

Getting Started With GLiCID: Advanced Session

Mir Junaid

January 18, 2024



OUTLINE

- Why HPC?
- Guix Package Manager
- Modules
- SLURM Workload Manager
 - Why do we need Slurm?
 - Slurm Configuration Options
 - Example Slurm Script
 - TP
 - Basic Slurm Script
 - Slurm for Parallel Programming
 - Install Conda/Micromamba
 - Fortran: Hello World
- Apptainer Containers (Next PPT)

Why HPC?



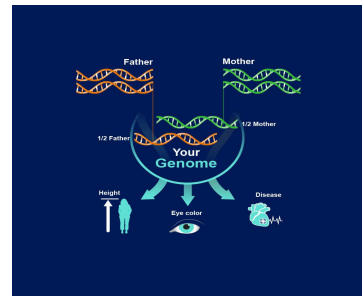
Q. Why would I be interested in High Performance Computing (HPC)?

Frequently, research problems that use computing can outgrow the capabilities of the desktop or laptop. For example,

Use Case 1: AI/ML/Statistics

- A statistics/data science researcher wants to cross-validate a model.
- This involves running the model 1000 times – but each run takes an hour.
- Running the model on a laptop will take over a month.
- In this research problem, final results are calculated after all 1000 models have run, but typically only one model is run at a time (in serial) on the laptop.
- Since each of the 1000 runs is independent of all others, and **given enough computers, it's theoretically possible to run them all at once (in parallel) and complete the task in one hour.**

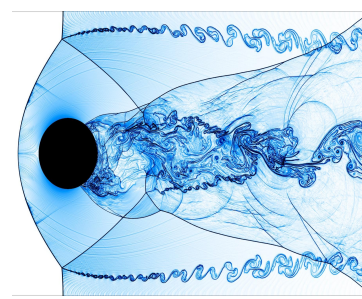
Why HPC?



Use Case 2: Genomics

- A genomics researcher has been using small datasets of sequence data, but some type of sequencing datasets are huge.
- It's challenging to open these datasets on a computer – analyzing these larger datasets will probably crash it.
- In this research problem, **the calculations required might be impossible to parallelize, but a computer with more memory would be required** to analyze the much larger future data set.

Why HPC?



Use Case 3: Fluid Dynamics/Mechanics

- An engineer using a fluid dynamics package has an option to run in parallel.
- In going from 2D to 3D simulations, the simulation time has more than tripled.
- In this research problem, the calculations in each region of the simulation are largely independent of calculations in other regions of the simulation.
- It's possible to run each region's calculations simultaneously (in parallel), communicate selected results to adjacent regions as needed, and repeat the calculations to converge on a final set of results.
- In moving from a 2D to a 3D model, both **the amount of data and the amount of calculations increases greatly, and it's theoretically possible to distribute the calculations across multiple computers communicating over a shared network.**

In all these cases, access to more computers with larger memories is needed.

Software Modules

Software Modules

- Modules
 - Lot of useful software packages
 - Different versions
 - Maintained by experts
 - Optimized for the architecture
 - Users cannot install a module
 - Have to request the administrator

How to use Modules?

- Useful commands

Command	Description
<code>module avail</code>	List modules
<code>module avail <module_name></code>	List all installed versions of python
<code>module load <module_name></code>	Load the default python version
<code>module load <module_name/3.11.5></code>	Load a specific version of python
<code>module unload <module_name></code>	Unload python
<code>module list</code>	List currently loaded modules

Guix Package Manager



What is Guix?

- Package building system/Package manager
- **Why Guix? Why is it better than modules?**
 - Allows each user to manage his/her own packages
 - without root privilege
 - without interfering with other users
 - Easy creation of isolated environments with designated packages
 - useful for per-project dependency management





Guix Package Manager

- Useful commands

Command	Description
<code>guix pull</code>	You need to run this at least once(maybe weekly :p)
<code>guix search <package_name></code>	Look for a package to install
<code>guix install <package_name></code>	To install a package
<code>guix remove <package_name></code>	To remove a package
<code>guix package -l</code>	List of installed packages

Search packages here <https://packages.guix.gnu.org/>

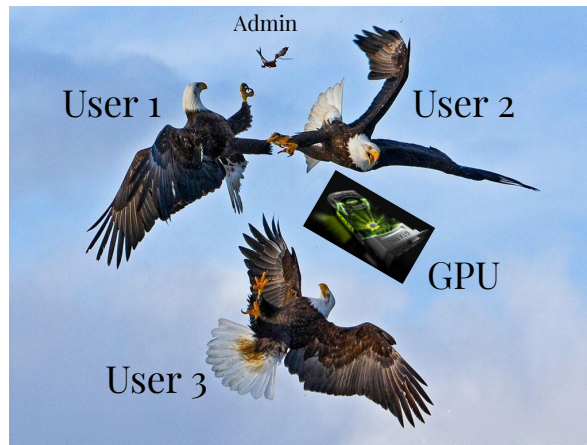
Note: To use Guix on Nautilus, load the guix module (module load guix)



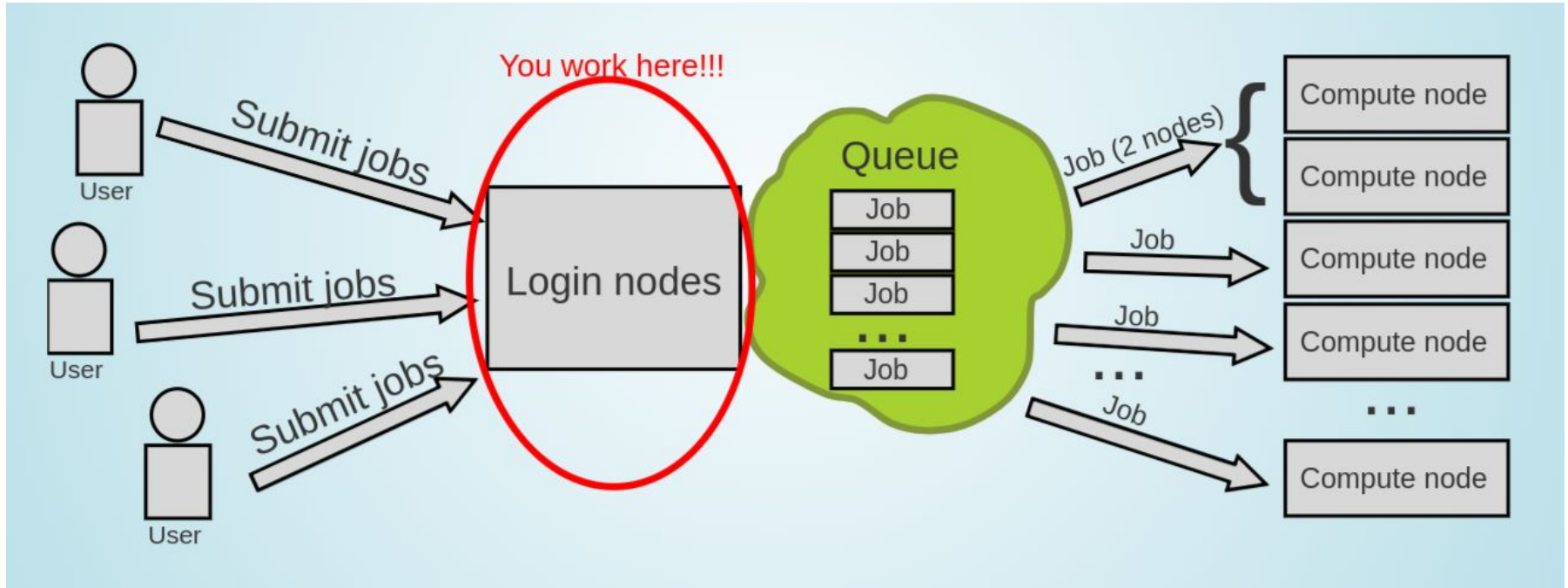
Bringing Order To Chaos

Competition for limited resources

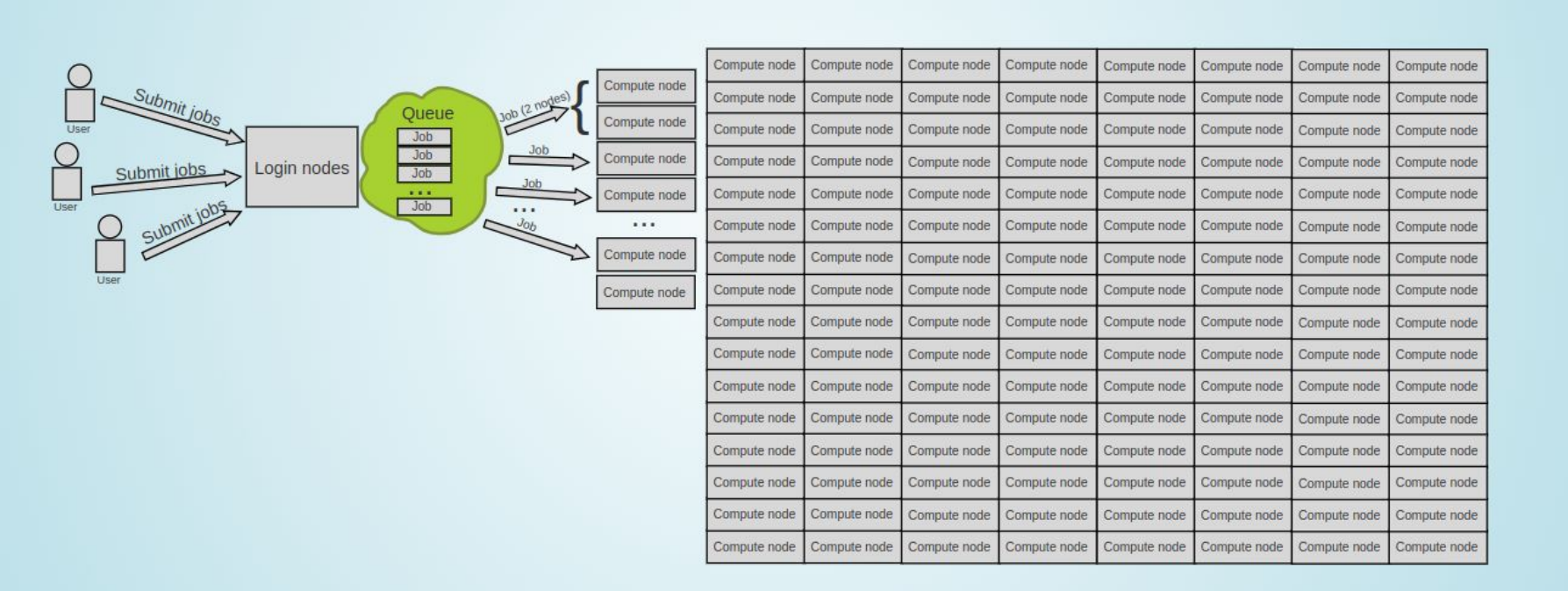
- On computing cluster, people compete to use a finite set of resources (CPUs/GPUs/RAM)
- If everyone just starts running code, then everyone will have a bad time as resources are shared
- To solve this problem, computing centers use resource manager and job scheduler called [Slurm](#)
- With Slurm, you can submit jobs and tell Slurm what resources you need
- Slurm will allocate those resources to your job and then schedule your job



This is how it works



But you don't use the whole Supercomputer

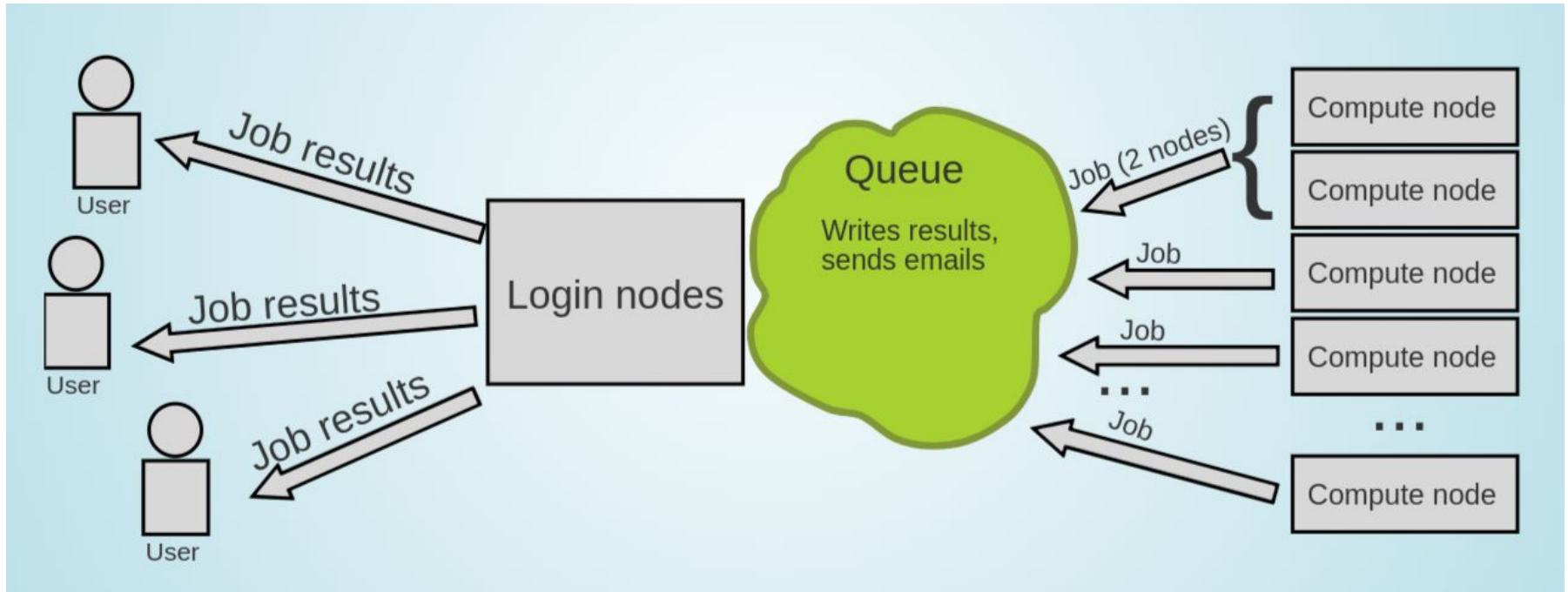


Enter the queue, and wait

- Your job(s) enter the queue, and wait for its turn
- When there are enough resources for that job, it runs



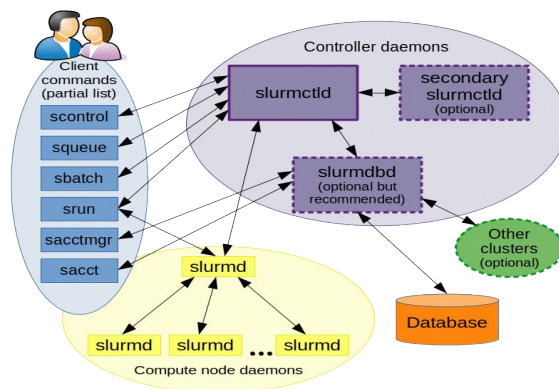
Results



SLURM - Workload Manager/Job Scheduler



- **Simple Linux Utility for Resource Management (SLURM)**
- Open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters
- It has centralized manager, **slurmctld**, to monitor resources and work
- Each compute node has a **slurmd daemon**, which can be compared to a remote shell: it waits for work, executes that work, returns status, and waits for more work.



Getting Started with Slurm



- To tell Slurm what resources you need, you will have to create an sbatch script/ Slurm script
- The sbatch scripts generally follow this format:

```
#!/bin/bash
```

```
# Declaring Slurm Configuration Options
```

```
# Loading Software/Libraries
```

```
# Running Code
```

- Note: `#!/bin/bash` above tells our terminal what program to run this file with. In this case, `bash`.
- You can write an sbatch script in any language as long as `#SBATCH` doesn't result in errors
- Examples: Ruby, Python, Bash, R.

Configuration Options for Slurm

- There are many configuration options for Slurm
- Some options are cluster specific and may not work
- We can help you find the best set of configurations for your computing needs
- Configuration options are specified in your sbatch script like this:

```
#SBATCH <option_1>=<value>  
#SBATCH <option_2>=<value>  
...  
#SBATCH <option_3>=<value>
```

- Note the pound sign(#) is not the comment here.
- Slurm looks for lines starting with **#SBATCH** so it can find configuration options

Accounting Configurations

- **Job Name:** `#SBATCH --job-name=<job_name>`
 - First thing you need to do is give your job a name and it should be descriptive
 - Example: `#SBATCH --job-name=RandomWalk`
 - The point of the job name is to remind yourself what you are doing
 - If it is not descriptive, you can easily get confused
- **Comment:** `#SBATCH --comment=<comment>`
 - To extend the description of your job, add a comment
 - Example: `#SBATCH --comment="To explore the nodes."`
- **Account:** `#SBATCH --account=<account_name>`
 - You need to tell Slurm which account to run your job under
 - This is not user account, but your project account
 - Example: `#SBATCH --account=glicid`

Accounting Configurations

- **Partition:** `#SBATCH --partition=<Partition_name>`
 - Slurm needs to know which partition to run your job on
 - Example: `#SBATCH --partition=standard`
 - Each partition has access to different resources and has a specific use case
- **Time Limit:** `#SBATCH --time=D-HH:MM:SS`
 - You need to tell Slurm how long your job needs to run
 - The format is Days-Hours:Minutes:Seconds
 - Example: `#SBATCH --time=1-12:30:00` (1 Day, 12 Hours, 30 Minutes, 0 Seconds)

Job Output Configurations

- **Output File:** `#SBATCH --output=%x_%j.out`
 - Any output from your compute job will be saved to the output file that you specify
 - `%x` is a variable that fills in your job name. `%j` is a variable that fills in your job ID number
 - Example: `#SBATCH --output=logs/%x_%j.out`
- **Error File:** `#SBATCH --error=%x_%j.err`
 - Any errors from your compute job will be saved to the error file that you specify
 - `%x` is a variable that fills in your job name. `%j` is a variable that fills in your job ID number
 - Example: `#SBATCH --error=logs/%x_%j.err`

Node Configurations

- A node is just a computer in a cluster
- Most of the time, it probably makes sense to only use one node
- **Nodes:** `#SBATCH --nodes=<num_nodes>`
 - The default is 1 node, so if you're using 1 node, you don't need to specify it in configuration
 - We recommend that you include it to remind yourself what resources your job is using
 - Example: `#SBATCH --nodes=4`
- **Excluding Nodes:** `#SBATCH --exclude=<node1, node2, ...>`
 - If for some reason you want to make sure your job does not run a specific node
 - Example: `#SBATCH --exclude=cnode301`
- **Exclusive Access to a Node:** `#SBATCH --exclusive`
 - If your job can utilize all of the resources on a single node, you can specify it

Nautilus Architecture

#Computing Nodes	Processor and Speed	RAM	#Cores
40 cnode[301-340]	BullSequana X440 (2 AMD EPYC 9474@3.6GHz 48c)	384 GB	3840
8 cnode[701-708]	BullSequana X440 (2 AMD EPYC 9474@3.6GHz 48c)	768 GB	768
4 visu[1-4]	BullSequana X450 (2 AMD EPYC 9474@3.6GHz 48c) with Nvidia A40 (48G) 2 GPUs per node	768 GB	384
4 gnode[1-4]	4 BullSequana X410 (2 AMD EPYC 9474@3.6GHz 48c) with Nvidia A100 (80G) 4 GPUs per node	768 GB	384

Note: Other than Nautilus, we have Waves and (Philius)MesoNET cluster as well.

Task Configurations



- In the context of computing, a "**job**" and a "**task**" refer to different entities and have distinct meanings
- **Job:**
 - A job is a higher-level unit of work or a computational task that you submit to a cluster
 - It represents a specific computational workload, which can consist of one or more tasks
 - When you submit a job, you provide information about the resources it needs, such as the number of nodes, CPU cores, memory, runtime, etc.
- **Task:**
 - A task is a lower-level unit of work that is part of a job
 - It represents a specific computational operation or process
 - These tasks are typically parallelized to take advantage of the cluster's computing power

For example, if you have a job that needs to perform a large-scale simulation, you might divide the simulation into multiple tasks, each of which can be run on a separate compute node or cores to expedite the computation. Tasks within a job can be parallel or distributed, and they often communicate with each other to complete the overall workload.

Task Configurations

- **Number of Tasks:** `#SBATCH --ntasks=<num_tasks>`
 - By default, Slurm will assign one task per node
 - These tasks can run on the same node or the different nodes
 - Example: `#SBATCH --ntasks=2`
- **Number of Tasks per Node:** `#SBATCH --ntasks-per-node=<num_tasks>`
 - If your job is using multiple nodes, you can specify the number of tasks per node
 - Example: `#SBATCH --ntasks-per-node=2`
 - For instance, if your job is allocated four compute nodes, each node will run two tasks, resulting in a total of eight tasks running in parallel
 - This option is used when you want to control how many tasks are executed on each individual node in your cluster

CPU and GPU Configurations

- **CPUs per Tasks:** `#SBATCH --cpus-per-task=<num_cpus>`
 - By default, Slurm will assign 1 CPU per task if you do not specify in the configuration
 - Slurm needs to know how many CPUs your job needs
 - Example: `#SBATCH --cpus-per-task=4`
- **GPUs per Job:** `#SBATCH --gres=gpu:<gpu_num>`
 - By default, **Slurm will not assign any GPU** to your job
 - You need to specify how many GPUs your job needs
 - Example: `#SBATCH --gres=gpu:4`

Memory Configurations

- **Memory per Node:** `#SBATCH --mem=<memory>`
 - You need to tell Slurm how much memory you need per node
 - Example: To get 10 GB of memory per node, use `#SBATCH --mem=10g`
 - Default is megabytes(MB), so if you specify `#SBATCH --mem=10`, you will be assigned only 10 MB
- **Memory per CPU:** `#SBATCH --mem-per-cpu=<memory>`
 - You can also specify a memory required per CPU core
 - Example: To get 10 GB of memory per CPU, use `#SBATCH --mem-per-cpu=10g`
 - You need to make sure `--mem` and `--mem-per-cpu` don't conflict with each other
 - Default value is 4 GB for cnode301 to cnode340
 - 8 GB for cnode701 to cnode708

Job Scheduling



- When you submit your job, Slurm checks **#SBATCH** configurations and finds a time/place to run your job
- Four things that impact when you run your job
 - The resources you request
 - The frequency that you submit jobs
 - The other jobs in the queue
 - The maintenance windows (sometimes)
- Note
 - If you request a lot of resources, you'll have to wait until those resources are available
 - If you submit a lot of jobs with a small amount of resources, they'll likely execute quickly

Job Speed



- **Using GPUs may or may not result in a speedup for your job**
- There are a lot of factors in play when it comes to GPUs
 - Your code needs to be able to use GPUs
 - Not all libraries can leverage GPUs, make sure you read the documentation of libraries/frameworks
 - If using multiple GPUs, make sure your code can use GPUs on different nodes
 - Some code can leverage GPUs, but not in an impactful way
 - Some code just isn't doing enough computations to make it worth the overhead of communicating between CPUs and GPUs, it may actually slow down your job

Example Slurm Script

```
#!/bin/bash

#SBATCH --job-name=myjob           # Name for your job
#SBATCH --comment="Run My Job"    # Comment for your job
#SBATCH --output=%x_%j.out        # Output file
#SBATCH --error=%x_%j.err         # Error file

#SBATCH --time=0-00:05:00         # Time limit
#SBATCH --nodes=1                 # How many nodes to run on
#SBATCH --ntasks=2               # How many tasks per node
#SBATCH --cpus-per-task=2         # Number of CPUs per task
#SBATCH --mem-per-cpu=10g         # Memory per CPU
#SBATCH --qos=short               # priority/quality of service

hostname                          # Run the command hostname
```

So, in this example, we have requested a job with the following dimensions:

- **Max Run Time:** 5 Minutes
- **Number of Nodes:** 1
- **Number of Tasks Per Node:** 2
- **Number of CPUs Per Task:** 2
- **Memory Per CPU:** 10GB

- **Submitting Your Job**

```
$ sbatch my-job.slurm
$ sbatch -M nautilus --reservation=training my-job.slurm
Submitted batch job 1411747 on cluster nautilus
```


Monitoring Your Job



- Monitoring Your Job

```
$ ls  
myjob_1411747.err  myjob_1411747.out  my-job.slurm
```

```
$ scontrol show job 1411747 -M nautilus
```

```
$ scancel 1411747
```

Monitoring Your Job

- Monitoring Your Job

```
$ squeue -u $USER
```

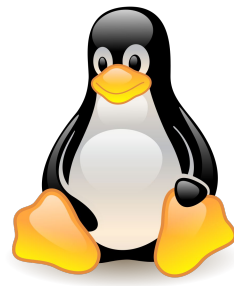
```
CLUSTER: nautilus
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	QOS	PRIORITY	NODELIST(REASON)
-------	-----------	------	------	----	------	-------	-----	----------	------------------

```
CLUSTER: waves
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	QOS	PRIORITY	NODELIST(REASON)
-------	-----------	------	------	----	------	-------	-----	----------	------------------

Data Management - Compress/Decompress Large Files



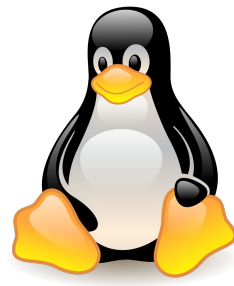
- Compress

```
$ tar -czvf <folder_name.tar.gz> <foldername>
```

- Decompress

```
$ tar -xzvf <folder_name.tar.gz>
```

Data Management: File Transfer



- Local to Remote

```
$ scp -r folder_name nautilus:/scratch/users/username
```

- Remote to Local

```
$ scp -r nautilus:/scratch/users/username/folder_name /local_location
```

Note: Run both commands from the local machine.

Hands-on: TP 1

- Submit your first job!
 - Download “**nautilus-tutorial**” → <https://indico.mathrice.fr/event/498/manage/attachments/>
 - Compress and transfer this folder to this location using SCP → **/scratch/nautilus/users/username**
 - Connect to cluster
 - Open a text editor and write a slurm script that will run the “**hostname**” command
 - Submit the job
 - Monitor your job



Monitoring Your Job

- **squeue**
 - The squeue command will show what jobs are currently scheduled

```
$ squeue
CLUSTER: nautilus
  JOBID PARTITION          NAME          USER ST          TIME  NODES QOS          PRIORITY
NODELIST(REASON)
  1443980      all      Exchange_Second nassaad2017@ PD          0:00      1 short      37307 (Dependency)
1443979_[]      all      Advection        nassaad2017@ PD          0:00      1 short      37307 (Dependency)
  1443978      all      Exchange_First  nassaad2017@ R           0:05      1 short      37307 cnode321
  1439197      all      edw_wave        adermatis202 R          25:12      1 medium     37026 cnode324
  1439000      all      edw_wave        adermatis202 R          26:02      1 medium     37026 cnode321
  1430806      all      edw_wave        adermatis202 R          1:09:23    1 medium     37026 cnode323
  1440954      all      SnappyMesh      ahernandez20 R          16:05      1 medium     36825 cnode325
  1441150      all diff_284_29_Tdiv580_moreRefin_ sakkari2022@ R          15:15      1 medium     36624 cnode325
  1406284      all      diff_300_46_Tdiv600_check sakkari2022@ R          3:24:53    1 medium     36624 cnode324
  1349601      all diff_284_29_Tdiv580_moreRefin_ sakkari2022@ R          18:39:15    1 medium     36624 cnode322
  1308984      all      diff_400_46_Tdiv800_check sakkari2022@ R          1-03:07:33 1 medium     36624 cnode321
  1405888      all      train_model     melaarabi202 R          3:26:58    1 long       27508 gnode1
  1404124      standard test_stability  jlopez@ec-na R          4:00:14    1 long       23725 cnode323
```

```
CLUSTER: waves
  JOBID PARTITION          NAME          USER ST          TIME  NODES QOS          PRIORITY
NODELIST(REASON)
```

Monitoring Your Job

- The `squeue` command gives us the following information:
 - JOBID: The unique ID for your job
 - PARTITION: The partition your job is running on (or scheduled to run on)
 - NAME: The name of your job
 - USER: The username for whomever submitted the job
 - ST: The status of the job. The typical status codes you may see are:
 - **CD** (Completed): Job completed successfully
 - **CG** (Completing): Job is finishing, Slurm is cleaning up
 - **PD** (Pending): Job is scheduled, but the requested resources aren't available yet
 - **R** (Running): Job is actively running
 - TIME: How long your job has been running
 - NODES: How many nodes your job is using
 - QOS: Quality of Service
 - PRIORITY: Priority of your job
 - NODELIST(REASON): List of nodes and which nodes your job is running on (or scheduled to run on). If your job is not running yet, you will also see reason

Partition Information

- `sinfo`
 - available partitions on the cluster and partitions time limit
 - how many nodes are available on the partition and what is the state of those nodes

```
$ sinfo
CLUSTER: nautilus
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
standard    up    infinite    5     mix  cnode[321-325]
standard    up    infinite   35    idle  cnode[301-320,326-340]
bigmem      up    infinite    1    down* cnode707
bigmem      up    infinite    7     idle  cnode[701-706,708]
gpu         up    infinite    1     mix  gnode1
gpu         up    infinite    3     idle  gnode[2-4]
visu        up    infinite    4     idle  visu[1-4]
all*        up    infinite    1    down* cnode707
all*        up    infinite    6     mix  cnode[321-325],gnode1
all*        up    infinite   49    idle  cnode[301-320,326-340,701-706,708],gnode[2-4],visu[1-4]

CLUSTER: waves
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*        up    9:00:00    1    unk*  budbud018
all*        up    9:00:00    8     idle  budbud[014-017,019-022]
med         up  4-04:00:00  3     idle  budbud[020-022]
devel       up    20:00      1    unk*  vmworker-001
```

- Try `sinfo -N`

Track Your Jobs

- **sacct**
 - Track your recent jobs to find their job IDs and other details

```
$ sacct
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
1404103	myjob	standard	glicid	1	COMPLETED	0:0
1404103.bat+	batch		glicid	1	COMPLETED	0:0
1404103.ext+	extern		glicid	1	COMPLETED	0:0
1419267	myjob	all	glicid	4	COMPLETED	0:0
1419267.bat+	batch		glicid	4	COMPLETED	0:0
1419267.ext+	extern		glicid	4	COMPLETED	0:0

- To view a specific job

```
$ sacct --jobs=1411747
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
1419267	myjob	all	glicid	4	COMPLETED	0:0
1419267.bat+	batch		glicid	4	COMPLETED	0:0
1419267.ext+	extern		glicid	4	COMPLETED	0:0

Check Job State



- **scontrol**
 - To check job state, start time/end time, command, workdir, stderr, stdout

```
$ scontrol show job 1411747 -M nautilus
JobId=1446614 JobName=myjob
UserId=jmir@ec-nantes.fr(8000019) GroupId=jmir@ec-nantes.fr(8000019) MCS_label=N/A
Priority=45942 Nice=0 Account=glicid QOS=short
JobState=COMPLETED Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
Runtime=00:00:00 TimeLimit=00:05:00 TimeMin=N/A
SubmitTime=2023-10-17T14:40:47 EligibleTime=2023-10-17T14:40:47
AccrueTime=2023-10-17T14:40:47
StartTime=2023-10-17T14:40:47 EndTime=2023-10-17T14:40:47 Deadline=N/A
SuspendTime=None SecsPreSuspend=0 LastSchedEval=2023-10-17T14:40:47 Scheduler=Backfill
Partition=all AllocNode:Sid=nautilus-devel-001:883696
ReqNodeList=(null) ExcNodeList=(null)
NodeList=cnode321
BatchHost=cnode321
NumNodes=1 NumCPUs=4 NumTasks=2 CPUs/Task=2 ReqB:S:C:T=0:*.:*
TRES=cpu=4,node=1,billing=4
Socks/Node=* NtasksPerN:B:S:C=0:0:*.:* CoreSpec=*
MinCPUsNode=2 MinMemoryCPU=10G MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/scratch/users/jmir@ec-nantes.fr/nautilus-tutorial/task_1/my-job.slurm
WorkDir=/scratch/users/jmir@ec-nantes.fr/nautilus-tutorial/task_1
Comment=Run My Job
StdErr=/scratch/users/jmir@ec-nantes.fr/nautilus-tutorial/task_1/myjob_1446614.err
StdIn=/dev/null
StdOut=/scratch/users/jmir@ec-nantes.fr/nautilus-tutorial/task_1/myjob_1446614.out
Power=
```

Job Priority Factors

- Slurm takes into account two different factors when scheduling jobs:
 - Requested Resources and Priority
 - If you request a lot of resources, your job may take longer to start than someone who requests very few resources because Slurm needs to wait for the resources you requested to be available
 - If you are constantly submitting and running jobs, Slurm may assign your jobs a lower priority than someone who rarely submits jobs.

```
Job_priority =  
site_factor +  
(PriorityWeightAge) * (age_factor) +  
(PriorityWeightAssoc) * (assoc_factor) +  
(PriorityWeightFairshare) * (fair-share_factor) +  
(PriorityWeightJobSize) * (job_size_factor) +  
(PriorityWeightPartition) * (priority_job_factor) +  
(PriorityWeightQOS) * (QOS_factor) +  
SUM(TRES_weight_cpu * TRES_factor_cpu,  
    TRES_weight_<type> * TRES_factor_<type>,  
    ...) - nice_factor
```

Job Priority Factors



- Nine factors in the Multifactor Job Priority plugin that influence job priority:
 - **Age:** the length of time a job has been waiting in the queue, eligible to be scheduled
 - **Association:** a factor associated with each association
 - **Fair-share:** the difference between the portion of the computing resource that has been promised and the amount of resources that has been consumed
 - **Job size:** the number of nodes or CPUs a job is allocated
 - **Nice:** a factor that can be controlled by users to prioritize their own jobs
 - **Partition:** a factor associated with each node partition
 - **QOS:** a factor associated with each Quality Of Service
 - **Site:** a factor dictated by an administrator or a site-developed job_submit or site_factor plugin
 - **TRES:** each TRES Type has its own factor for a job which represents the number of requested/allocated TRES Type in a given partition

Priorities in GLiCID Cluster

- `sacctmgr`
 - To view or modify Slurm account information

```
$ sacctmgr show qos format="name%20,priority,MaxJobsPerUser,MaxWall"
```

Name	Priority	MaxJobsPU	MaxWall
normal	1		00:05:00
short	50		1-00:00:00
medium	40		3-00:00:00
long	30		8-00:00:00
unlimited	10	1	
debug	100		00:20:00
priority	200		8-00:00:00

Different Resource Scenarios

Here is a summary of different resource utilization scenarios:

- **RAM:**
 - Request too little: Job will die when it runs out of RAM
 - Request too much: Lots of RAM will sit idle and no one else can use it
 - *Ideal:* Request slightly more RAM than you need
 - Recommendation: Try to keep idle RAM at less than 10% of the total RAM you requested
- **CPUs:**
 - Request too little: Your job will trip over itself because of kernel scheduling; your job will take a massive performance hit as a result
 - Request too much: Lots of CPUs will sit idle and no one else can use them
 - *Ideal:* Request exactly the number of CPUs that your job can use

Different Resource Scenarios

- **GPUs:**
 - Request too little: You may not actually see a speedup (due to communication overhead between CPUs and GPUs)
 - Request too much: Your code may not be able to use multiple GPUs; idle GPUs cannot be used by anyone else until your job finishes
 - *Ideal:* Request exactly the number of GPUs that your job can use
 - Recommendation: Get your job working with one GPU, and make sure you're actually using the GPU before trying to use more
- **Time:**
 - Request too little: Your job will not finish before the time limit runs out; lots of time will be wasted
 - Request too much: Slurm may give your job a lower priority to let smaller jobs go first. If a maintenance window is coming up, your job may not schedule until after the maintenance window
 - *Ideal:* Request slightly more time than you need, but not too much

Parallel Programming Examples using Slurm



Parallel programming on a cluster can be challenging, but it is a powerful technique for harnessing the computational resources of a cluster effectively.

- Some reasons why parallel programming can be tricky on a cluster:
 - Distributed computing, load balancing, synchronization, communication overhead, debugging and troubleshooting, scalability, heterogeneous resources
- To overcome these challenges, developers often use parallel programming libraries,
 - such as MPI (Message Passing Interface) for distributed memory systems and
 - OpenMP for shared memory systems
- These libraries provide abstractions and tools for handling parallelism, communication, and synchronization
- Additionally, understanding the architecture of the cluster and the specifics of the job scheduler (e.g., Slurm) can be crucial for resource allocation and job management

Example 1: Intel/IntelMPI



Sample Script: `job-intel.slurm`

```
#!/bin/bash
#SBATCH --job-name=HelloWorldMpi
#SBATCH --partition=standard
#SBATCH --ntasks=4

module purge
module load intel/compiler intel/mpi

export I_MPI_PMI_LIBRARY=/lib64/libpmi2.so
export I_MPI_COLL_EXTERNAL=0
export I_MPI_ADJUST_BCAST=0
export I_MPI_FABRICS=shm:ofi
export FI_PROVIDER=psm3

srun --mpi=pmi2 hello-mpi
```

Example 1: Intel/IntelMPI

- Intel Compiler and IntelMPI

```
$ module load intel/compiler intel/mpi  
$ mpicxx -cxx=icpx -O3 -o hello-mpi hello-mpi.cpp
```

- Submit your slurm script

```
$ sbatch -M nautilus -p standard -q short job-intel.slurm
```

Example 2: GNU/OpenMPI



Sample script: `job-mpi.slurm`

```
#!/bin/bash
#SBATCH --job-name=HelloWorldMpi
#SBATCH --partition=standard
#SBATCH --ntasks=4

module purge
module load gcc openmpi/ucx/4.1.5_gcc_8.5.0_ucx_1.14.1_rdma_46.0

export UCX_WARN_UNUSED_ENV_VARS=n
export OMPI_MCA_btl=^openib
export UCX_NET_DEVICES=mlx5_2:1

srun ./hello-openmpi
```

Example 2: GNU/OpenMPI

- GNU Compiler and OpenMPI
 - `module load gcc openmpi/ucx/4.1.5_gcc_8.5.0_ucx_1.14.1_rdma_46.0`
 - `mpicxx -O3 -o hello-openmpi hello-mpi.cpp`
- Submit your slurm script

```
$ sbatch -M nautilus -p standard -q short job-mpi.slurm
```

Example 3: GNU/OpenMP

Sample script → `job-omp.slurm`

```
#!/bin/bash
#SBATCH --job-name=HelloWorldOmp
#SBATCH --partition=standard
#SBATCH --cpus-per-task=12

module purge
module load gcc

if [[ "${SLURM_CPUS_PER_TASK}" ]]
then
  c=${SLURM_CPUS_PER_TASK}
else
  c=1
fi

export OMP_NUM_THREADS=$c
srun ./hello-omp
```

Example 3: GNU/OpenMP

- GNU Compiler and OpenMP
 - `module load gcc`
 - `g++ -fopenmp -o hello-omp hello_omp.cpp`
- Submit your slurm script

```
$ sbatch -M nautilus -p standard -q short job-omp.slurm
```

Example 4: GNU/hybrid OpenMPI/OpenMP

Sample script: `job-hybrid.slurm`

```
#!/bin/bash
#SBATCH --job-name=HelloWorldHybrid
#SBATCH --partition=standard
#SBATCH --cpus-per-task=6
#SBATCH --ntasks=16

module purge
module load gcc openmpi/ucx/4.1.5_gcc_8.5.0_ucx_1.14.1_rdma_46.0

export UCX_WARN_UNUSED_ENV_VARS=n
export OMPI_MCA_btl=openib
export UCX_NET_DEVICES=mlx5_2:1

if [[ "${SLURM_CPUS_PER_TASK}" ]]
then
    c=${SLURM_CPUS_PER_TASK}
else
    c=1
fi

export OMP_NUM_THREADS=$c
srun ./hello-hybrid
```

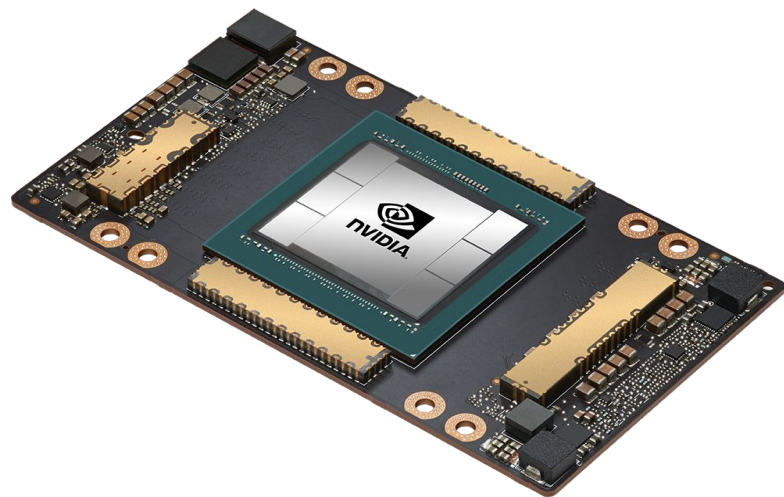
Example 4: GNU/OpenMP

- GNU Compiler and OpenMP
 - `module load gcc openmpi/ucx/4.1.5_gcc_8.5.0_ucx_1.14.1_rdma_46.0`
 - `mpicxx -fopenmp -o hello-hybrid hello-mpi-omp.cpp`
- Submit your slurm script

```
$ sbatch -M nautilus -p standard -q short job-hybrid.slurm
```


Test GPU: Vector Multiplication of 2 billion elements

- Test Example Demo (commands for this particular session only as we have reserved GPU)
- Load Module
 - `module load nvhpc/23.9`
- To compile
 - `nvc++ -o dp_acc -O3 -acc dp_acc.cpp`
- To submit job
 - `sbatch -M nautilus --reservation=training run_acc.slurm`
- To monitor
 - `ssh gnode1`
 - `nvidia-smi -l 1`



Hands-on: TP 2

- Create a Slurm script for any of the above 4 examples
 - Submit your job
 - Monitor your job





Micromamba/Anaconda

- No Anaconda module for now
- But you can use Micromamba - lighter version of conda

```
# Download micromamba
mkdir -p $HOME/.local/bin
wget -P $HOME/.local/bin https://s3.glicid.fr/pkgs/micromamba
chmod u+x $HOME/.local/bin/micromamba
```

```
# Initialize micromamba
$HOME/.local/bin/micromamba -r /micromamba/$USER/ shell init --shell=bash
--prefix=/micromamba/$USER/
```

```
# [OPTIONAL] Add a `conda` alias
echo -e '\n\n#Alias conda with micromamba\nalias conda=micromamba' >> ~/.bashrc
```

```
# Recharger le .bashrc
source ~/.bashrc
```

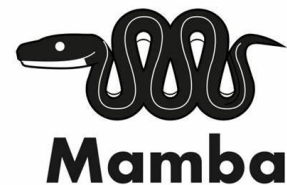
- Source: <https://doc.glicid.fr/GLiCID-PUBLIC/0/logiciels/logiciels.html>

Micromamba/Anaconda

 PyTorch

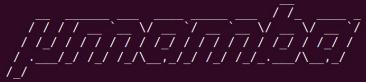


TensorFlow



```
$ micromamba --version
$ conda create --name myenv
$ conda env list
$ conda activate myenv
$ conda install numpy
$ conda list
$ conda deactivate
```

```
jmir@ec-nantes.fr@nautilus-devel-001/scratch/users/jmir@ec-nantes.fr/nautilus-tutorial
[jmir@ec-nantes.fr@nautilus-devel-001 nautilus-tutorial]$ micromamba --version
1.4.0
[jmir@ec-nantes.fr@nautilus-devel-001 nautilus-tutorial]$ conda env list
```



```
Name      Active Path
-----
base      /home/jmir@ec-nantes.fr/.conda/envs/pytorch
pytorch   /micromamba/jmir@ec-nantes.fr

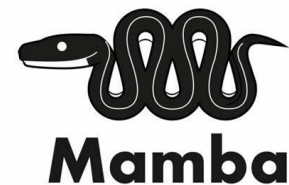
[jmir@ec-nantes.fr@nautilus-devel-001 nautilus-tutorial]$ conda activate pytorch
(pytorch) [jmir@ec-nantes.fr@nautilus-devel-001 nautilus-tutorial]$
```

Hands-on: TP 3

PyTorch



TensorFlow



- Install Micromamba and check version
- Create a Conda environment and check environment list
- Try to install numpy and check installed packages



Hands-on: TP4_Fortran

- Load gcc compiler and compile

```
$ module load gcc/13.1.0  
$ gfortran hello-fortran.f90 -o hello
```

- Submit your slurm script

```
$ sbatch -M nautilus -p standard -q short my-job.slurm
```



Thank you. Any questions?



Please answer the survey if you haven't yet
<https://forms.gle/B4dto4axGm4EVPwaA>

Useful links:

User Doc: <https://doc.glicid.fr>

Support: <https://help.glicid.fr> or help@glicid.fr

Chat: On CLAM website

Admins: tech@glicid.fr

Forum: Coming soon

Status page: <https://ckc.glicid.fr>