SLURM Best Practice Guide for HPC Users

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

Content

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

1.- INTRODUCTION[1]

Slurm (Simple Linux Utility for Resource Management) is one of the most widely used task management systems in computer clusters.

It has three main functions:

1. **Resource allocation**: allocates users access to resources (compute nodes) to users for a given period of time so that they can perform their work.

2. **Work initiation and monitoring:** provides a framework for initiating, executing and monitoring work on the set of allocated nodes.

3. **Resource arbitration**: arbitrarily allocates resources by managing a queue of pending work.

Slurm is an open source, fault-tolerant job scheduling and cluster management framework that requires no kernel modifications to operate. It is widely used in supercomputers and clusters, and more specifically it is used by some of **NCC Spain's** partners.

Therefore, at **NCC Spain**, we consider it of great interest for users requesting access to our **HPC** resources, either through the **EuroCC Spain Testbed** or other means, to know and know how to use it, and to have at their disposal a series of good practices and/or tips that can facilitate its use. For this reason, a series of recommendations and commands that may be useful have been compiled in this document

Slurm is written in C language and uses a GNU **autoconf** configuration engine. Although initially written for Linux, other UNIX-like operating systems should be easy targets for porting. The code should adhere to the Linux kernel coding style.

Architecture

Slurm consists of a **slurmd** daemon running on each compute node and a central **slurmctld** daemon running on a management node (with optional failover twin).

The **slurmd** daemons provide hierarchical error-tolerant communications.

The entities managed by these Slurm daemons include **nodes**, the Slurm computing resource, **partitions,** which group nodes into (possibly overlapping) logical sets, **jobs** or resource allocations assigned to a user for a specified amount of time, and job steps, which are sets of (possibly parallel) tasks within a job. Partitions can be considered as queues of jobs, each of which has a variety of constraints such as job size limit, job time limit, users authorised to use it, etc. Priority-ordered jobs are assigned nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted. Once a job is assigned a set of nodes, the user can start a parallel job in the form of job steps in any configuration within the assignment. For

[1] Source: Slurm Workload Manager - Descripción general (schedmd.com)

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

example, a single job step can be initiated that uses all nodes assigned to the job, or multiple job steps can independently use a portion of the assignment.

2.- BASIC CONCEPTS [i]

The most important concepts within slurm are:

- Computation nodes.
- Partitions.
- Jobs.
- Tasks that represent a process within a job.

a) Computation nodes

It is from the login nodes that the user interacts with Slurm and from where he launches and monitors his jobs. From here, the user accesses his data and the results of the executions.

b) Partitions

Partitions are a mechanism for logically grouping nodes, each of which can have a variety of constraints such as job size limit, job time limit, users authorised to use it, etc. As jobs are launched in a particular partition, they are ordered by priority within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted and can be left waiting.

c) Most common SLURM commands

To familiarise yourself with the task manager, the most common slurm commands are listed. To learn more about the command as options you can run:

man <command>
<command> --help

- **sbatch** <script file>: launch a script to the queue manager.

  Used to submit a job script for further execution. Typically, the script will contain one or more execution commands to initiate parallel tasks.

- **squeue:** to check the status of work in the queues.

  Reports the status of jobs. It has a wide variety of filtering, sorting and formatting options. By default, it reports running jobs in order of priority and then pending jobs.

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

- **scancel** <job_id list>: to cancel a job.

  It is used to cancel a or execute the job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.

- **scontrol show job** <job_id>: to obtain information on the status of a job.

  This is the administrative tool used to view and/or modify the status of Slurm.

- **sinfo**: to view the status of the system queues.

  Reports the status of partitions and nodes managed by Slurm. Has a wide variety of filtering, sorting and formatting options.

- **salloc** <opcions>: to log in interactively (get a node to run).

  Used to allocate resources to a real-time job. Typically, this is used to allocate resources and generate a shell, i.e. run an interactive session.

- **srun** <aplicacion>: to submit a job for execution or initiate job steps in real time.

  SRUN has a wide variety of options for specifying resource requirements, including: minimum and maximum number of nodes, number of processors, specific nodes to be used or not, and specific node characteristics (memory, disk space, features, etc.). A job may contain several job steps that are executed sequentially or in parallel on independent or shared resources within the job's node allocation.

- **sacct**: to consult the accounting of one's own account.

  It is used to report information on active or completed jobs.

- **sstat:** obtain information on the resources used by a work in progress.

  It is used to obtain information about the resources used by a running job or a work step.

- **Sview:** is a graphical user interface for obtaining and updating status information of jobs, partitions and nodes managed by Slurm.

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

3.- USE OF USEFUL COMMANDS IN SLURM [2]

Slurm provides a variety of tools that allow the user to manage and understand their jobs. This section will introduce these tools and provide details on how to use them.

a) Finding information in the work queue with squeue

The squeue command is a tool used to obtain information about the jobs in the queue. By default, the squeue command will print the job ID, partition, job name, job user, job status, job running time, number of nodes, and list of assigned nodes:

```
squeue
JOBID  PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
111111     batch   my_job   myuser  R    1:21:59      1 node0101-1
```

We can generate unabbreviated information with the --long flag. This flag will print the default unabbreviated information with the addition of a time limit field.

```
squeue -l
squeue --long
```

The squeue command also gives users a means to calculate the estimated start time of a job by adding the --start flag to our command. This will add Slurm's estimated start time for each job to our output data.

```
squeue --user=username --start
```

Note

The start time provided by this command may be inaccurate. This is because the calculated time is based on the jobs queued or running on the system. If a job with a higher priority is queued after executing the command, your job may be delayed.

When checking the status of a job, you may want to repeatedly call the squeue command to check for updates. We can accomplish this by adding the --iterate flag to our squeue command. This will execute squeue every n seconds, allowing for frequent and continuous updating of queue information without the need to repeatedly call squeue:

```
squeue --start --iterate=n_seconds
```

[2] Source: https://doc.hpc.iter.es/2023.03/slurm/how_to_slurm_useful_commands/

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

Press ctrl-c to stop the command loop and return to the terminal.

b) Status of work

Once a job has been submitted to a job queue, the execution will follow these states:

- PENDING or PD: the job has entered the queue but the requested resources are not yet available for it to start working, i.e. there are no free nodes.
- RUNNING or R: the work is running in the queue with the resources that have been requested.
- COMPLETED or CD: the job has been executed correctly, or at least what has been specified in the launch script.
- COMPLETING or CG: the work is in the process of being completed in a good state.
- SUSPEND or S: implementation of the work has been suspended and the resources used have been released for other work.
- CANCELLED or CA: the job has been cancelled either by the user or by the system administrators.
- FAILED or F: the execution of the work has failed.
- NODE_FAIL or NF: an error occurred with the node and the job could not be launched. By default, Slurm relaunches the job again.

c) Reasons for a job being in PENDING

When a job is in the PENDING status, the reason why it is pending to execute is added and can be:

- (**Resources):** the job is waiting until the requested resources are available.
- (**Dependency):** the job is dependent on another job and, therefore, it will not start executing until the condition established for the dependency is fulfilled.
- (**DependencyNeverSatisfied):** The job is waiting for a dependency that has not been fulfilled. The job will stay in this state forever, therefore, the job must be cancelled.
- **(AssocGrpCpuLimit):** The job cannot be executed because the allocated CPU quota has been consumed.

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

- (**AssocGrpJobsLimit):** The job cannot be executed because the limit of concurrent jobs that the user or account is allowed to execute has been reached.
- **(ReqNodeNotAvail):** The specified node is not available. It may be in use, it may be reserved, or it may be marked as "out of service".

Info

For more information on squeue, visit the Slurm page on squeue.

d) Stopping work with scancel

Occasionally, you may need to stop a job completely while it is running. The best way to accomplish this is with the **scancel** command. This command allows you to cancel jobs that are running on Research Computing resources using the job ID. The command looks like this:

```
scancel your_job-id
```

To cancel multiple jobs, you can use a comma-separated list of job IDs:

```
scancel your_job-id1, your_job-id2, your_jobiid3
```

Info

For more information on squeue, visit the Slurm page on squeue.

e) Status information with sstat

The sstat command allows users to easily obtain information about the status of their currently running jobs. This includes CPU usage information, task information, node information, resident set size (RSS) and virtual memory (VM). We can invoke the **sstat** command as such:

```
sstat --jobs=your_job-id
```

f) Formatting the sstat output

By default, sstat will extract much more information than would be needed in the default **output** of commands. To remedy this, we can use the --format flag to choose what we want in our output. The format flag takes a comma-separated list of variables that specify the output data:

```
sstat --jobs=your_job-id --format=var_1,var_2, ... , var_N
```

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

Some of these variables may include:

Variable	Description
avecpu	Average CPU time of all tasks in the job.
averss	Average size of the resident set of all tasks.
avevmsize	Average virtual memory of all tasks in a job.
jobid	Job ID.
maxrss	Maximum number of bytes read by all tasks in the job.
maxvsize	Maximum number of bytes written by all tasks in the job.
ntasks	Number of tasks in a job.

For example, print the average job ID of a job, the CPU time, the maximum rss and the number of tasks. We can do this by typing the command:

```
sstat --jobs=your_job-id --format=jobid,cputime,maxrss,ntasks
```

Info

A complete list of variables specifying data handled by sstat can be found with the --helpformat flag or by visiting the slurm page at sstat.

g) Analysing finished works with sacct

The **sacct** command allows users to obtain information about the status of completed jobs. This command is very similar to sstat, but is used on jobs that were previously run on the system instead of jobs that are currently running. We can use the job ID.

- **For all work carried out:**

```
sacct
```

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

- **For a single job, identified by its ID:**

```
sacct --jobs=your_job_id
```

By default, sacct will only pull jobs that have run **on the current day**. We can use the --starttime flag to tell the command to look beyond its cache of short-term jobs.

```
sacct –-jobs=your_job-id –-starttime=YYYY-MM-DD
```

To see an unabbreviated version of the sacct output, use the --long flag:

```
sacct –-jobs=your_job-id –-starttime=YYYY-MM-DD --long
```

h) Formatting the sacct output

Like sstat, the standard output may not provide the information we want. To remedy this, we can use the --format flag to choose what we want in our output. Similarly, the format flag is driven by a comma-separated list of variables that specify the output data:

```
sacct --user=your_rc-username --format=var_1,var_2, ... ,var_N
```

A list of some variables is provided below:

Variable	Description
account	Account on which the work was executed.
avecpu	Average CPU time of all tasks in the job.
averss	Average resident set size of all tasks in the job.
cputime	Elapsed CPU time used by a job or step.
elapsed	Elapsed time of jobs with format DD-HH:MM:SS
exitcode	The output code returned by the job script or salloc.

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

Variable	Description
jobid	Job ID.
jobname	Name of the work.
maxdiskread	Maximum number of bytes read by all tasks.
maxdiskwrite	Maximum number of bytes writte by all tasks
maxrss	The output code returned by the job script or salloc.
ncpus	Number of allocated CPUs.
nnodes	Number of nodes used.
ntasks	Number of tasks in a job.
priority	Slurm priority.
qos	Quality of service.
reqcpu	Number of CPUs ordered
reqmem	Amount of memory required for a job
user	User name of the person who carried out the work.

For example, suppose you want to find information about jobs that ran on 12 March 2018. You want to display information about the job name, the number of nodes used in the job, the amount of CPU, **the maxrss**, and the elapsed time. Your command would look like this:

```
sacct --starttime=2018-03-12 --format=jobname,nnodes,ncpus,maxrss,elapsed
```

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

Info:

A complete list of variables specifying data handled by sacct can be found with the --helpformat flag or by visiting the slurm page at sacct.

i) Control of queued and running jobs through scontrol

The **scontrol** command gives users greater control over their jobs running through Slurm. This includes actions such as suspending a job, stopping the execution of a job, or extracting detailed information about the status of jobs.

To suspend a job that is currently running on the system, we can use **scontrol** with the suspend command. This will stop a running job at its current step which can be resumed at a later time. We can suspend a job by typing the command:

```
scontrol suspend job_id
```

To resume a paused job, we use scontrol with the resume command.

```
scontrol resume job_id
```

Slurm also provides a utility to hold jobs that are queued in the system. Holding a job will place the job at the lowest priority, effectively "holding" the job from running. A job can only be held if it is waiting for the system to run. We use the hold command to place a job in a waiting state:

```
scontrol hold job_id
```

We can then release a held job using the **release** command:

```
scontrol release job_id
```

scontrol can also provide job information via the show job command. The information provided by this command is quite extensive and detailed, so be sure to clear your terminal window, collect some information from the command, or pipe the output to a separate text file:

```
scontrol show job job_id
```

j) Streaming output to a text file

```
scontrol show job job_id > outputfile.txt
```

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

k) Channel the output to Grep and find lines containing the word "Time".

```
scontrol show job job_id | grep Time
```

4.- TIPS3

Tip: Using modules in a script

In order to be able to use the modules environment, it is necessary to source the modules profile file. To do this, the following line must be added to the submit scripts before using any module command:

```
source /etc/profile.d/profile.modules.sh
```

Tip: Reserve complete compute nodes

On compute nodes, it is recommended to reserve complete nodes using the -N <nodes> option, so that other users' executions do not interfere with each other. Remember that the billing of these nodes is per node usage.

Tip: Submit a job with sbatch

It is advisable to submit the job via sbatch as well as using the modifiers -D <directory> and -t <time>.

-t <time> or its equivalent #SBATCH --t <days-HH:MM>.

-D <directory> or its equivalent #SBATCH --D <directory>.

Tip: Number of tasks per node

It is possible to run without using all the available cores on the node. To do this, just request the number of nodes with -N X and the number of processes to run on each node with --tasks-per-node:srun -N 4 --tasks-per-node=8 <software>

```
#SBATCH --N 4
#SBATCH --tasks-per-node=8
```

[3] Source: Consejos de uso - TeideHPC (iter.es)

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

★ Tip: Notifications from the slurm job manager

It is possible to manage the automatic notification of certain job events with the following policies.

```
#SBATCH --mail-user=EMAIL                # Event notification email
#SBATCH --mail-type=EVENT1, EVENT2,...   # Reportable events
```

Clarification on Slurm's mail events:

Slurm can send mails to the specified address about a number of events that happen to the job. Such events can be:

- **BEGIN**: when the work goes into execution.

- **END**: when the execution of the work is completed.

- **FAIL**: when the execution of the work fails.

- **TIME_LIMIT**: when the work reaches the maximum execution time.

- **TIME_LIMIT_50**: when the work has reached 50% of the time limit.

- **TIME_LIMIT_80**:. when the work has reached 80% of the time limit

- **TIME_LIMIT_90**: when the work has reached 90% of the time limit.

- **ARRAY_TASKS**: sends an email notification for each job in the array. If, when using arrays, this option is not specified, an email will be sent as if it were a single job.

- **ALL**: all types of events.

Of all the possible events, it is recommended to use the ones related to the allowed time limit consumption, TIME_LIMIT_50, TIME_LIMIT_80 and TIME_LIMIT_90. In this way, the user is aware of the time remaining for the job and, if necessary, can send an email to the administrators in time to extend the execution time of the job.

★ Tip: Bash header

We recommend using #!/bin/bash -e instead of simple #!/bin/bash, so that failure of any command within the script causes your job to stop immediately instead of trying to continue with an unexpected environment or erroneous intermediate data. It also ensures that your failed jobs show a FAILED status in the sacct output.

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

★ Tip: Resources

Do not request more resources (CPU, memory, GPU) than you will need. In addition to using your core hours faster, resource-intensive jobs will take longer to queue. Use the information provided at the end of your job (e.g. via the sacct command) to better define your resource requirements.

★ Tip: Wall time

Long jobs will spend more time in the queue, as there are more opportunities for the scheduler to find a time slot to run shorter jobs. Therefore, consider using job checkpoints or, where possible, more parallelism, to reduce the duration of jobs to a few hours or, in the worst case, days.

Leave some margin for safety and variability between runs in the system, but try to be as accurate as possible.

If you have many jobs of less than 5 minutes, then they should probably be combined into larger jobs using a simple loop in the batch script to amortise the overhead of each job (start-up, accounting, etc.).

★ Tip: Memory (RAM)

If you request more memory (RAM) than you need for your job, you will wait longer in the queue and it will be more expensive when it runs. On the other hand, if you do not request enough memory, the job may be cancelled for trying to exceed the allocated memory limits.

We recommend that you request a little more RAM, but not much more, than your program will need at its maximum usage.

We also recommend using --memen instead of --mem-per-cpu in most cases. There are a few types of jobs for which --mem-per-cpu is more suitable

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

Tip: Parallelism

In general, only MPI jobs should set ntasks greater than 1 or use srun. If you don't know if your program supports MPI, it probably does not.

Only multiprocess jobs should set cpus-per-task . If you don't know if your program supports multi-threading, try benchmarking with 2 CPUs and with 4 CPUs and see if there is a two-fold difference in elapsed job time.

Job arrays are an efficient mechanism for managing a collection of batch jobs with identical resource requirements. Most Slurm commands can handle job arrays either as individual items (tasks) or as a single entity (e.g., delete an entire job array in a single command).

5.- BEST PRACTICES FOR LARGE NUMBERS OF JOBS

Consider placing related work in a single Slurm job with multiple jobs, both for performance reasons and for ease of management. Each Slurm job can contain a multitude of job steps and the overhead in Slurm to manage the job steps is much less than that of individual jobs.

Job arrays are an efficient mechanism for managing a collection of batch jobs with identical resource requirements. Most Slurm commands can manage job arrays as individual elements (tasks) or as a single entity (e.g., delete an entire job array in a single command).

MPI

The use of MPI depends on the type of MPI being used. Three fundamentally different modes of operation are used by these various MPI implementations.

1. Slurm directly initiates tasks and performs communication initialisation via the PMI2 or PMIx APIs. (Supported by most modern MPI implementations).

2. Slurm creates a resource allocation for the job, and then mpirun launches tasks using the Slurm infrastructure (older versions of OpenMPI).

3. Slurm creates a resource allocation for the job, and then mpirun launches tasks using some mechanism other than Slurm, such as SSH or RSH. These tasks are started outside of Slurm's supervision or control. Slurm's epilogue should be configured to purge these tasks when the job assignment is relinquished. The use of pam_slurm_adopt is also recommended.

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

OPENMP[ii]

Open MP (Open Multi-Processing) is an application programming interface (API) that enables parallel programming on shared-memory systems. OpenMP has become a standard for parallel programming on shared-memory systems and is compatible with several programming languages, such as C, C++, and Fortran.

OpenMP is characterised by being easy to learn, flexible and portable. The source code of an application can be adapted to different systems without having to modify the original source code. In addition, OpenMP offers a wide variety of directives for threading, synchronisation and job allocation.

OpenMP Directives

OpenMP directives are keywords used in the source code to tell the application which sections of the code should be executed in parallel. OpenMP directives are also used to specify thread creation and synchronisation.

Some of the most commonly used OpenMP directives:

- **parallel:** this directive creates a parallel region where work is divided among multiple threads. Each thread executes a copy of the parallel region and then they are joined at the end.

- **for:** used to parallelise loops. It divides the work of the loop among the available threads, where each thread executes a portion of the loop.

- **sections**: used to parallelise independent sections of code. Each section runs on a separate thread.

- **single:** with the single directive, a section of code runs on only one thread. It can be used for initialisations or to perform one-time operations.

- **task:** Used to create independent tasks that can be executed by available threads. It provides a more flexible execution model than the for and sections directives.

- **critical:** used to define a critical section of code, where only one thread can execute it at a time. It is used to protect sections of code that access shared resources.

- **atomic:** used to perform atomic operations on shared variables. It ensures that the operation is performed without interference from other threads.

- **barrier**: used to synchronise all threads at a specific point in the program. Threads will wait until all other threads reach the same point before continuing.

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

FOR MORE INFORMATION YOU CAN ALSO CONSULT:

QUICK USER GUIDE: https://slurm.schedmd.com/quickstart.html

VIDEO TUTORIALS: https://www.schedmd.com/publications/

[i] Source: Cómo iniciar sesión en TeideHPC - TeideHPC (iter.es)

[ii] Source: https://medium.com/@leonardoaguirre_97179/openmp-y-la-programaci%C3%B3n-paralela-8990f14b95f3

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.