Introduction to HPC and Cyclone

Introduction to High Performance Computing, High Performance Data Analytics and Large-Scale Machine Learning - 9th Nov 2023

Pantelis Georgiades

HELLO!

I am Pantelis Georgiades

Associate Research Scientist at CaSToRC

You can find me at p.georgiades@cyi.ac.cy

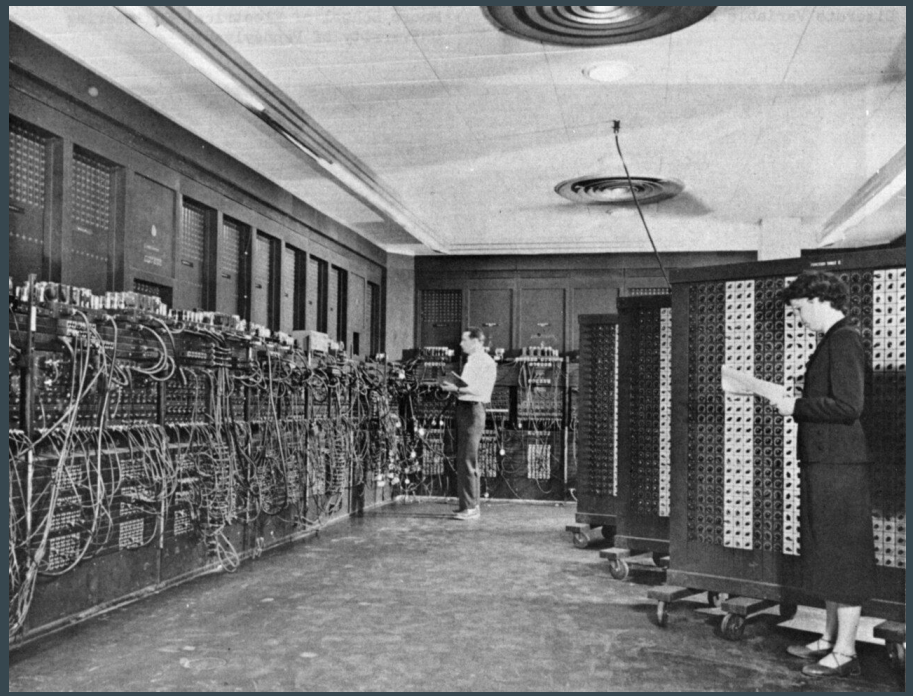
1

High Performance Computing

“The use of supercomputers to solve complex computational tasks”

The evolution of high performance computing

- ENIAC (1940s)
  - One of the first computers
  - Effectively a supercomputer for it's time
  - Large footprint
  - High acquisition and operation cost



The evolution of high performance computing

- ENIAC (1940s)
  - One of the first computers
  - Effectively a supercomputer for it's time
  - Large footprint
  - High acquisition and operation cost
- Cray-1 (1970s)
  - The dawn of personal computing, but supercomputers were needed for specialised tasks
  - Parallelism emerges
  - First to implement vector processing

The evolution of high performance computing

- 1990s-2000s
  - Supercomputers are being designed using commodity hardware
  - Parallelism as we understand it today: distributed and shared memory, message passing etc.

Earth Simulator 1 supercomputer (2002-2009)

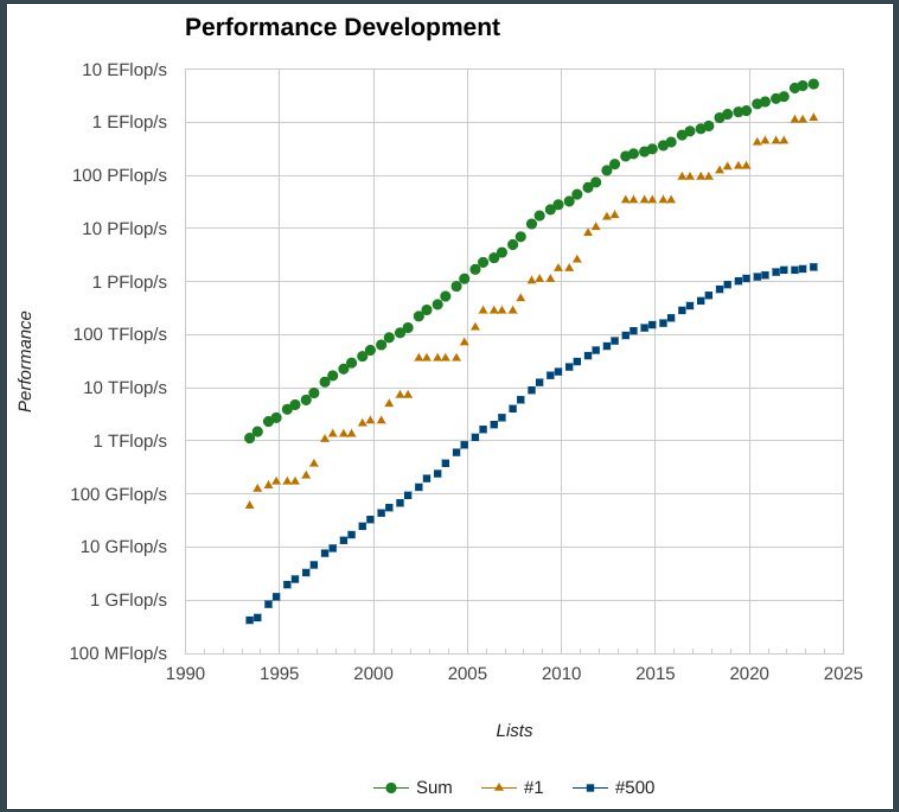
The evolution of high performance computing

- 1990s-2000s
  - Supercomputers are being designed using commodity hardware
  - Parallelism as we understand it today: distributed and shared memory, message passing etc.
- 2010s onwards
  - Dawn of heterogeneous supercomputers
  - Many sockets per node; co-processors, e.g. GPUs, potentially multiple architectures within same system

LUMI supercomputer

362,496 cores, AMD EPYC CPUs, 10,240 AMD Radeon Instinct MI250X GPUs (144,179,200 cores)

HPC systems performance



- The performance of HPC systems is measured in **FLOPs/s** (Floating Point Operations per second).
- The "Top500" list is published twice per year and ranks the top 500 HPC systems globally in terms of performance.

https://www.top500.org/statistics/overtime/

HPC systems performance

- The performance of HPC systems is measured in **FLOPs/s** (Floating Point Operations per second).
- The "Top500" list is published twice per year and ranks the top 500 HPC systems globally in terms of performance.
- The first exascale (10^{18} FLOP/s) HPC system went online earlier this year (Frontier).

2

High Performance Computing in Cyprus

Cyprus Institute - High Performance Facility

- The highest performance computing facility of the island and regionally competitive.
- Hosted at the Cyprus Institute.
- Heterogeneous design
  - CPU nodes
  - GPU nodes
- Combination of clusters with varying architectures.

Cyprus Institute - High Performance Facility

Cyclone cluster

- 7 40-core compute nodes
- 16 40-core compute nodes and 4 NVidia V100 GPUs each
- 2 20-core sockets with Intel Xeon Gold 6248
- 192 GB memory per node

AMD Epyc cluster (**rome & milan**)

- 8 128-core compute nodes
- 2 64-core sockets with AMD EPYC 7702
- 256 GB memory per node

Cyclamen cluster

- 8 32-core compute nodes and 2 NVidia P100 GPUs each
- 2 16-core sockets with Intel Xeon G6130
- 128 GB memory per node

Cy-Tera cluster (legacy 2012)

- 98 12-core compute nodes
- 18 12-core compute nodes and 2 NVidia M2070 GPUs each (deprecated)
- 2 6-core sockets with Intel Westmere X5650
- 48 GB memory per node

Cyprus Institute - High Performance Facility

- The highest performance computing facility of the island and regionally competitive.
- Hosted at the Cyprus Institute.
- Heterogeneous design
  - CPU nodes
  - GPU nodes
- Combination of clusters with varying architectures.
- Open to you to access to meet your computational needs.

Cyprus Institute - High Performance Facility

Preparatory access

- Projects whose software codes require porting, scalability testing, development or specialised assistance.
- These projects undergo only a light technical review. The upper limit for allocations for such projects is 20,000 core hours and 1000 GPU hours and access is given on **Cyclone** system.

- **https://hpcf.cyi.ac.cy/apply.html**

Production access

- Production access is intended for production-ready projects where significant amounts of computing resources are required.
- All proposals are assessed by technical evaluation on the suitability and compatibility of the project with the requested computing resources
  - 350,000 CPU core hours
  - 35,000 GPU hours
  - Or a combination of the above

3

How to use Cyclone

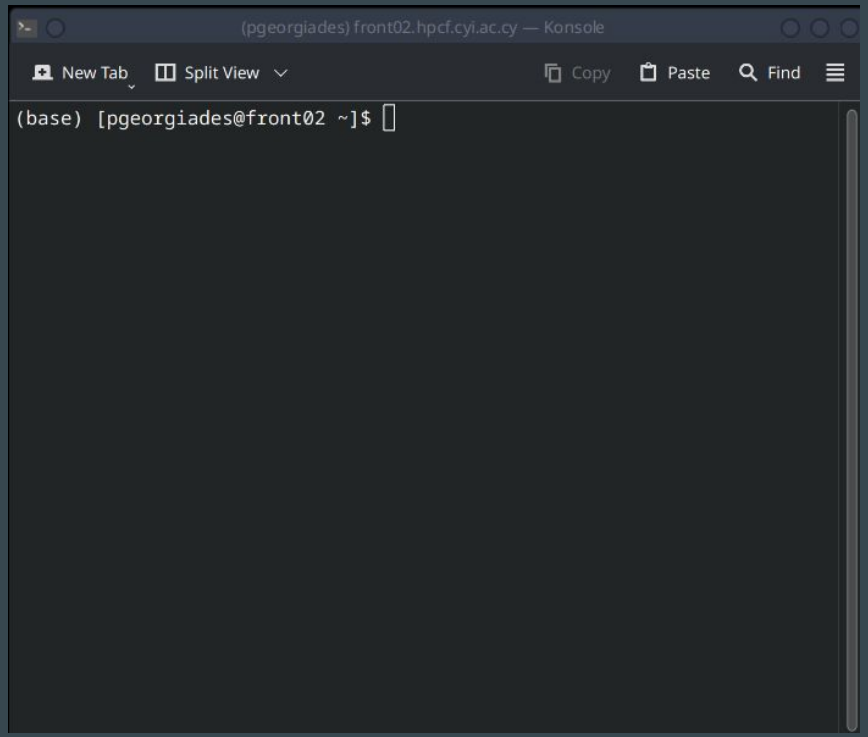
Cyprus Institute - High Performance Facility

- As with the vast majority of HPC systems across the world, the CyI's systems use the **linux** operating system.

Meet Tux, the mascot of linux!

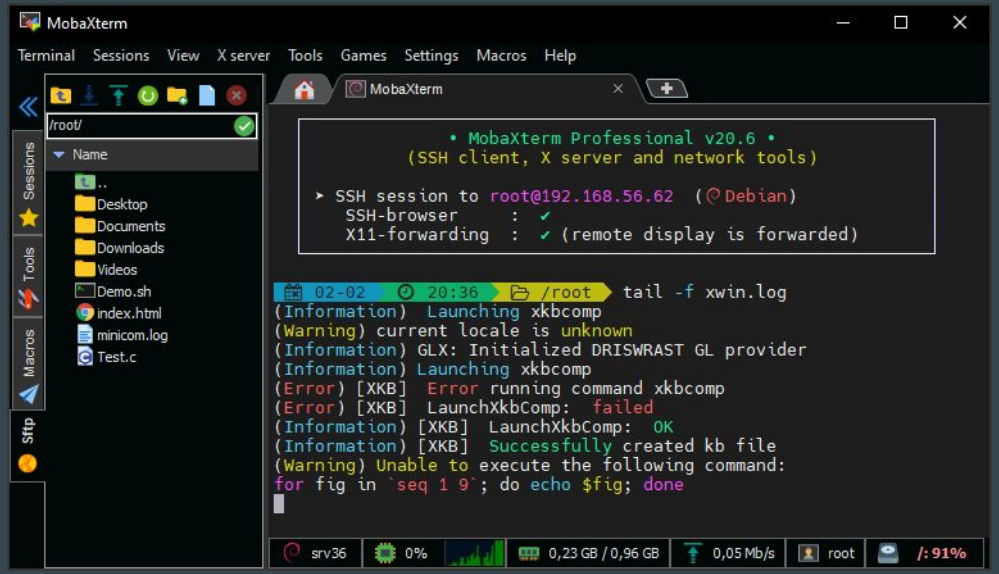
Cyprus Institute - High Performance Facility

- As with the vast majority of HPC systems across the world, the CyI's systems use the **linux** operating system.
- There is no graphical user interface on the HPC system, so users should be comfortable with using the **terminal** and the **bash** shell (don't worry, it's not as awful as it sounds!).



Cyprus Institute - High Performance Facility

- As with the vast majority of HPC systems across the world, the CyI's systems use the **linux** operating system.
- There is no graphical user interface on the HPC system, so users should be comfortable with using the **terminal** and the **bash** shell (don't worry, it's not as awful as it sounds!).
- There are, however, several tools with a graphical interface you can use while connected to the HPC system to aid you
  - VS Code (code development IDE with A LOT of useful plugins)
  - Mobaxterm (ssh client)
  - FileZilla (SFTP client)



Cyclone - ssh

- **ssh** (secure shell) is used to connect to the HPC system.
- In linux and MacOS (and UNIX systems) we can simply use the terminal
- In windows there are a few popular ssh clients. e.g.
  - putty
  - Mobaxterm

Cyclone - ssh

- **ssh** (secure shell) is used to connect to the HPC system.
- In linux and MacOS (and UNIX systems) we can simply use the terminal
- In windows there are a few popular ssh clients. e.g.
  - putty
  - Mobaxterm

Hands on session

Use your preferred ssh client to connect to cyclone.

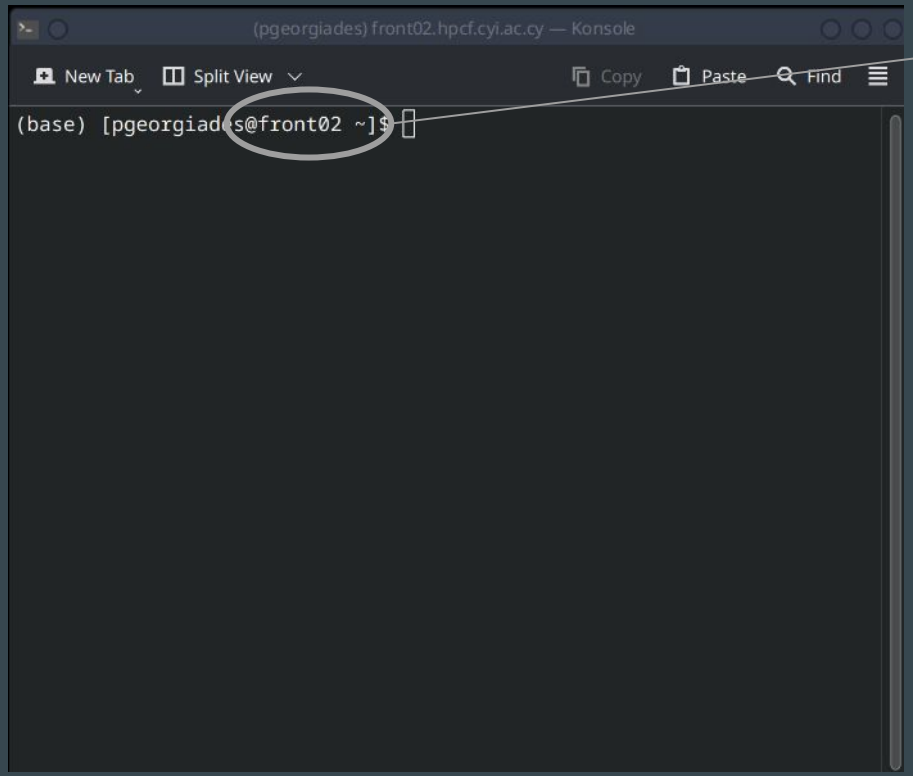
Linux/MacOS:
In the terminal enter:

ssh <username>@front02.hpcf.cyi.ac.cy

(make sure your ssh key is in the .ssh directory with the right permissions)

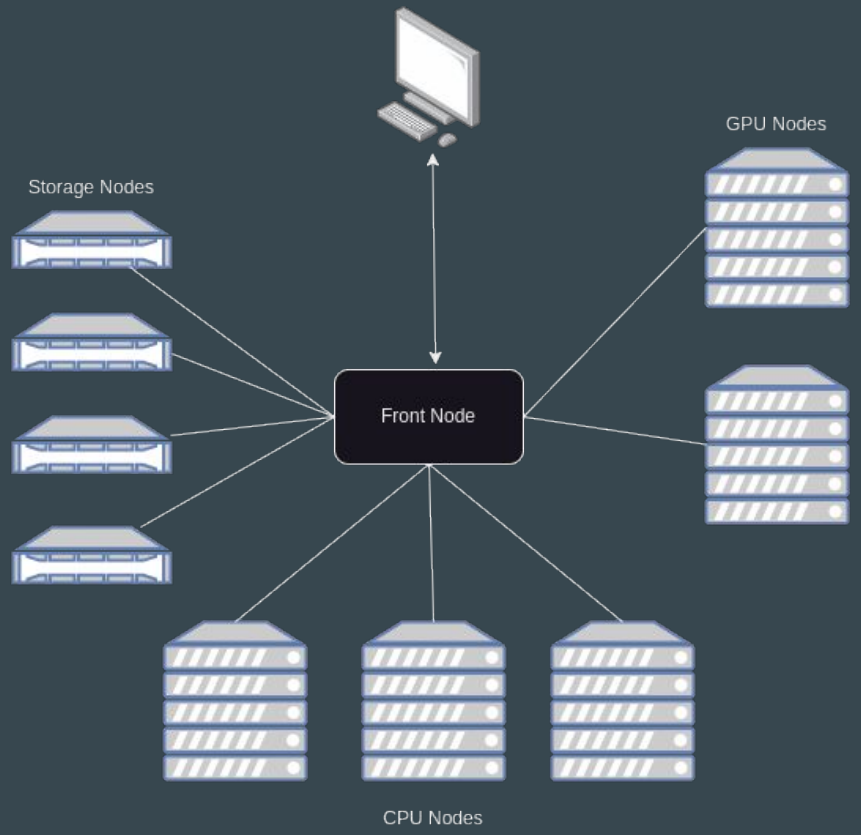
Windows:
Follow instructions on
https://hpcf.cyi.ac.cy/documentation/login.html

Cyclone - ssh



When we ssh to cyclone we are connected to the **front node**.

Cyclone - ssh



When we ssh to cyclone we are connected to the **front node**.

HPC systems are a collection of nodes, which are controlled by the front node (or head node), connected via **infiniband** (a computer networking communications standard used in high performance computing that features very high throughput and very low latency).

CAUTION: You should never run any compute tasks on the front node!

Cyclone - **SLURM**

- When we ssh to cyclone we are connected to the **front node**.
- From there, we can use the Simple Linux Utility for Resource Management (**SLURM**), an open-source workload manager designed for Linux clusters of all sizes.

Cyclone - **SLURM**

- When we ssh to cyclone we are connected to the **front node**.
- From there, we can use the Simple Linux Utility for Resource Management (**SLURM**), an open-source workload manager designed for Linux clusters of all sizes.

Hands on

Run the "**sinfo**" command to view the available partitions of the CyI HPC system.

Cyclone - **SLURM**

```
(base) [pgeorgiades@front02 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
milan        up 1-00:00:00      1 drain~ ne32
milan        up 1-00:00:00     14  idle~ ne[06,12-19,26,28,30-31,33]
milan        up 1-00:00:00     12  alloc ne[01,03,07-11,20-21,23-25]
milan        up 1-00:00:00      1   resv ne02
milan        up 1-00:00:00      4    mix ne[04-05,22,34]
milan        up 1-00:00:00      1   idle ne27
rome         up 7-00:00:00      3  idle~ a[03-05]
rome         up 7-00:00:00      1   idle a02
skylake      up 1-00:00:00      2  idle~ dis[03-04]
skylake      up 1-00:00:00      1   idle dis02
nehalem      up 1-00:00:00     16  idle~ cwg02,cwp02,g[11-12],n[018-024,028],ph[01-04]
nehalem      up 1-00:00:00      1  down~ g07
nehalem      up 1-00:00:00      2   idle cwg01,cwp01
cpu*         up 1-00:00:00      2   plnd cn[01-02]
cpu*         up 1-00:00:00      1  down~ cn05
cpu*         up 1-00:00:00     14    mix cn[03-04,06-17]
p100         up 1-00:00:00      1  down~ cyc05
p100         up 1-00:00:00      6  idle~ cyc[02-04,06-08]
p100         up 1-00:00:00      1   idle cyc01
a100         up 1-00:00:00      4  idle~ sim[03-06]
a100         up 1-00:00:00      1    mix sim02
a100         up 1-00:00:00      1   idle sim01
gpu          up 1-00:00:00      1   plnd gpu01
gpu          up 1-00:00:00      1  idle~ gpu02
gpu          up 1-00:00:00      7    mix gpu[03,05-06,08-09,12-13]
gpu          up 1-00:00:00      7  alloc gpu[04,07,10-11,14-16]
```

These are the partitions available in the system:

- **cpu**: default Cyclone CPU nodes partition
- **gpu**: Cyclone GPU nvidia v100 nodes partition
- **nehalem**: Cytera CPU nodes partition
- **a100**: Simea GPU nvidia a100 nodes partition
- **skylake**: Disarm nodes partition
- **milan**: AMD milan nodes partition
- **p100**: Cyclamen GPU nvidia p100 nodes partition

Cyclone - **SLURM**

```
(base) [pgeorgiades@front02 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
milan        up 1-00:00:00      1 drain~ ne32
milan        up 1-00:00:00     14  idle~ ne[06,12-19,26,28,30-31,33]
milan        up 1-00:00:00     11  alloc ne[01,03,07-11,20-21,23-25]
milan        up 1-00:00:00      1   resv ne02
milan        up 1-00:00:00      4    mix ne[04-05,22,34]
milan        up 1-00:00:00      1   idle ne27
rome         up 7-00:00:00      3  idle~ a[03-05]
rome         up 7-00:00:00      1   idle a02
skylake      up 1-00:00:00      2  idle~ dis[03-04]
skylake      up 1-00:00:00      1   idle dis02
nehalem      up 1-00:00:00     16  idle~ cwg02,cwp02,g[11-12],n[018-024,028],ph[01-04]
nehalem      up 1-00:00:00      1  down~ g07
nehalem      up 1-00:00:00      2   idle cwg01,cwp01
cpu*         up 1-00:00:00      2   plnd cn[01-02]
cpu*         up 1-00:00:00      1  down~ cn05
cpu*         up 1-00:00:00     14    mix cn[03-04,06-17]
p100         up 1-00:00:00      1  down~ cyc05
p100         up 1-00:00:00      6  idle~ cyc[02-04,06-08]
p100         up 1-00:00:00      1   idle cyc01
a100         up 1-00:00:00      4  idle~ sim[03-06]
a100         up 1-00:00:00      1    mix sim02
a100         up 1-00:00:00      1   idle sim01
gpu          up 1-00:00:00      1   plnd gpu01
gpu          up 1-00:00:00      1  idle~ gpu02
gpu          up 1-00:00:00      7    mix gpu[03,05-06,08-09,12-13]
gpu          up 1-00:00:00      7  alloc gpu[04,07,10-11,14-16]
```

These are the partitions available in the system:

- **cpu**: default Cyclone CPU nodes partition
- **gpu**: Cyclone GPU nvidia v100 nodes partition
- **nehalem**: Cytera CPU nodes partition
- **a100**: Simea GPU nvidia a100 nodes partition
- **skylake**: Disarm nodes partition
- **milan**: AMD milan nodes partition
- **p100**: Cyclamen GPU nvidia p100 nodes partition

You can also check the state of the various nodes in each partition, i.e. **drain, idle, alloc, resv, mix, down**

Cyclone - **SLURM (useful commands)**

- **squeue**: The **squeue** command will report the state of running and pending jobs. You can use this command to find out which node your job is running on.

Cyclone - **SLURM (useful commands)**

- **squeue**: The **squeue** command will report the state of running and pending jobs. You can use this command to find out which node your job is running on.

Hands on

Run the "**squeue**" command. This shows all the running and pending jobs on the HPC system.

To view your submitted jobs:
squeue -u <username>

Cyclone - **SLURM (useful commands)**

- **squeue**: The **squeue** command will report the state of running and pending jobs. You can use this command to find out which node your job is running on.
- **salloc** is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute **srun** commands to launch parallel tasks.
  - In order to open an interactive shell on a node we need to specify a number of parameters:

Hands on

Run the "**salloc -h**" command to see all the available options for **salloc**.

Adding -h (or --help) after any terminal command provides information on built-in commands.

Cyclone - **SLURM (useful commands)**

- **squeue**: The **squeue** command will report the state of running and pending jobs. You can use this command to find out which node your job is running on.
- **salloc** is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute **srun** commands to launch parallel tasks.
  - In order to open an interactive shell on a node we need to specify a number of parameters. e.g.
    - `-A, --account=name  charge job to specified account`
    - `-J, --job-name=jobname  name of job`
    - `-N, --nodes=N  number of nodes on which to run (N = min[-max])`
    - `--ntasks-per-node=n  number of tasks to invoke on each node`
    - `-p, --partition=partition  partition requested`
  - To access the nodes reserved for this session
    - **--reservation=eurocc-cpu**

Cyclone - **SLURM (useful commands)**

- **salloc** is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute **srun** commands to launch parallel tasks.
  - In order to open an interactive shell on a node we need to specify a number of parameters:
    - `-A, --account=name` charge job to specified account
    - `-J, --job-name=jobname` name of job
    - `-N, --nodes=N` number of nodes on which to run (N = min[-max])
    - `--ntasks-per-node=n` number of tasks to invoke on each node
    - `-p, --partition=partition` partition requested
    - `-t, --time=minutes` time limit
  - To access the nodes reserved for this session
    - **--reservation=eurocc-cpu**

Hands on - Exercise

Ask for a real time shell on a **cpu** node for **10 min, 1 node, 1 cpu**. The account name for this event is **edu18**. Name your session **test_session**.

Cyclone - **SLURM (useful commands)**

- **salloc** is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute **srun** commands to launch parallel tasks.
  - In order to open an interactive shell on a node we need to specify a number of parameters:
    - `-A, --account=name  charge job to specified account`
    - `-J, --job-name=jobname  name of job`
    - `-N, --nodes=N  number of nodes on which to run (N = min[-max])`
    - `--ntasks-per-node=n  number of tasks to invoke on each node`
    - `-p, --partition=partition  partition requested`
    - `-t, --time=minutes  time limit`
  - To access the nodes reserved for this session
    - **--reservation=eurocc-cpu**

Hands on - Exercise

Ask for a real time shell on a **cpu** node for **10 min, 1 node, 1 cpu**. The account name for this event is **edu18**. Name your session **test_session**.

>> salloc --account=edu18 --partition=cpu --time=10 --nodes=1 --ntasks-per-node=1 --reservation=eurocc-cpu

Cyclone - **modules**

- The **module** system is a concept available on most supercomputers, simplifying the use of different software (versions) in a precise and controlled manner.
  →https://hpc-wiki.info/hpc/Modules
  - **module avail** to view the available modules on the system
  - **module spider <name>** to search for a specific module
  - **module load <name>** to load a module

Cyclone - **modules**

- The **module** system is a concept available on most supercomputers, simplifying the use of different software (versions) in a precise and controlled manner.
  →https://hpc-wiki.info/hpc/Modules
  - **module avail** to view the available modules on the system
  - **module spider <name>** to search for a specific module
  - **module load <name>** to load a module

Hands on

In your real time shell load the Python 3.10 module:

>> module avail | grep Python
>> module load SciPy-bundle/2019.10-foss-2019b-Python-3.7.4

- This module has useful packages installed already.
- **conda** and **pyenv** environments are quite popular but beware of conda, it creates a large number of files and we all have a limit on how many files we can store in our home directory

Cyclone - **cloning github repositories**

- **github** is a platform and cloud-based service for software development and version control using Git, allowing developers to store and manage their code.
- To clone (copy) a github repository on the HPC system (and unix-based systems in general) you can use the

  git clone <github url>

  command

Hands on

- Clone the event's github repository in your home directory and access it.

```
>> git clone https://github.com/CaSToRC-CyI/EuroCC2_training_Nov23
>> cd EuroCC2_training_Nov23
```

Cyclone - **How to run scripts**

- In a real time shell using **salloc** (not recommended)
- Using **sbatch** to submit a batch script to Slurm. https://slurm.schedmd.com/sbatch.html
  - It will run when the requested resources are available.

Cyclone - **How to run scripts**

- In a real time shell using **salloc** (not recommended)
- Using **sbatch** to submit a batch script to Slurm. https://slurm.schedmd.com/sbatch.html
  - It will run when the requested resources are available.

Hands on

- Navigate to **Intro_HPC** in the **EuroCC2_training_Nov23** directory.

>> cd EuroCC2_training_Nov23/Intro_HPC

- List the contents of the directory

>> ll

Cyclone - **How to run scripts**

- In a real time shell using **salloc** (not recommended)
- Using **sbatch** to submit a batch script to Slurm. https://slurm.schedmd.com/sbatch.html
  - It will run when the requested resources are available.

Hands on

- Navigate to **Intro_HPC** in the **EuroCC2_training_Nov23** directory.

>> cd EuroCC2_training_Nov23/Intro_HPC

- List the contents of the directory

>> ll

```
(base) [pgeorgiades@front02 Intro_HPC]$ ll
total 4
-rw-r--r-- 1 pgeorgiades eewrc  22 Nov  7 15:01 Hello_world.py
-rw-r--r-- 1 pgeorgiades eewrc 865 Nov  7 18:15 pi_1.py
-rw-r--r-- 1 pgeorgiades eewrc 627 Nov  7 18:15 pi_2.py
-rw-r--r-- 1 pgeorgiades eewrc 343 Nov  8 13:50 submit_job.sh
-rw-r--r-- 1 pgeorgiades eewrc 343 Nov  8 13:50 submit_job2.sh
(base) [pgeorgiades@front02 Intro_HPC]$
```

Cyclone - **How to run scripts**

- In a real time shell using **salloc** (not recommended)
- Using **sbatch** to submit a batch script to Slurm. https://slurm.schedmd.com/sbatch.html
  - It will run when the requested resources are available.

Hands on

- Run the Hello_world.py script.

>> python Hello_world.py

Cyclone - **How to run scripts**

- In a real time shell using **salloc** (not recommended)
- Using **sbatch** to submit a batch script to Slurm. https://slurm.schedmd.com/sbatch.html
  - It will run when the requested resources are available.

Hands on

- Run the **cat** command on the pi_1.py script to view its contents.

>> cat pi_1.py

```
"""
Perform a Monte Carlo simulation to calculate pi
"""
import random
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--n", help="Number of iterations",
                    type=int)
args = parser.parse_args()

# Generate two vectors of n random numbers in the 0-1 range
# First create two empty lists
x = []
y = []

# Populate them
for i in range(args.n):
    x.append(random.random())
    y.append(random.random())

# Create a new list to note the points which are a maximum of
# 1 unit away from the origin (0, 0)
distances = []
for i in range(args.n):
    if (x[i]**2 + y[i]**2)**0.5 < 1:
        distances.append(1)
    else:
        distances.append(0)

# Sum up the ones in the distances list
dist_sum = 0
for i in range(args.n):
    dist_sum += distances[i]

# Finally calculate pi
pi = 4*(dist_sum/args.n)
print(f"pi = {pi}")
```

Cyclone - **How to run scripts**

- In a real time shell using **salloc** (not recommended)
- Using **sbatch** to submit a batch script to Slurm. https://slurm.schedmd.com/sbatch.html
  - It will run when the requested resources are available.

Hands on

- Run the **cat** command on the pi_1.py script to view its contents.

>> cat pi_1.py
>> python pi_1.py –n 1000

Run for 1000 iterations

```
(base) [pgeorgiades@front02 Intro_HPC]$ python pi_1.py --n 1000
pi = 3.096
(base) [pgeorgiades@front02 Intro_HPC]$
```

Cyclone - **How to run scripts**

- In a real time shell using **salloc** (not recommended)
- Using **sbatch** to submit a batch script to Slurm. https://slurm.schedmd.com/sbatch.html
  - It will run when the requested resources are available.

Hands on

- Now exit the live shell

```
>> cat pi_1.py
>> python pi_1.py –n 1000
>> exit
```

Cyclone - **How to run scripts**

- The recommended way is to submit your job to SLURM to run when the requested resources are available.

Cyclone - **How to run scripts**

- The recommended way is to submit your job to SLURM to run when the requested resources are available, using **sbatch**.
- The batch script may contain options preceded with "#SBATCH" before any executable commands in the script.
- **sbatch** will stop processing further #SBATCH directives once the first non-comment non-whitespace line has been reached in the script.

```
#!/bin/bash
#SBATCH --job-name=HPC_Intro
#SBATCH --nodes=1
#SBATCH -o runner.%J.out
#SBATCH --time=00:02:00
#SBATCH --ntasks-per-node=1
#SBATCH -A edu18
#SBATCH --partition=cpu
#SBATCH --reservation=eurocc-cpu


module load
SciPy-bundle/2019.10-foss-2019b-Python-3.7.4

printf "\nCalculating pi for $1 iterations\n\n"

time python pi_1.py --n $1
```

Hands on

- Run the **cat** command on the submit_job.sh script.

>> cat submit_job.sh

Cyclone - **How to run scripts**

- The recommended way is to submit your job to SLURM to run when the requested resources are available, using **sbatch**.
- The batch script may contain options preceded with "#SBATCH" before any executable commands in the script.
- **sbatch** will stop processing further #SBATCH directives once the first non-comment non-whitespace line has been reached in the script.

Shell scripts start with this

```
#!/bin/bash
#SBATCH --job-name=HPC_Intro
#SBATCH --nodes=1
#SBATCH -o runner.%J.out
#SBATCH --time=00:02:00
#SBATCH --ntasks-per-node=1
#SBATCH -A edu18
#SBATCH --partition=cpu
#SBATCH --reservation=eurocc-cpu
```

The output is directed to a .out file in the same directory.

Same arguments we used for **salloc**

```
module load
SciPy-bundle/2019.10-foss-2019b-Python-3.7.4

printf "\nCalculating pi for $1 iterations\n\n"

time python pi_1.py --n $1
```

Shell argument to bash script

Shell argument to python script

Hands on

- Run the **cat** command on the submit_job.sh script.

>> cat submit_job.sh

Cyclone - **How to run scripts**

- The recommended way is to submit your job to **SLURM** to run when the requested resources are available, using **sbatch**.
- The batch script may contain options preceded with "#SBATCH" before any executable commands in the script.
- **sbatch** will stop processing further #SBATCH directives once the first non-comment non-whitespace line has been reached in the script.
- Now let's submit our first job to **SLURM.**

Hands on - Exercise

- Submit the pi_1.py using **sbatch** to calculate pi (1000 iterations).

>> sbatch submit_job.sh 1000

- List the contents of the directory and view the contents of the .out file created after the job has finished.

Cyclone - **How to run scripts**

- The recommended way is to submit your job to **SLURM** to run when the requested resources are available, using **sbatch**.
- The batch script may contain options preceded with "#SBATCH" before any executable commands in the script.
- **sbatch** will stop processing further #SBATCH directives once the first non-comment non-whitespace line has been reached in the script.
- Now let's submit our first job to **SLURM.**

Hands on - Exercise

- Submit the pi_1.py using **sbatch** to calculate pi (1000 iterations).

>> sbatch submit_job.sh 1000

- List the contents of the directory and view the contents of the .out file created after the job has finished.
- Submit the script again but now run it for 1 000 000 iterations. Compare the execution time between 1000 and 1000000 iterations.

Cyclone - **HPC is not the silver bullet!**

- Using the HPC systems is not always the answer though.

Hands on

- Print the pi_2.py script on the shell (use the **cat** command)

```
"""
Perform a Monte Carlo simulation to calculate pi
Numpy version
"""
import numpy as np
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--n", help="Number of iterations",
                    type=int)
args = parser.parse_args()

# Populate two numpy arrays with n random numbers in the 0-1 range
x, y = np.random.random(args.n), np.random.random(args.n)
# Now calculate the distances from the origin
distances = np.sqrt(x**2 + y**2)
# Sum the number of points that lie below distance=1
dist_sum = np.sum(np.where(distances < 1, 1, 0))
# Finally calculate pi
pi = 4*(dist_sum/args.n)
print(f"pi = {pi}")
```

Cyclone - **HPC is not the silver bullet!**

- Using the HPC systems is not always the answer though.

Hands on - Exercise

- Print the pi_2.py script on the shell (use the **cat** command)
- Use the submit_job2.sh bash script to submit the pi_2.py scrint to SLURM for 1000 and 1000000 iterations.
- Compare the time taken by the two scripts for the two calculations.

Cyclone - **HPC is not the silver bullet!**

- Using the HPC systems is not always the answer though.

Hands on - Exercise

- Print the pi_2.py script on the shell (use the **cat** command)
- Use the submit_job2.sh bash script to submit the pi_2.py scrint to SLURM for 1000 and 1000000 iterations.
- Compare the time taken by the two scripts for the two calculations.

```
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 1000
pi = 3.152

real    0m0.090s
user    0m0.019s
sys     0m0.027s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 1000
pi = 3.184

real    0m0.334s
user    0m0.106s
sys     0m0.058s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 1000000
pi = 3.141764

real    0m0.911s
user    0m0.816s
sys     0m0.057s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 1000000
pi = 3.14202

real    0m0.307s
user    0m0.132s
sys     0m0.058s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 10000000
pi = 3.1422748

real    0m6.299s
user    0m5.908s
sys     0m0.341s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 10000000
pi = 3.1416432

real    0m0.530s
user    0m0.298s
sys     0m0.113s
```

Cyclone - **HPC is not the silver bullet!**

- Using the HPC systems is not always the answer though.

```
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 1000
pi = 3.152

real    0m0.090s
user    0m0.019s
sys     0m0.027s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 1000
pi = 3.184

real    0m0.334s
user    0m0.106s
sys     0m0.058s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 1000000
pi = 3.141764

real    0m0.911s
user    0m0.816s
sys     0m0.057s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 1000000
pi = 3.14202

real    0m0.307s
user    0m0.132s
sys     0m0.058s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 10000000
pi = 3.1422748

real    0m6.299s
user    0m5.908s
sys     0m0.341s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 10000000
pi = 3.1416432

real    0m0.530s
user    0m0.298s
sys     0m0.113s
```

Why do you think this happens?

Cyclone - **HPC is not the silver bullet!**

- Using the HPC systems is not always the answer though.
- Make sure your code is optimised before running it on the HPC system.
  - HPC systems are expensive to maintain and operate (both in human labour and monetary cost)
  - HPC resources are limited
  - We want to use these resources as efficiently as possible

```
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 1000
pi = 3.152

real    0m0.090s
user    0m0.019s
sys     0m0.027s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 1000
pi = 3.184

real    0m0.334s
user    0m0.106s
sys     0m0.058s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 1000000
pi = 3.141764

real    0m0.911s
user    0m0.816s
sys     0m0.057s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 1000000
pi = 3.14202

real    0m0.307s
user    0m0.132s
sys     0m0.058s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_1.py --n 10000000
pi = 3.1422748

real    0m6.299s
user    0m5.908s
sys     0m0.341s
(ufp2) [pgeorgiades@ne27 Intro_HPC]$ time python pi_2.py --n 10000000
pi = 3.1416432

real    0m0.530s
user    0m0.298s
sys     0m0.113s
```

Why do you think this happens?

THANKS!

ANY QUESTIONS?

You can find me at:

- p.georgiades@cyi.ac.cy