Ejecuciones secuenciales

La unidad de reserva mínima para ejecutar en TeideHPC es el nodo, por lo que, independientemente del trabajo a ejecutar, a la hora de realizar el accounting del mismo, se considerará que se ha hecho uso de nodos completos. Es decir, **el cómputo de horas de uso de la infraestructura viene dado por el número de horas que cada nodo está ejecutando el trabajo en cuestión, se use o no todos los recursos disponibles del mismo.**

Para que el usuario no se vea perjudicado por este hecho, habrá de estructurar sus ejecuciones de manera que pueda agruparlas en igual cantidad de número de cores al de los nodos (16).

Las ejecuciones secuenciales suelen requerir muchas ejecuciones de trabajos a pocos cores, por lo que el usuario deberá estructurar los datos de entrada para las ejecuciones en carpetas o nombres de ficheros identificados mediante un número de trabajo, de manera que éste se pueda manipular de manera sencilla dentro del script de lanzamiento.

Una vez se tengan los datos de entradas organizados de esta manera se puede proceder a la ejecución de la misma, lanzando en un script de submit tantas tareas como quepan en un nodo.

Ejemplo de ejecución sequencial de trabajos de 1 core

El siguiente script de ejemplo lanza 16 trabajos de 1 core simultáneamente.

```
# !/bin/bash

#SBATCH -J <job_name>
#SBATCH -p <partition>
#SBATCH -N 1
#SBATCH --constrains=<node arquitecture>  # sandy, ilk (icelake)... arquitecture
#SBATCH -t <days-HH:MM:SS>
#SBATCH -o <file.out>
#SBATCH -D .
#############################################################



module purge
module load <modules>

for i in {1..16}; do
   srun --ntasks=1 --exclusive --output=slurm-%J.out serial-program --input /path/to/input.$i
```

```
&
done
# wait until all background processes are ended
wait
```

El comando `srun` lanzará los trabajos individuales en diferentes cores. El símbolo `&` al final de la línea significa que el trabajo se ejecute en segundo plano, de manera que no haya que esperar para lanzar la siguiente tarea. El comando `wait` del final evita que el trabajo termine hasta que no hayan terminado todos los procesos anteriores.

Ejemplo ejecución secuencial trabajos de 4 cores

El siguiente script de ejemplo lanza 4 trabajos de 4 cores simultáneamente lo que hace un total de 16 cores utilizados:

```
# !/bin/bash

#SBATCH -J <job_name>
#SBATCH -p <partition>
#SBATCH -N 1
#SBATCH --constrains=<node arquitecture>  # sandy, ilk (icelake)... arquitecture
#SBATCH -t <days-HH:MM:SS>
#SBATCH -o <file.out>
#SBATCH -D .
#############################################################

module purge
module load <modules>

srun --ntasks=4 --exclusive --output=slurm-%J.out serial-program --input /path/to/input_1 &
srun --ntasks=4 --exclusive --output=slurm-%J.out serial-program --input /path/to/input_2 &
srun --ntasks=4 --exclusive --output=slurm-%J.out serial-program --input /path/to/input_3 &
srun --ntasks=4 --exclusive --output=slurm-%J.out serial-program --input /path/to/input_4 &

# wait until all background processes are ended
wait
```