpenMP+CUDA
- MPI+OpenCL

4 Presentation of the HPC Cluster Emir

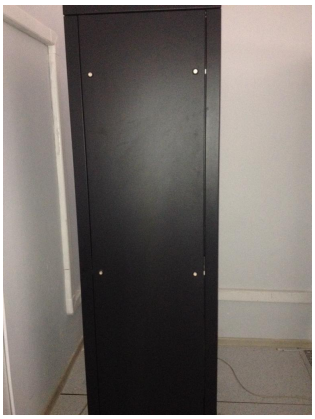
In this guide, we will use the HPC Cluster Emir of the University of Mascara as a showcase. The main component of the Cluster are shown in the following figures.

[1] well-structured program is a program that is arranged in a logical order and is easy to read, debug, and update.

HPC Cluster

Voltage stabilizer

UPS



Cooling system

4.1 HPC Custer Emir Hardware

The HPC Cluster Emir has been installed at the University of Mascara in 2015. It is manufactured by BULL, a leading company in the technology of Supercomputers. The cluster has in total 35 nodes. One node is dedicated to the administration and management of the cluster and it has a capacity of 8x3000GB with two Intel Xeon processors and PYNY nVidia Quadro K5000 Graphic card. One node is dedicated to the storage and it has a capacity of 36000 GB (12x3000GB HDD). One node is dedicated to visualization. It has a storage capacity of 8x3000GB HDD and a graphic card

of type PYNY Nvidia Quadro K5000, 4GB GDDR5, and 1536 CUDA cores (see https://www.gpuzoo.com/GPU-NVIDIA/Quadro_K5000.html for more specifications). The remaining 32 nodes are dedicated to computation.
One each one of the 35 nodes, there are two Intel Xeon E5-2620v2 CPUs (processors). Each CPU has a RAM of 64GB and a frequency of 2.5-3.3GH. So in total, the compute nodes have 640 cores.
The communication between you (the user) and the HPC cluster can be done via Ethernet. The communication between all the nodes of the cluster is done via the Infiniband interconnect with an extremely fast bandwidth (40GB/s). The table below summarizes the resources of the HPC-Emir

Matériels	Configuration
1 Management Node	Bullx R425-E3 : CPU 2 x E5-2670v2 10 cores 2.5 Ghz RAM 64GB@1600MHz - HDD 8x3000GB
1 Storage Node	Bullx R423-E3 : CPU 2 x E5-2620v2 2.1Ghz RAM 64GB@1600MHz - HDD 12 x 3000GB
1 Visualization Node	Bullx R425-E3 : CPU 2 x E5-2670v2 10 cores 2.5 Ghz RAM 64GB@1600MHz - HDD 8x3000GB
32 Compute Nodes	Bullx R425-E3 : CPU 2 x E5-2670v2 10 cores 2.5 Ghz RAM 64GB@1600MHz, 500GB HDD

The host names for the different nodes are:
emir0: The management node.
emir1: The storage (NFS) node.
emir10: The visualization node.
emir[11-42]: Compute nodes.

4.2 HPC Custer Emir Software

The Cluster Emir is a Linux-based HPC cluster running CentOS 7 (**C**ommunity **ent**erprise **O**perating **S**ystem) which is based on Red Hat Enterprise Linux (RHEL) sources. The software installed on the cluster are summarized in the table below.

gcc/g++	C/C++ compilers	Open Source
gfortran	Fortran compiler	Open Source
python3, python	Python compiler	Open Source
GO	Go programming language	Open Source
Singularity	Containers Technology	Open Source
gcj	Java compiler	Open Source
openmpi	MPI compiler	Open Source
openmp	OpenMP compiler	Open Source
Anaconda, Miniconda Pandas, OpenCV NumPy, SciPy Matplotlib, TensorFlow PyTorch, Keras Pillow, Scikit-learn mysqldb, tkinter	Development tools for Machine Learning, Artificial Intelligence, Computer Vision and IoT	Open Source
opencl	OpenCL compiler	Open Source
nvcc	CUDA compiler	Open Source
opengl	OpenGL compiler	Open Source
Octave	Computer Algebra (Similar to Matlab)	Open Source
Quantum Espresso	Ab-initio simulations	Open Source
Wien2k	Ab-initio simulations	License available

Table 1: Software installed on HPC-Cluster Emir.

5 Accessing the HPC Cluster Emir

Access to the Cluster Emir is available via encrypted and secured connection **ssh**. The ssh program allows the user to open a text console session on a remote computer. When a user connects to a computer at University, the user must enter their **username** and the appropriate **password**. This is done from a shell or terminal prompt on a Linux or Mac OS system. On a windows machine, the user can access the cluster using a graphical ssh program under Windows, such as PuTTY (Cygwin or MKS can also be used).
From inside the University, you can access your account as follow:

5.1 On Linux

Connections to the Cluster via ssh can be made from a terminal. Once the terminal window is open, the syntax of the commands is usually the same on all Linux systems (Ubuntu, RedHat, Fedora, CentOS, Debian, SUSE,...). On terminal type [2] (see Fig. (11))

ssh username@172.19.30.13

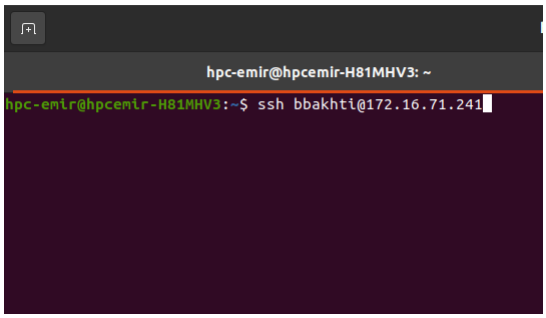


Figure 11: Typing username

then press Return (Enter) on the keyboard. These commands should bring up another line (see Fig. (12)) asking you to type in your password (note that nothing is shown on the screen when you type the password).

[2] Note that the IP address "172.16.71.241" appearing in the figures is the one used previously in the cluster. It is not working any more now. Instead, use the IP address: 172.19.30.13

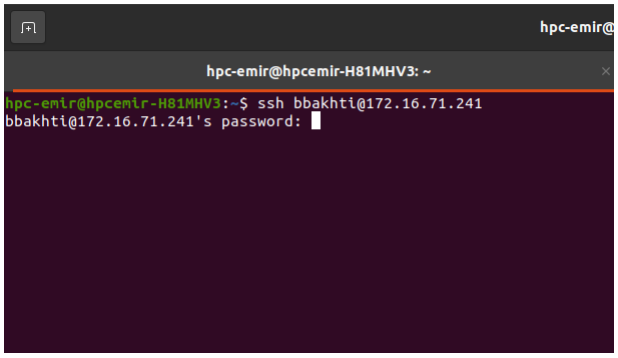


Figure 12: Typing password

Enter your password, and press Return (see Fig. (12)). username must be replaced by your account name. For example, my user name is **bbakhti**, login to the cluster can be done as [3] (see Fig. (11)):

ssh bbakhti@172.19.30.13

You are now logged in on the HPC Cluster (see Fig. (13)), and can use any of the Linux, module, or queue system commands described in this manual.

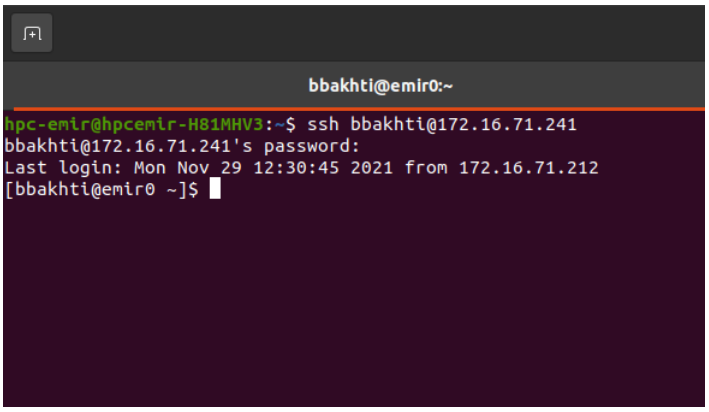


Figure 13: Login

[3] Check footnote in the previous page

5.2 On Windows

Windows does not come with an ssh program, but several free ssh programs are available. The easiest to use, free ssh program is **PuTTY**. Here you need to install putty on your windows computer. Depending on your windows version, you can download Putty from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html (see Fig. (14)).

Figure 14: Download PuTTy.

The installation can be done following the steps below:
1- Double click on the downloaded file.
2- Click **Next** on the welcome screen (see Fig. (15)).

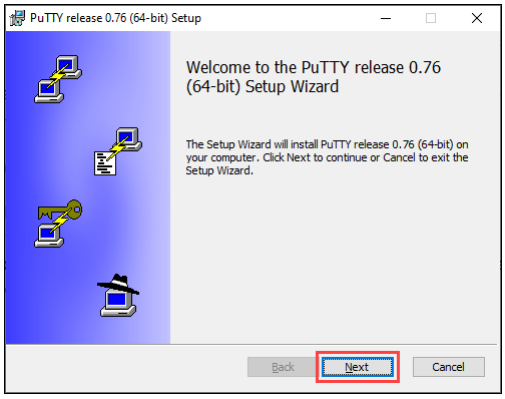


Figure 15: Double click on the downloaded file.

3. Click **Next** if you don't need to modify the installation path (see Fig. (16)). Click **Change**... to specify another path (but this is not necessary).

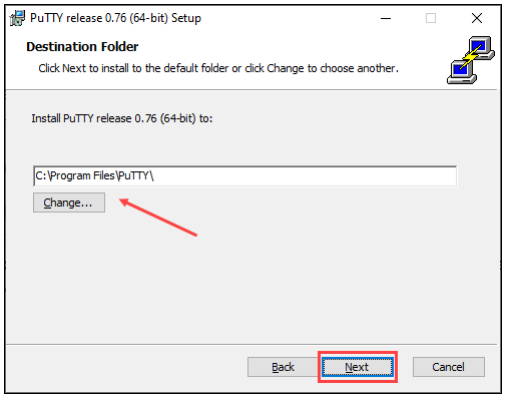


Figure 16: Press Next.

4- Click **Install**(see Fig. (17)).

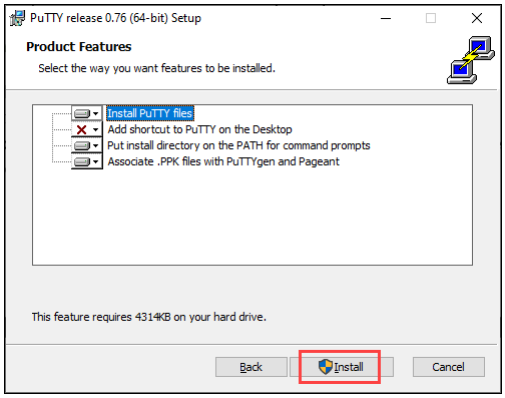


Figure 17: Press Install.

5. Upon completing the installation, the program shows a 'Setup complete' screen. Check/uncheck the View README file option if you want to see the developer's notes. Click **Finish** to exit the installer (see Fig. (18)).

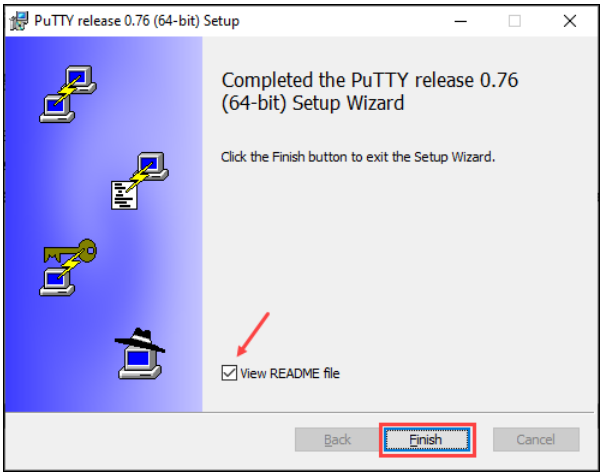


Figure 18: Press Finish.

Putty is installed now on your PC. To connect to the Cluster with PuTTY, first, double click on the PuTTY icon on the Windows desktop or Laptop or search PuTTY in the search bar and press Enter (see Fig. (19)). This will open the PuTTY Configuration window.

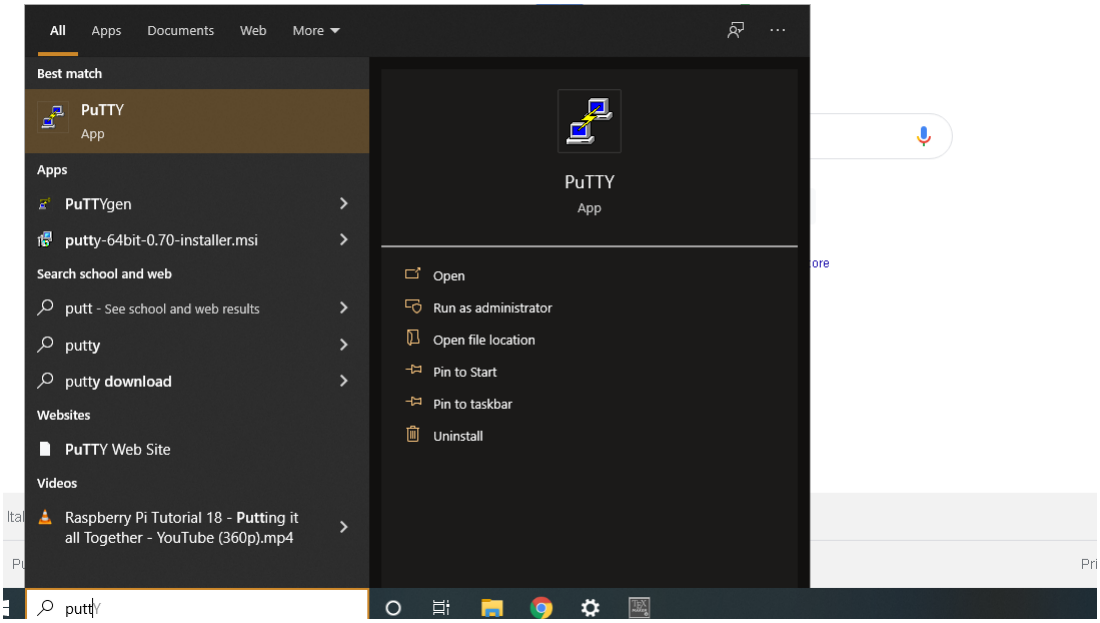


Figure 19: Open PuTTy.

In the PuTTY **configuration window** (see Fig. (20)), choose the **ssh** for the connection type, keep the port 22 and type in the **hostname** or **IP address** of the cluster.

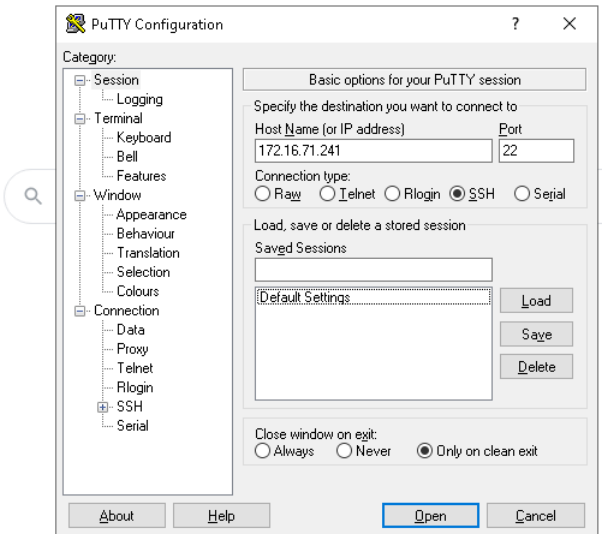


Figure 20: Type IP address.

Windows may pop up a security warning message that requires you to click on "yes", "Allow", or "Run", or "Continue" in order to allow the PuTTY software to run Fig. (21)). Click **yes**.

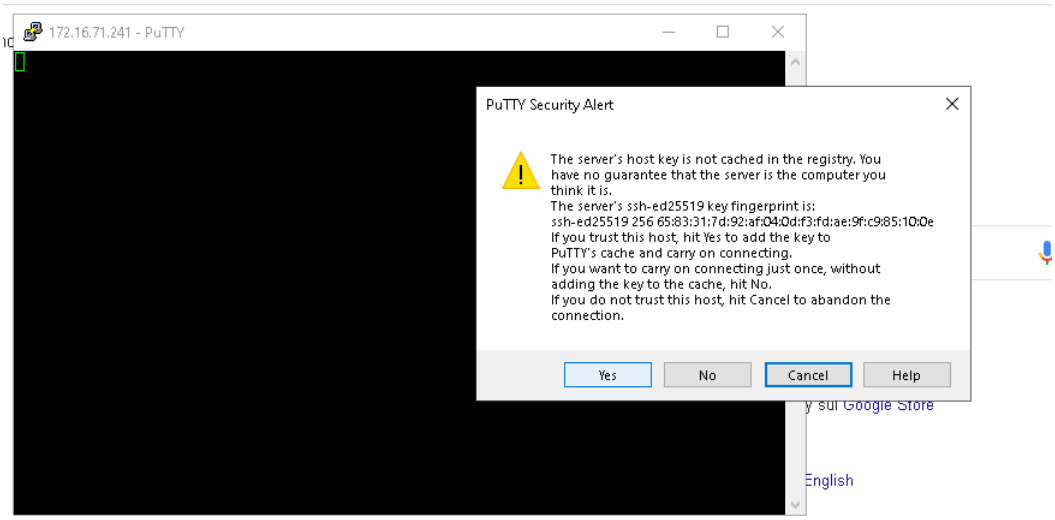


Figure 21: Press Yes.

The PuTTY terminal window will appear with the "**login as:**" prompt Fig. (22)). Enter your **username**, and press Return.

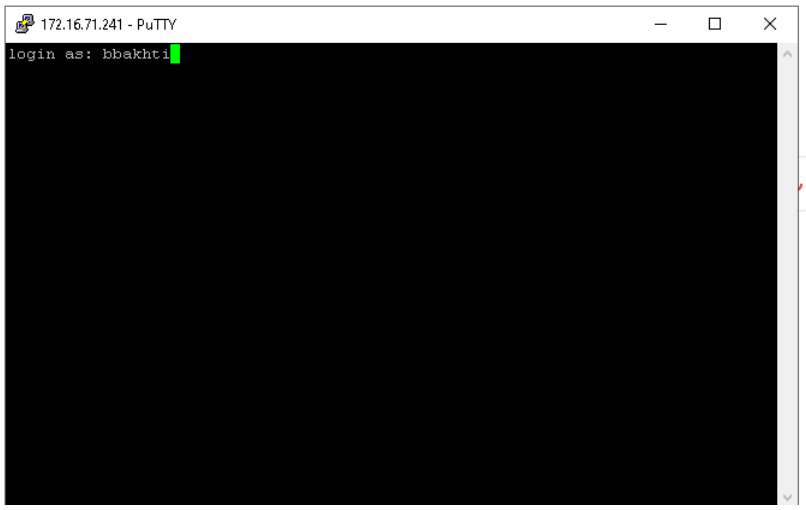


Figure 22: Type username.

Next, the "**Password:**" prompt will appear Fig. (23)). Enter your pass-

word and press Return. Note that nothing is shown on the screen when you type the password,

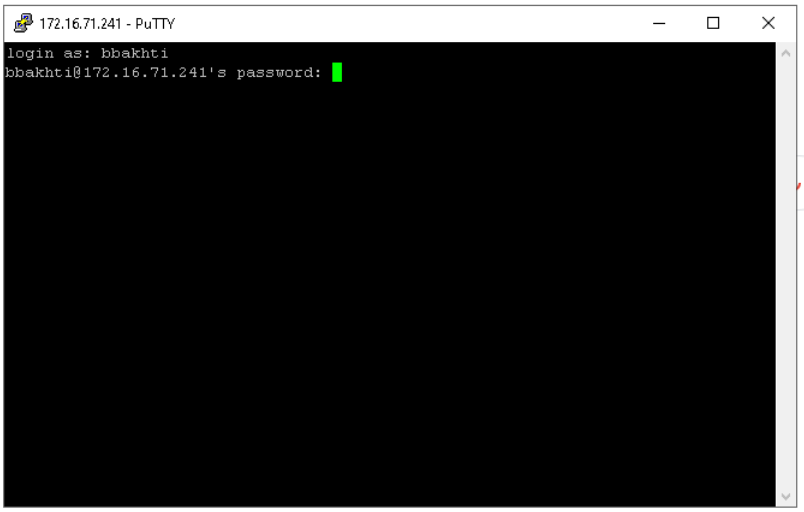


Figure 23: Type password.

Congratulation! you should be in the terminal console of your account Fig. (24)).

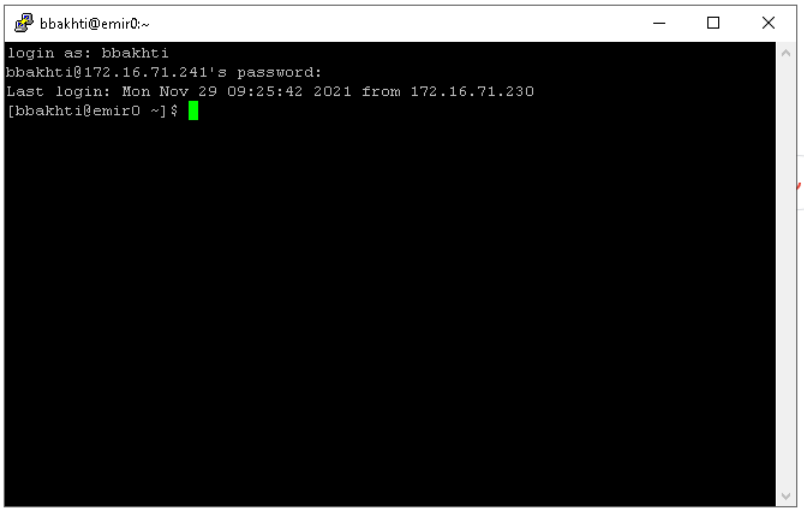


Figure 24: You are login.

If you want to exit your account (**log off** the Cluster), simply type **exit** and press Return.
For an advanced user, a much better and recommended way to ssh to the HPC Cluster from a Windows PC/Desktop is to use **MobaXterm**. the installation and use of mobaXterm are described in the appendix.

5.3 Transferring files between PC and Cluster

Files can be transferred between your PC on the Cluster using the **scp** command. To transfer the file to the cluster, open the terminal prompt on Linux or PuTTy on Windows. Then you should navigate to the path where your file is located. Suppose your file is a python file named $fileExample.py$ (but it can be a file with any extension such as c, cpp, doc, pdf,...). To transfer the $fileExample.py$ to a folder named $myFolder$ in your cluster home directory, use the following command

scp fileExample.py / username@172.16.71.241:./myFolder

Then press Return. The file should be transferred. An example is shown in Fig. (25)

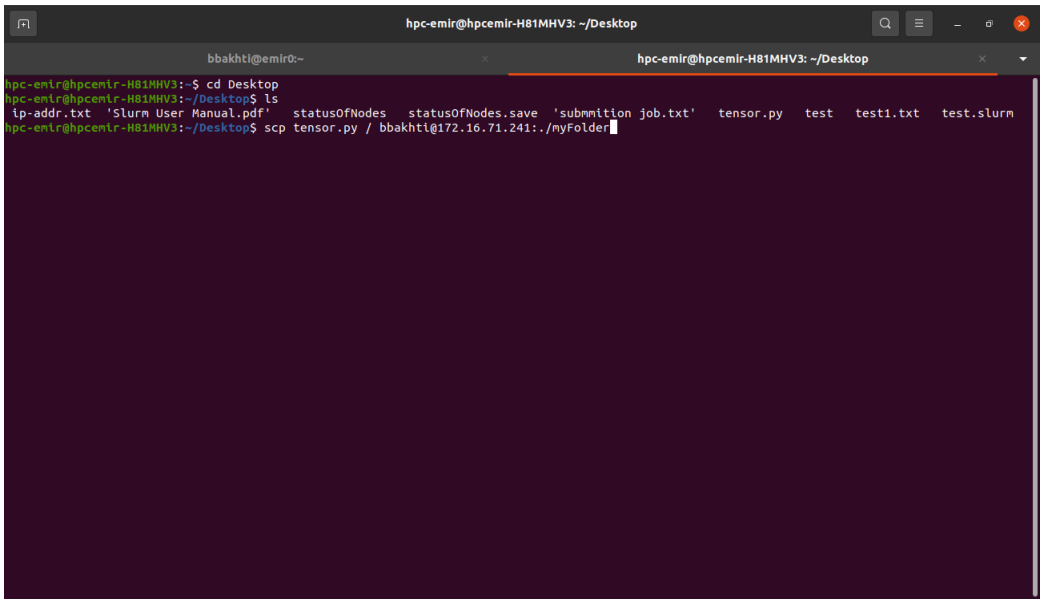


Figure 25: Copy file from PC to Cluster.

In this example; the file **tensor.py** will be transferred from the Desktop to myFolder in the cluster home directory.
To move a folder named $folderExample$ to the $myFolder$ in the cluster use.

scp -r folderExample / username@172.16.71.241:./myFolder

and press Return.
Transfer of files (for example output file) from the Cluster to your computer can be done as follow. Open a terminal prompt on your computer. Navigate to the place where you want to move the file to. Suppose the file you want to move is named $output.txt$ and it exists in the $myFolder$ folder in your home directory in the cluster. use the following command to move the file to your PC

scp usename@172.16.71.241:./myFolder/output.txt .

and press Return. Note the space between $output.txt$ and the dot is mandatory. An example is shown in Fig. (26)

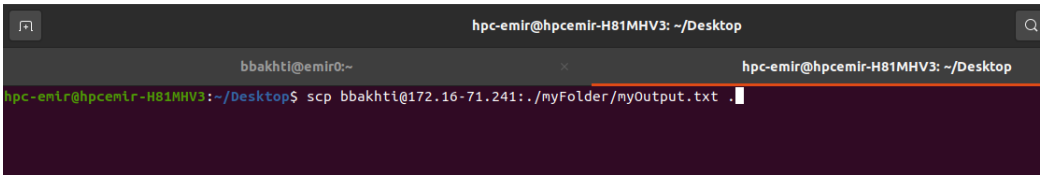


Figure 26: Copy file from Cluster to PC.

In this example; the file **myOutput.txt** will be transferred from the myFolder folder in the cluster home directory to the Desktop of my PC:

5.4 Submitting Job to the Cluster

Executing a job on the HPC Cluster Emir will be done using the following steps:

- Determine the resources necessary for the specific job (number of nodes, number of CPUs, required time, GPU, amount of memory,...).
- Create a Batch job script.

- Submit the job to the scheduler.
- Check the job status.
- Retrieve job information and output.

All these steps will be performed using a specific software called **SLURM**. Details on these operations will be given below and for more information on Slurm, please check https://schedmd.com/ or https://hpc.llnl.gov/training/tutorials#trainingmaterials.

5.4.1 What is Slurm

Slurm (**S**imple **L**inux **U**tility for **R**esource **M**anagement) is an open source, fault-tolerant, and highly scalable cluster management software. It is the most commonly used job scheduler for Linux-based HPC clusters and is the software we use to run all jobs on the HPC Cluster Emir. Slurm performs several important functions: it allows the user to allocate time and resources on the compute node(s) to perform a job. It also allows the user to start and monitor the status of the job as it runs on the worker nodes. Finally, it queues and balances job submissions for all users on the cluster.

5.4.2 Determine resources for job

Choosing appropriate resources for your jobs is essential to ensuring your jobs will be scheduled as quickly as possible while wasting as few resources as possible. There are no recipes to follow when choosing the cluster resources for your job. But generally, you will learn this with experience working with the cluster.
The key resources you need to optimize for are:

- Time allocation
- Nodes and Cores allocation
- Memory-allocation

Time requirements are highly dependent on how many CPU cores your job is using - using more cores may significantly decrease the amount of time the job spends running.
In order to determine your CPU requirements, you should investigate if your

program/job supports parallel execution. If the program only supports serial processing, then you can only use 1 CPU. If your job/program supports multiple cores, you need to allocate multiple CPUs. At the moment, the Maximum number of cores that you can allocate is 30 cores. For a GPU job, you can only use the visualization node of the HPC-Emir Cluster.
For the memory allocation, submit your job **sbatch** by specifying very generous memory and time requirements to ensure that it runs to completion and also using the $--$**mail**$-$**user=** and $--$**mail**$-$**type=ALL** parameters to receive an email-report. The mail message will list the maximum memory usage (maxvmem/MaxVMSize) as well as the wallclock time used by the job.
Remember, the larger your request, the longer it will take for the resources to become available and the time taken to queue is highly dependent on other cluster jobs.

5.4.3 Create a Batch script for the job

I present here some slurm batch scripts samples that can be used as a template for building your own Slurm submission scripts for use on HPC-Emir. Please make sure you understand each #SBATCH directive before using the script to submit your jobs. First, you need to create an empty batch script (with **.sh** or **.slurm** extension). On both Linux (terminal) or Windows (PuTTy console) you can use either the command **nano** or the command **vi** as follow (see Fig .(27) for PuTTy on Windows):

nano batch_example.sh

vi batch_example.sh

Both commands bring a a similar empty file.

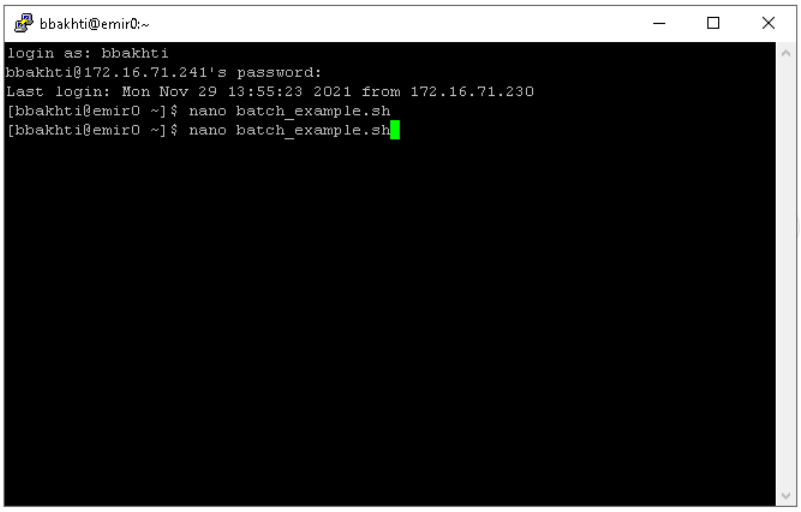


Figure 27: Create batch file.

After pressing enter you should get an empty file like the one displayed in Fig .(28) (for Linux terminal, but you get exactly the same thing for PuTTy console)

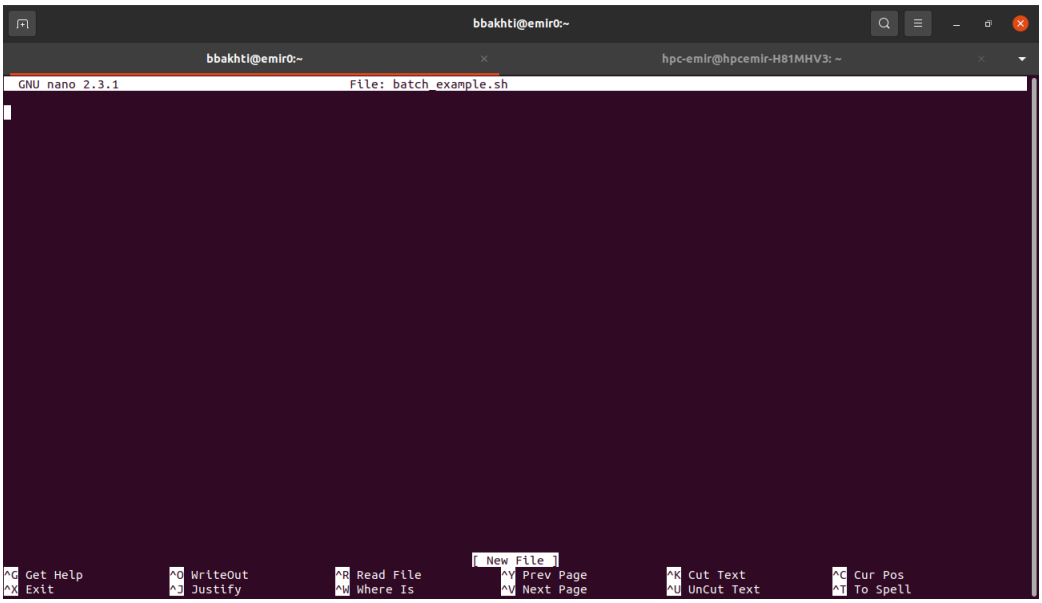


Figure 28: Empty batch file (on Linux).

Copy the following batch file into the empty terminal screen (see Fig .(29) for both Windows putty and Linux terminal). If you have used **vi** editor to create the batch file, then you need to press the letter "**i**" to insert or modify text.

```
#!/bin/bash
# set name of the job
#SBATCH -job-name=mpi_program
# The "all" partition is the default partition
#SBATCH --partition=all
# set the number of nodes (2 in the present case)
#SBATCH --nodes=2
# number of processes per node (processors cores)
#SBATCH --ntasks-per-node=20
# Job memory request
#SBATCH --mem=1gb
# set max wallclock time (D-hh:mm:ss)
#SBATCH --time=0-00:40:00
# load the required modules
module load openmpi
# Define the mail address for notifications
#SBATCH --mail-type=end
# When to send mail notifications Options: BEGIN,END,FAIL,ALL
#SBATCH -mail-user=email@univ-mascara.dz
# Redirect standard error
#SBATCH --error=mpi_program.err
# Redirect standard output
#SBATCH --output=mpi_program.out

# And finally run the job
srun ./mpi_program
```

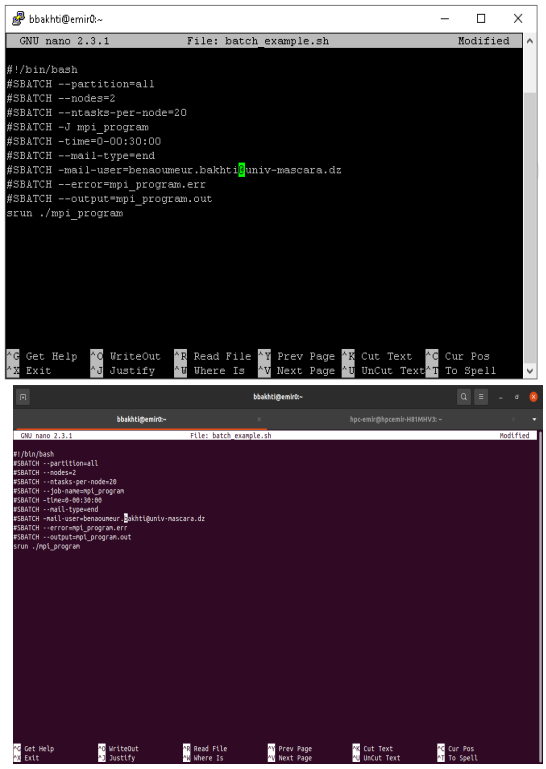


Figure 29: Batch file on Windows (up) and Linux (bottom).

To exit and save the batch file in the **nano** editor, press **Ctlr+X** then **y** then **Return**. To exit and save the batch file in the **vi** editor, press first **Esc** in the keyboard, then type **:wq**, then press **Return** and you are done.

The batch file has many options. You can check all options by running the command **man batch** in the cluster terminal. The most used commands of the slurm batch file are:
−−**job−name=** or −**J** gives a name for the job.
−−**partition** or simply −**p** is used to specify the partition to which you want to submit your job. For the Cluster Emir, we have only one partition "**all**".
−−**ncpus-per-task=** or −**c** is used to fix the required number of CPU.
−−**nodes** or simply −**N** is used to fix the required number of nodes.
−−**ntasks−per−node=** specify how many tasks will run on each allocated node.

−−**mem-per−cpu=** fixes the minimum memory required per allocated CPU in megabytes.
−−**mem=** used to fix the required memory per node.
−−**gres=gpu:** used to fix the number fo GPU.
−−**ntasks** or in short −**n** is used to fix the number of cores.
−−**time=** or −**t** is used to fix the time (in the format D−h:m:s).
−−**output=** or −**o** names the file in which to store the job output.
−−**error=** or −**e** specifies the file name in which to store job error messages.
−−**mail−user=** this is the mail address to contact the job owner (send notifications).
−−**mail−type=** specifies when to notify a job owner: none, all, begin, end, fail, requeue, array_tasks.
−−**input=** specifies the file name from which the job read input data.
−−**accounts=** displays the job with specific accounts.

Here are some examples of SLURM batch scripts for running a job on the HPC cluster Emir with MPI−C/C++, MPI−Fortran, MPI−Python, OpenC, and CUDA:

MPI−C++ Slurm batch file

```
#!/bin/bash

#SBATCH -p all
#SBATCH -J MPIProgram
#SBATCH -o MPIProgram-%j.out
#SBATCH -e MPIProgram-%j.err
# Run for a maximum time of 0 days, 12 hours, 00 mins, 00 secs
#SBATCH -t 0-12:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --mem-per-cpu=4000
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your email>

# list modules loaded by default.
module list
load module gcc openmpi
```

```
# prints current working directory
pwd
# prints the date and time
date

# run the MPI job
mpirun my_program.cpp
```

MPI–Fortran Slurm batch file

```
#!/bin/bash

#SBATCH -p all
#SBATCH -J MPIProgram
#SBATCH -o MPIProgram-%j.out
#SBATCH -e MPIProgram-%j.err
#SBATCH -t 0-12:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --mem-per-cpu=4000
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your_email>
# list modules loaded by default(GNU8 compilers and OpenMPI3 MPI libraries)
module list
# swap the MPI library from the default openmpi3 to mpich.
module swap openmpi3 mpich
module list
pwd
date
mpirun my_program.f
```

MPI–*P*ython Slurm batch file

```
#!/bin/bash
#SBATCH -p all
#SBATCH -J MPIProgram
#SBATCH -o MPIProgram-%j.out
#SBATCH -e MPIProgram-%j.err
#SBATCH -t 0-12:00:00
```

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --mem-per-cpu=4000
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your_email>
module list
pwd
date
mpirun python my_program.py
```

CUDA Slurm batch file
A SLURM job script to submit a CUDA application on a host with a Nvidia GPU could be done as follow:

```
#!/bin/bash

#SBATCH --job-name=gpuProgram
#SBATCH --account=hpc_emir_account
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=10     # CPU cores/threads
#SBATCH --time=12:00:00       # format  HH:MM:SS
#SBATCH --mem-per-cpu=4000
#SBATCH --chdir=/work/<hpcDirectory>/     # work directory
#SBATCH --partition=all
#SBATCH --gres=gpu:1              # Number of GPUs (per node)
#SBATCH --mail-user=username@univ-mascara.dz
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output=gpuProgram_%j.out
#SBATCH --error=gpuProgram_%j.err

#load the CUDA module
module load cuda-10-0/10.0

echo "== Start memory test ============"
srun ./memtestG80

# run the job
srun ./my_cuda_program
```

The option "#SBATCH −−gres=gpu:1" lets SLURM allocate one GPU for the job.

OpenCL Slurm batch file

Because there is an Nvidia GPU installed on the Cluster, you can use the Nvidia OpenCL implementation that supports only Nvidia GPUs. It can be used as follows:

```
#!/bin/sh

#SBATCH -J gpuProgram
#SBATCH -p all
#SBATCH --account=hpc_emir_account
#SBATCH --t=00:15:00
#SBATCH -N 1
#SBATCH --gres=gpu:1
#SBATCH --mail-user=username@univ-mascara.dz
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH -o gpuProgram_%j.out
#SBATCH -e gpuProgram_%j.err

module load opencl-nvidia/11.2

# run the job
./opencl-program-nvidia
```

Note that if needed, you can specify the node with the installed GPU as:

```
#SBATCH -C GK104
```

where GK104 is the GPU installed on the Nvidia Quadro K5000. Before showing how to submit a job, let talk about the concept of Modules.

5.4.4 Modules: Setting up your environment

The module is a concept used in HPC clusters to specify the different versions of software and compilers. This concept is extremely important because on an HPC cluster a large number of software packages are installed and often there will be several versions provided for a package where it will be necessary for a user to choose between them. For example in the case of python one can load the version python2.7 or the version python3. Equally, two different

packages may clash with each other: for example, the commands for Intel MPI and Open MPI would overlap if simultaneously installed. One always has to choose the compiler that could improve the performance of his job or the one that can make things easier or run faster. So in the batch file, we need to set up the environment by loading all the required modules. When the package is no longer needed, users can unload the module from their environment. To get a list of available modules, use one of the following (in the cluster terminal):

module avail

To load a module into your environment to start using an application, use one of the following,

module load module_name/version

For example to load Matlab (it is not available at the moment in the Cluster Emir) use,

module load matlab

In most cases, you need to specify the version of the software, for example,

module load matlab/R2016a

To get a list of currently loaded modules, use:

module list

To unload a specific module, use:

module unload module_name/version

For example, to unload Matlab use:

module unload matlab

Modules can have multiple versions of the software, and you can see the versions either with module avail or by doing (example for CUDA):

module apropos cuda

If you want to unload the old module and load the new one, use:

module switch old_module_name new_module_name

To get information about the modules, use:

module whatis module_name

and to get a short description of the module, use:

module show/display module_name

An important point to note here is that some software packages depend on other software. In this case, loading module for the package may load modules for the other software as well. This process is called dependency. Examples of modules are given in the following table

Software	Modules
Intel compilers	intel
GNU compilers	gcc
PGI compilers	pgi64
Python compiler	python/2.7.12, python/3.6.0
MPI compiler	mpicc, mpic++, openmpi/2.1.2, openmpi
Matlab	matlab, matlab/R2016a
Mathematica	mathematica/11.3.0
Gnuplot	gnuplot/5.0.6
Nvidia CUDA Comiler	CUDA/10.0.130, CUDA/10.2.89-GCC-8.3.0
Wien2K compiler	wien2k
Octave	octave/3.8.1

To load compilers (for example Intel compilers), use:

module load intel

this will load intel compilers such as icc (c compiler), icpc (c++ compiler) and ifort (fortran compiler).
To load the GNU compilers such as gcc (c compiler), g++ (c++ compiler) and gfortran (Fortran compiler), use:

module load gcc

To load CUDA 11.1 (for GPU programming) use:

module load CUDA/11.1

You can load CUDA combined with other compilers. For example, to load CUDA 10.2 plus the GCC 8.3 compiler, use:

module load CUDA/10.2.89-GCC-8.3.0

To load CUDA 10.1 plus the GCC 8.x compiler, OpenMPI, OpenBLAS, SCALAPACK and FFTW use:

module load fosscuda/2019b # includes GCC 8.3
module load fosscuda/2019a # includes GCC 8.2

A list of different compilers is given in the following table:

Compiler	C++	C	Fortran 77	Fortran 90	MPI
GNU	g++	gcc	gfortran	gfortran	none
GNU	mpic++	mpicc	mpifort, mpif77	mpifort, mpif90	OpenMPI
GNU					Intel MPI
Intel	icc	icc	ifort	ifort	none
Intel	mpiicc	mpiicc	mpiifort	mpiifort	Intel MPI
Intel	mpic++	mpicc	mpifort, mpif77	mpifort, mpif90	OpenMPI
PGA	pgc++	pgcc	pgf77	pgf90	none
PGA	mpic++	mpicc	mpifort	mpifort	OpenMPI

5.4.5 Submit and check the status of a job

There are 2 commands for job allocation: **sbatch** is used for batch jobs and **salloc** is used to allocate resources for interactive jobs. The format of these commands:

- sbatch [options] jobscript [args...]
- salloc [options] [command [command args]]

If the batch file is named "*my_batch.sh*" or "*my_batch.slurm*", then you submit your job using

sbatch my_batch.sh
or
sbatch my_batch.slurm

This command will automatically queue your job using SLURM and produce a job ID number (shown below). You can check the status of your job at any time with the ***squeue -j <JOB_ID>*** command.

squeue -j job_ID

You can also stop your job at any time with the **scancel** command.

scancel job_ID

Other useful commands to check the job status are summarized in the following table

List all current jobs for a user	squeue −u username
List all running jobs for a user	squeue −u username −t RUNNING
List all pending jobs for a user	squeue −u username −t PENDING
List all current jobs in the general partition for a user	squeue −u username −p general
List detailed information for a job (useful for troubleshooting)	scontrol show jobid −dd jobid
To cancel one job	scancel jobid
To cancel all the jobs for a user	scancel −u username
To cancel all the pending jobs for a user	scancel −t PENDING −u username
To cancel one or more jobs by name	scancel –name JobName
To pause a particular job	scontrol hold jobid
To resume a particular job	scontrol resume jobid
To requeue (cancel and rerun) a particular job	scontrol requeue jobid

When running the **squeue** command, you get information (in columns) displaying the Job_IDs, names of the job, STs, the users, and the nodes. The ST column gives the state of the job, with the following codes:

- R for Running
- PD for PenDing
- TO for TimedOut
- PR for PReempted
- S for Suspended
- CD for CompleteD
- CA for CAncelled
- F for FAILED
- NF for jobs terminated due to Node Failure

5.4.6 Retrieve output

As stated before, different commands can be used to retrieve the results and output of the programs from the HPC cluster. The most used one is the **scp** command. The latter can be used as follow

scp usename@172.16.71.241:./myFolder/output.txt .

and transfer a directory or a folder use

scp -r usename@172.16.71.241:./myFolder/output.txt .

I recall that the space between *output.txt* and the dot is mandatory. Another command that can be used is the **sftp**.

5.4.7 Slurm most used Commands

- **salloc** to request interactive jobs/allocations
- **sattach** to attach standard input, output, and error plus signal capabilities to a currently running job or job step
- **sbatch** to submit a batch script (which can be a bash, Perl, or Python script)
- **scancel** to cancel a pending or running job or job step
- **sbcast** to transfer a file to all nodes allocated for a job
- **sgather** to transfer a file from all allocated nodes to the currently active job. This command can be used only inside a job script
- **scontrol** provides also some functionality for the users to manage jobs or queries and get some information about the system configuration
- **sinfo** to retrieve information about the partitions, reservations, and node states
- **smap** graphically shows the state of the partitions and nodes using a curses interface.
- **sprio** can be used to query job priorities

- **squeue** to query the list of pending and running jobs
- **srun** to initiate job steps mainly within a job or start interactive jobs. A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job's node allocation
- **sshare** to retrieve fair-share information for each user
- **sstat** to query status information about a running job
- **sview** is a graphical user interface to get state information for jobs, partitions, and nodes
- **sacct** to retrieve accounting information about jobs and job steps in Slurm's database
- **sacctmgr** allows also the users to query some information about their accounts and other accounting information in Slurm's database.

6 A short introduction to Linux command

In this section, we present the list of the most used basic Linux commands:

sudo: Short for (short for SuperUser Do), the sudo command is used when you want to perform tasks that require administrative or root permissions.
pwd: the pwd (short for present working directory) command is used to find out the path of the current working directory (folder) you're in.
cd: the cd (short for change directory) command is used to navigate through the Linux files and directories. There are some shortcuts to help you navigate quickly:

- **cd ..** (with two dots) to move one directory up
- **cd** to go straight to the home folder
- **cd-** (with a hyphen) to move to your previous directory
- **cd** or **cd** $\sim$ to navigate to your home directory
- **cd /** to navigate into the root directory

ls: the ls command is used to view the contents of a directory

- **ls -R** will list all the files in the sub-directories as well
- **ls -a** will show the hidden files
- **ls -al** will list the files and directories with detailed information like permissions, size, owner, etc.

cp: the cp command is used to copy files from the current directory to a different directory.
mv: the mv command is used to move files, although it can also be used to rename files.
mkdir: the mkdir command is used to make (create) a new directory.
rmdir: the command rmdir is used to delete a directory.
rm: the rm command is used to delete files. If you want to delete the directory (as an alternative to rmdir) use rm -r.
cat: cat (short for concatenate) is one of the most frequently used commands in Linux. It is used to list the contents of a file on the standard output (sdout). As an example:
cat file.txt
will display the content of the text file file.txt in the terminal. You can use cat also as:

- **cat > filename**: creates a new file
- **cat filename1 filename2 > filename3**: joins two files (1 and 2) and stores the output of them in a new file (3)

touch: the touch command is used to create a blank new file through the Linux command line.
clear: the clear command is used to clear the Linux window terminal.
grep: the grep command is used to search through all the text in a given file.
kill: used to terminate manually a process.
ping is used to check your connectivity status to a server.
top: the top command will display a list of running processes and how much CPU each process uses.
zip, **unzip**: these commands are used to compress or decompress a file.

hostname: used to display the hostname of your host/network. Adding a $-i$ to the end will display the IP address of your network.
passwd: this command is used to change password.

7 References

Here is a list of some books to learn parallel programming. Please, check also youtube for some nice lectures on all the programming languages.
[1] − D. B. Kirk, W-m. W. Hwu, *Programming Massively Parallel Processors*, Morgan Kaufmann; 3rd ed. (2016).
[2] −J. Sanders, E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional; 1st ed (2010); 2nd ed. (2017).
[3] −G. Ruetsch, M. Fatica, *CUDA Fortran for Scientists and Engineers*, Morgan Kaufmann; 1st ed. (2013).
[4] −D. R. Kaeli, P. Mistry, D. Schaa, D. P. Zhang, *Heterogeneous Computing with OpenCL 2.0*, Morgan Kaufmann; 3rd ed. (2015).
[5] −M. Scarpino, *OpenCL in Action: How to accelerate graphics and computation*, Manning; 1st ed. (2011).
[6] −A. Munshi, *OpenCL Programming Guide*, Addison-Wesley Professional; 1st ed. (2011).
[7] − *Parallel Programming with MPI*, Morgan Kaufmann (1996).
[8] −G. Barlas, *Multicore and GPU Programming: An Integrated Approach*, Morgan Kaufmann Publishers; 2nd ed. (2022).
[9] −M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*; McGraw-Hill Education (2008).

Appendices

Appendix A MobaXterm

Putty is easy and works fine but for an advanced user, a much better and recommended way to ssh to the HPC Cluster from a Windows machine is to

use **mobaXterm**. MobaXterm offers support for a lot of protocols, including SSH, VNC, SFTP, FTP, and it has a tabbed interface for easy access to all of the sessions. It is incredibly easy to use and can perform more advanced operations than PuTTY. It also integrates a full set of Unix-based commands. First, you need to install MobaXterm. The latter can be downloaded from https://mobaxterm.mobatek.net/download-home-edition.html. Make sure to select the Installer edition rather than the Portable edition as shown in Fig. (30). You can put the downloaded file anywhere in your computer.

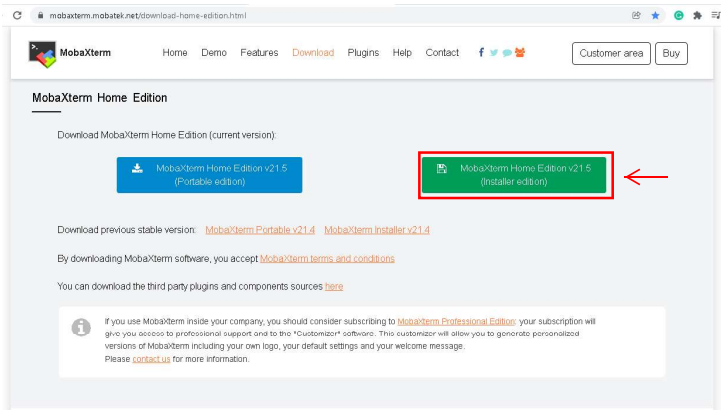


Figure 30: Click on MobaXterm Home Edition

Unzip the downloaded zip file, and then double-click on the MobaXterm installer msi file to begin the installation Fig. (31).

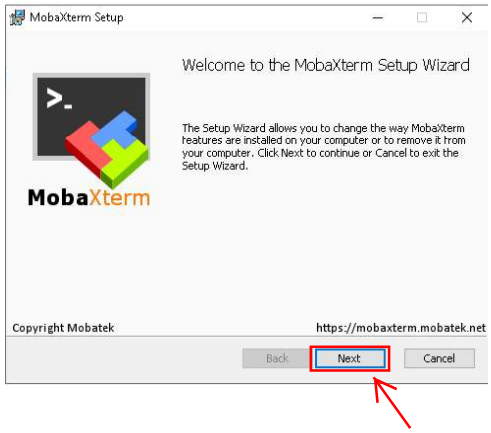


Figure 31: Double click on the msi file

Accept the terms in the licence agreement, then click **next** as in Fig. (32)

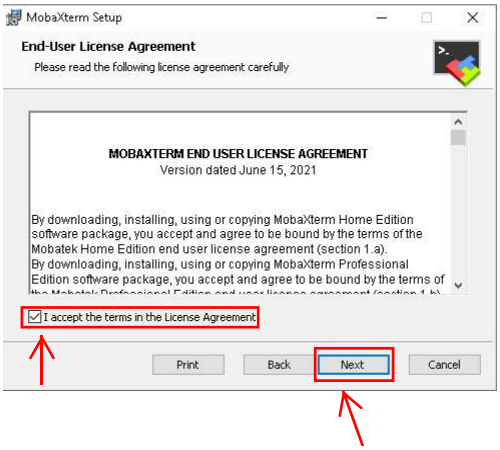


Figure 32: Double click on the msi file

Click **next** to confirm the installation folder. By default it is **C:\program files x86\Mobatex\MobaXterm** (see Fig. (33))

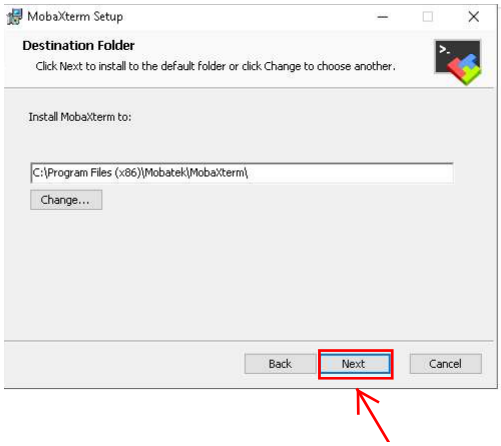


Figure 33: Press next

Sometimes, a window is popping up asking for permission, click **yes**, then click **Install**, Fig. (34)

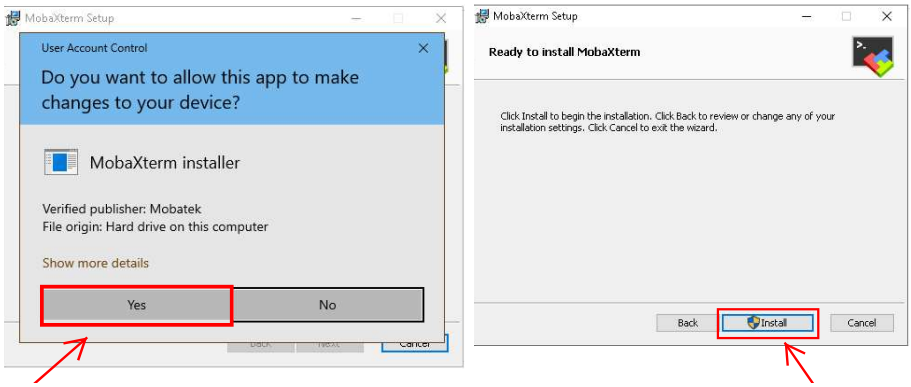


Figure 34: Press Yes then Install

and finally, click **Finish** to finish the installation, Fig. (35))

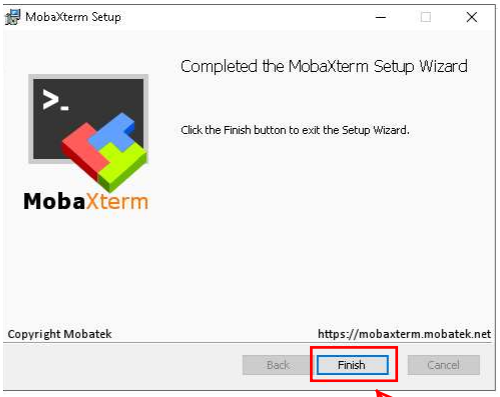


Figure 35: Press Finish

MobaXterm is installed now on your machine. To connect to the Cluster with MobaXterm, Search Mobaxterm in the search bar, and press Enter (see Fig. (37))

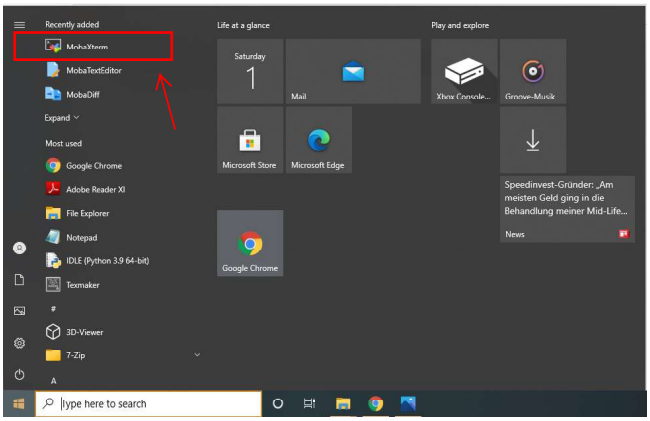


Figure 36: Open MobaXterm from Windows start menu

and then press **Start local terminal**, Fig. (36))

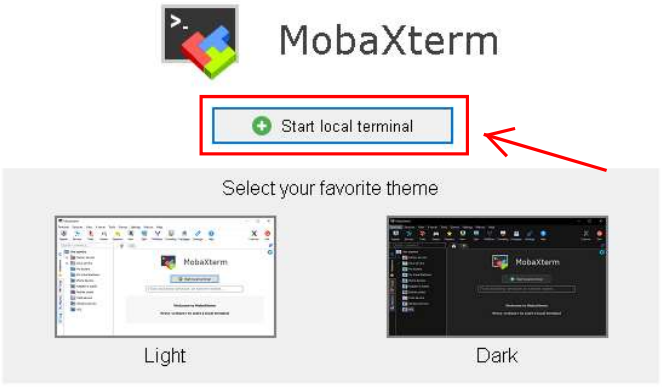


Figure 37: Open MobaXterm from Windows start menu

This will open the MobaXterm Configuration console Fig. (37)

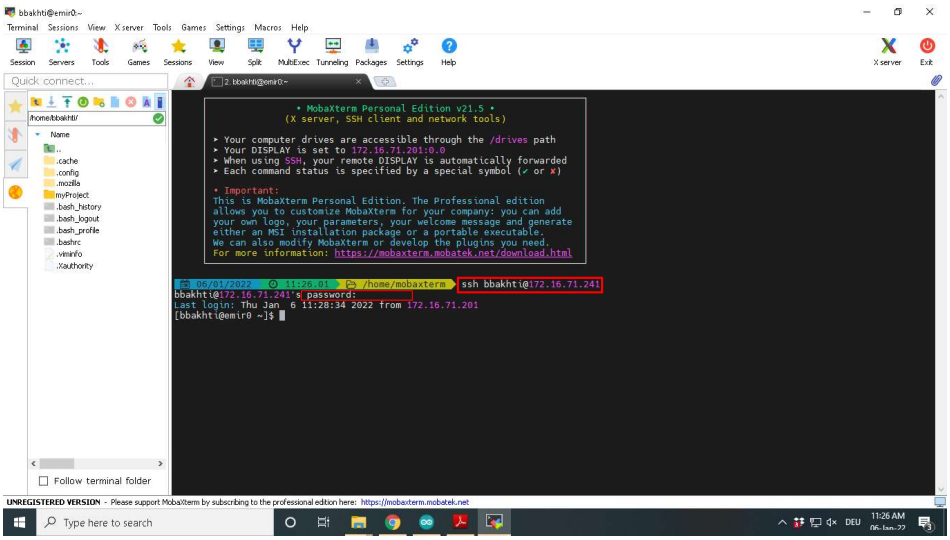


Figure 38: Enter username and password

Now, login to the cluster using the same ssh command, Fig. (38).

Appendix B Singularity Containers

To run in an excellent way the applications development, technology today massively uses virtual machines (VM). These machines run applications within a guest operating system, which uses virtual hardware emulated by the host operating system of the real machine. This technique of isolation between guest and host is excellent but comes at a high price by emulating virtual hardware and running a full guest operating system. **Containers** can be seen as a lighter variant of VM. By exploiting more directly the lower layers of the host (kernel) system, containers provide almost as strong isolation as virtual machines, but with much higher performance. Singularity and similarly Docker are containers technologies that allow users to create and fully control their environment. However, docker is not suitable for HPC applications due to security issues. Unlike the VM which requires a complete copy of the operating system (OS), singularity and docker containers only require the user space of the OS and the kernel space of the OS will be shared with the host OS. With singularity, you can create a container (one or many) and install all workflows, software, and libraries you need into that container. In addition, you can upload the container from the singularity and docker hub and access it from everywhere. Singularity is available on the compute nodes on the HPC-Cluster Emir and users can use SBATCH or an interactive session in SLURM to access it. To use singularity, you need to install it on your computer. There are multiple ways to install singularity. A detailed description of the installation procedure can be found on the singularity main documentation https://sylabs.io/guides/3.0/user-guide/installation.html. Before you can use the singularity command on the system, you may need to load the singularity module using the command:

module load singularity

To get help with the singularity commands use

singularity −−help

To use any software (ubuntu, tesnorflow, mpi, opencv, ..), you need first to pull an image of the software. For example, to pull tensorflow from docker Hub use

singularity pull tensorflow.sif docker://tensorflow/tensorflow:latest

To pull the Ubuntu:18.04 container from Docker Hub use

singularity pull docker://ubuntu:18.04

If the version of the software is not specified in the command, then singularity will pull the latest version.
The next step is to push the image using the **push** command. For example for ubuntu use

singularity push docker://ubuntu:18.04

you are able now to use the Ubuntu container. To use octave:5.2.0 for example, you need first to pull the image using

singularity pull library://siko1056/default/gnu_octave:5.2.0

Then run the image using

singularity run gnu_octave_5.2.0.sif

To run the Octave in the graphical user interface (GUI) mode, use

singularity run gnu_octave_5.2.0.sif −−gui

To use the command line interface (CLI), run:

singularity exec gnu_octave_5.2.0.sif octave-cli

Here are some basic commands of singularity and more detail on how to use these commands for specific applications can be found easily on the web.
singularity run: run the singularity image or the pre-defined script inside the container.
singularity exec: execute a command inside the container.
singularity shell: provide you with a shell within the container.
exit: used to exit the shell command line.
singularity create: used to create your own image.

sudo singularity build: used to build a container.
singularity import: import images from docker.
It is also possible to use Singularity images within a non-interactive batch script. All previous commands can be written within the batch script and then run the latter using the sbatch command. Below is an example of batch-job submission script using the hello-world.simg to print out information about the native OS of the image. You can use the same command to submit this job

sbatch singularity.sbatch