

Curso Avanzado NLHPC



Contenido

- Introducción
 - Metodología
- Computación paralela
 - Computación Paralela vs Computación Secuencial
 - Memoria Compartida
 - OpenMP
 - Memoria Distribuida
 - MPI
 - Tareas Híbridas
- Tareas con Dependencias
- Programación de Tareas
- Instalación de software
 - Compilación
 - Módulos Python y R
- Problemas frecuentes

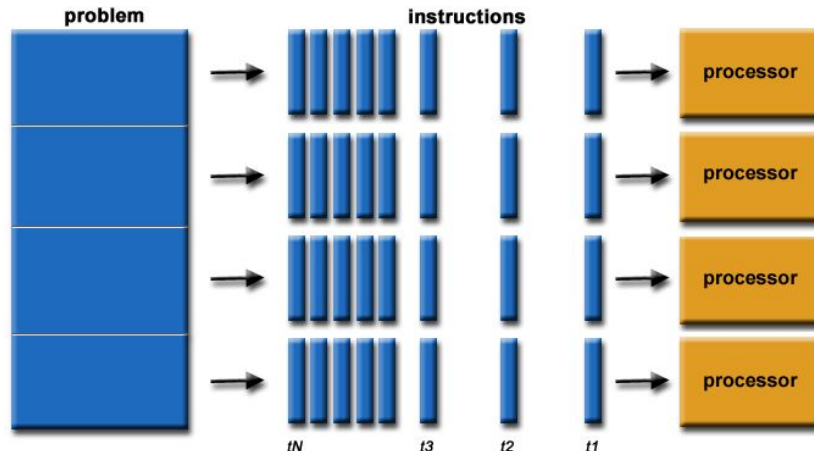
Metodología

Nuestra metodología busca ser dinámica y participativa

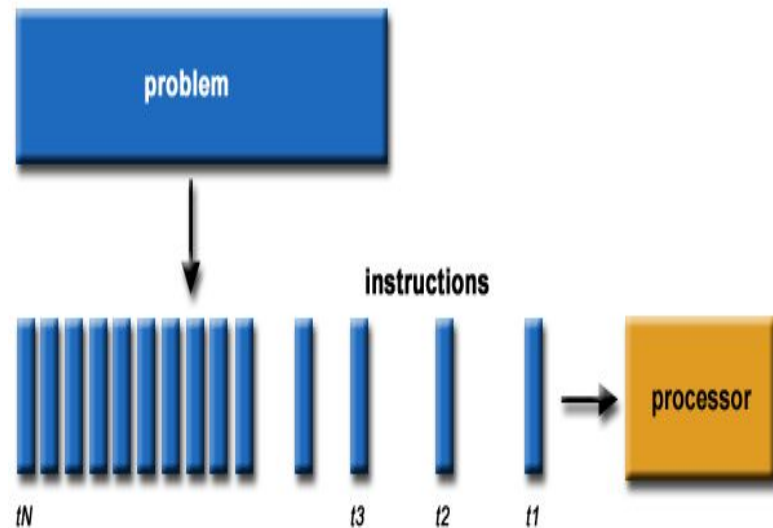
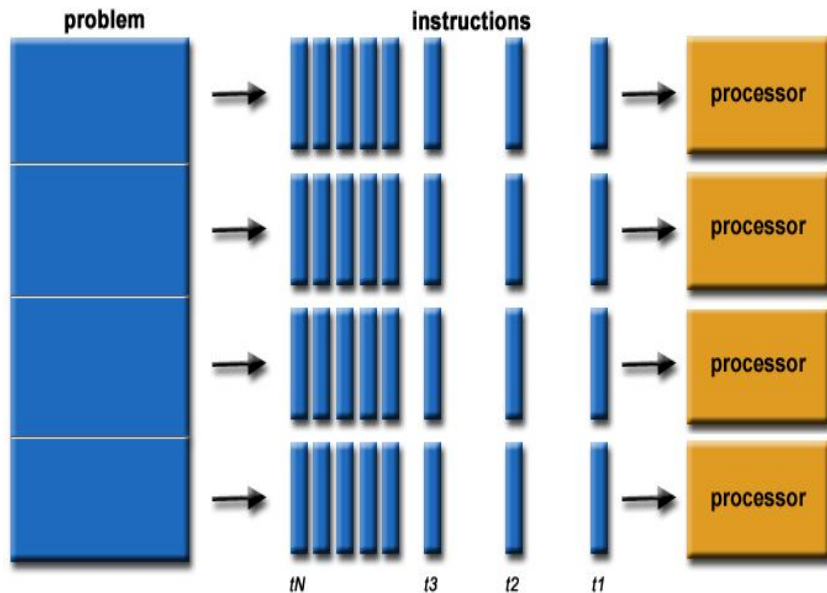
- Realizar consultas durante la presentación
- Se realizarán ejercicios en grupo
- En cada ejercicio los usuarios deberán:
 - Participar en la realización de los ejercicios
 - Compartir pantalla de manera grupal
 - Explicar los resultados de los ejercicios
- Se asignarán distintos usuarios para la realización de cada ejercicio

Computación Paralela

- Uso simultáneo de múltiples recursos computacionales
- Un problema se divide en partes discretas que se pueden resolver simultáneamente
- Cada parte se descompone en una serie de instrucciones
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores
- Se emplea un mecanismo global de control y coordinación

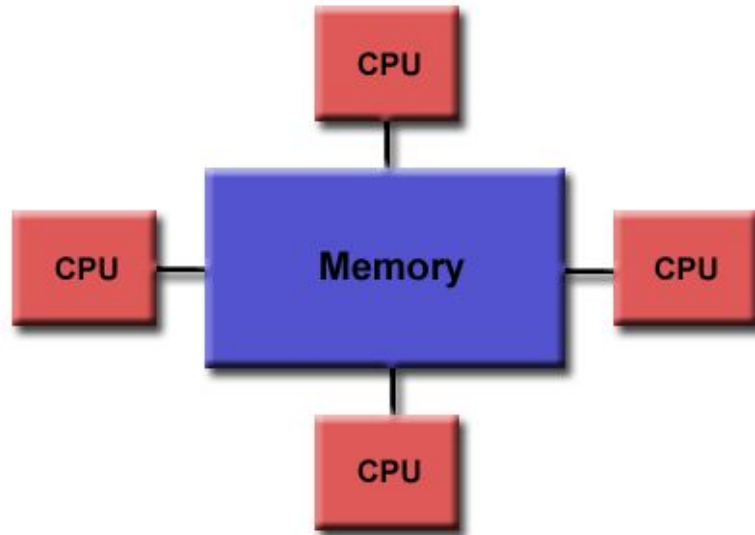


Computación Paralela v/s Secuencial



Arquitecturas de memoria de Computación Paralela: Memoria Compartida

- Los procesos comparten un espacio de memoria común
- Escriben y leen de manera asíncrona
- No es necesario especificar cómo se comunican los datos entre las tareas
- Se usan semáforos o locks para controlar el acceso a la memoria compartida



OpenMP

- Es una interfaz de programación de aplicaciones para la programación paralela de memoria compartida
- Permite añadir concurrencia: ejecutar distintas instrucciones al mismo tiempo
- Se compone de:
 - Directivas de compilación
 - Rutinas de biblioteca
 - Variables de entorno
- Modelo de programación portable y escalable
- Proporciona a los desarrolladores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas

Ejecución de Simulaciones en Slurm - Trabajos paralelos (OpenMP)

- Los trabajos paralelos diseñados para ejecutarse en un sistema multi-core (shared memory) requieren especificar el número de cpu (-c 44) a utilizar:

```
#!/bin/bash
#-----Script SBATCH - NLHPC -----
#SBATCH -J openmp
#SBATCH -p general
#SBATCH -n 1
#SBATCH -c 44
#SBATCH --mem-per-cpu=4250
#SBATCH --mail-user=example@foo.bar
#SBATCH --mail-type=ALL
#SBATCH -o openmp_%j.out
#SBATCH -e openmp_%j.err

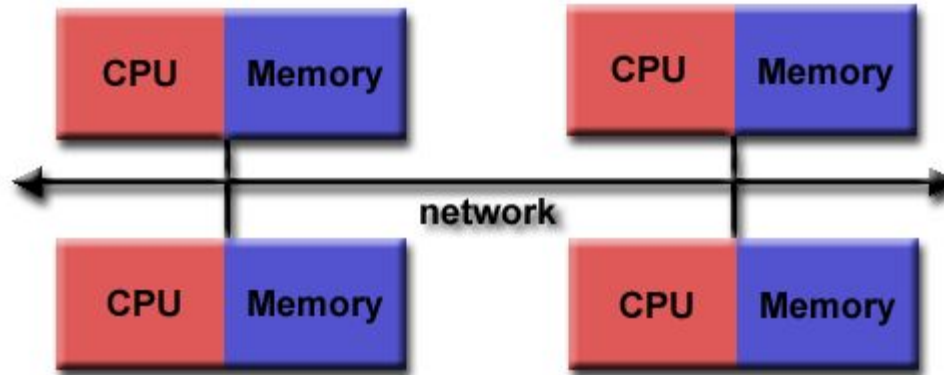
#-----Toolchain-----
ml purge
ml intel/2022.00
# -----Modulos-----
ml Programa
# -----Comando-----
./binario entrada
```


Ejercicio 1

1. Descarga el siguiente código OpenMP: [pi_omp.c](#) en tu directorio de trabajo.
2. Compile el código con el comando `icc pi_omp.c -o pi_omp -qopenmp`
3. Crea un script de ejecución para lanzarlo con sbatch, con las siguientes consideraciones:
 - a) La partición a lanzar es slims.
 - b) Ejecute 1 proceso.
 - c) Esta tarea debe tener 20 cpu asignadas.
 - d) Cada CPU requiere 2300MB de memoria RAM.
 - e) El binario a ejecutar es pi_openmp.
4. Ejecuta el script en el gestor de tareas y observa el archivo de salida.
5. Repite el ejercicio variando el número de cpu asignadas
¿Qué ocurre con el tiempo de ejecución?

Arquitecturas de memoria de Computación Paralela: Memoria Distribuida

- Esta arquitectura se basa en usar múltiples cpu con su propia memoria física privada
- Los procesos pueden operar solo con información local
- Se prefiere esta arquitectura ya que se pueden añadir múltiples unidades de procesamiento independientes entre sí
- Requiere una red de comunicación para los distintos procesos
- Los procesos intercambian datos por medio del paso y recepción de mensajes



MPI

- Es una especificación para programación de computación paralela de memoria distribuida(pasos de mensajes)
- Proporciona una librería de funciones para C, C++ o Fortran (existen implementaciones para Python y R)
- Características:
 - Estandarización
 - Portabilidad (Multiprocesadores, multicomputadores)
 - Buenas prestaciones
 - Múltiples implementaciones (MPICH, **OpenMPI**, **IMPI**, LAM-MPI, MVAPICH)
- El usuario escribirá su aplicación como un proceso secuencial del que se lanzarán varias instancias que, mediante el paso de mensajes cooperan entre sí

Ejecución de Simulaciones en Slurm - Trabajos paralelos (MPI)

- En el caso de trabajos en paralelo con MPI, es necesario especificar la cantidad de procesos en total (-n 132) y cuántos queremos que entren por nodo (--ntasks-per-node=44):

```
#!/bin/bash
#-----Script SBATCH - NLHPC -----
#SBATCH -J mpi
#SBATCH -p general
#SBATCH -n 132
#SBATCH --ntasks-per-node=44
#SBATCH -c 1
#SBATCH --mem-per-cpu=4250
#SBATCH --mail-user=example@foo.bar
#SBATCH --mail-type=ALL
#SBATCH -o mpi_%j.out
#-----Toolchain-----
ml purge
ml intel/2022.00
# -----Modulos-----
ml WRF/4.3.3-dmpar
# -----Comando-----
srun wrf.exe
```

Ejercicio 2

1. Descarga el siguiente código MPI: [pi_mpi.c](#) en tu directorio de trabajo.
2. Compile mediante el comando `mpiicc pi_mpi.c -o pi_mpi`
3. Crea un script de ejecución para lanzarlo con sbatch, con las siguientes consideraciones:
 - a) La partición a lanzar es slims.
 - b) Ejecuta 20 procesos.
 - c) Deben entrar 10 procesos por cada nodo.
 - d) Supón que cada CPU requiere 2300MB de memoria RAM.
 - e) El binario a ejecutar es `pi_mpi`
4. Ejecuta el script en el gestor de tareas y observa el archivo de salida.
5. Repite el ejercicio variando el número de procesos.
¿Qué ocurre con el tiempo de ejecución?

Ejecución de Simulaciones en Slurm - Trabajos paralelos Híbrido (MPI+OpenMP)

- En el caso de trabajos híbridos MPI+OpenMP, es necesario declarar el número de procesos MPI (-n 2) y además el número de CPU a asignar por cada una de esas tareas (-c 44):

```
#!/bin/bash
#-----Script SBATCH - NLHPC -----
#SBATCH -J hybrid
#SBATCH -p general
#SBATCH -n 2
#SBATCH --ntasks-per-node=1
#SBATCH -c 44
#SBATCH --mem-per-cpu=4250
#SBATCH --mail-user=example@foo.bar
#SBATCH --mail-type=ALL
#SBATCH -o hybrid_%j.out
#-----Toolchain-----
ml purge
ml intel/2022.00
# -----Modulos-----
ml Programa
# -----Comando-----
srun ./programa
```

Ejercicio 3

1. Descarga el siguiente código MPI_OpenMP híbrido: [pi_hybrid.cpp](#) en tu directorio de trabajo.
2. Compila utilizando el comando: `mpiicc pi_hybrid.cpp -o pi_hybrid -qopenmp`
3. Crea un script de ejecución para lanzarlo con sbatch, con las siguientes consideraciones:
 - a) La partición a lanzar es slims.
 - b) Ejecuta 2 procesos.
 - c) Cada proceso debe de tener 20 CPU asignadas.
 - d) Cada CPU requerirá 2300MB de memoria RAM.
 - e) El binario a ejecutar es `pi_hybrid`
4. Ejecuta el script en el gestor de tareas y observa el archivo de salida.

Parámetros de SLURM

Parámetro	Uso	Acción
-J	-J mi-tarea	Asigna nombre a la tarea.
-p	-p slims	Indica partición a utilizar.
-n	-n 1	Nº de procesos.
-c	-c 20	CPUs por proceso.
--ntasks-per-node	--ntasks-per-node=20	Procesos por nodo.
--mem-per-cpu	--mem-per-cpu=2300	Memoria por CPUs.
-o	-o salida_%j.out	Log de salida.
-e	-e errores_%j.err	Log de salida de errores.
--mail-user	--mail-user=user@abc.xyz	Donde se envia info del JOBS.
--mail-type	--mail-user=ALL	Tipo de información a enviar.
--array	--array=<indices>	Envía una lista de trabajos idénticos.
-t	-t <D:HH:MM:SS>	Tiempo estimado de ejecución de tarea.



Ejecutando tareas en Slurm - Trabajos secuenciales

- En este ejemplo consideramos el caso de enviar una tarea utilizando 1 núcleo:

```
#!/bin/bash
#-----Script SBATCH - NLHPC -----
#SBATCH -J secuencial
#SBATCH -p slims
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem-per-cpu=2300
#SBATCH --mail-user=example@foo.bar
#SBATCH --mail-type=ALL
#SBATCH -o secuencial_%j.out
#SBATCH -e secuencial_%j.err

#-----Toolchain-----
ml purge
ml intel/2022.00
# -----Modulos-----
ml Python/3.10.4
# -----Comando-----
python programa.py
```

Ejecutando tareas en Slurm - Array de trabajos

- Un job array en Slurm es una colección de trabajos idénticos que sólo difieren entre sí por un parámetro. La dimensión del arreglo se define con el parámetro `--array=1-n`, donde `n` es la cantidad de jobs. Se puede definir la máxima cantidad de procesos que pueden entrar simultáneamente de la siguiente forma: `1-100%10`.

```
#!/bin/bash
#-----Script SBATCH - NLHPC -----
#SBATCH -J secuencial_array
#SBATCH -p slims
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem-per-cpu=2300
#SBATCH --mail-user=example@foo.bar
#SBATCH --mail-type=ALL
#SBATCH --array=1-100%10
#SBATCH -o secuencial_array_%A_%a.out
#-----Toolchain-----
ml purge
ml intel/2022.00
# -----Modulos-----
ml Python/3.10.4
# -----Comando-----
python programa.py $SLURM_ARRAY_TASK_ID
```

Ejemplo de Gaussian y arrays

- Script para ejecutar Gaussian, el cual realizará 63 simulaciones, cada una de estas utilizará 8 CPU y podrá alcanzar un uso máximo de 8 GB de memoria ram.

```
#!/bin/bash
# -----SLURM Parameters-----
#SBATCH -J prueba
#SBATCH -p slims
#SBATCH -n 1
#SBATCH -c 8
#SBATCH --mem-per-cpu=1000
#SBATCH --mail-user=prueba@nlhpc.cl
#SBATCH --mail-type=ALL
#SBATCH --array=1-63
#SBATCH -o prueba_%A_%a.out
#SBATCH -e prueba_%A_%a.err
#-----Toolchain-----
ml purge
ml intel/2019b
# -----Módulos-----
ml g16/B.01
# -----Comandos-----
file=$(ls Child_10_*.com | sed -n ${SLURM_ARRAY_TASK_ID}p)
srun g16 $file
```

Ejercicio 4

1. Descarga el siguiente código: [average.py](#) en tu directorio de trabajo.
2. Crea un script de ejecución para lanzarlo con sbatch, con las siguientes consideraciones:
 - a) La partición a lanzar es slims.
 - b) Ejecuta un proceso.
 - c) Este proceso debe tener asignados 2 CPU.
 - d) Cada CPU requiere 2300MB de memoria RAM.
 - e) Utiliza una versión reciente de Python.
 - f) La dimensión del array es de 100 y deben entrar como máximo 10 tareas simultáneas
3. Ejecuta el script en el gestor de tareas y observa el archivo de salida.

Ejecución de tareas en Slurm - Trabajos paralelos (GPU)

- Los trabajos que utilizarán las GPUs deben indicar el parámetro `--gres=gpu:1`
Cada nodo tiene 2 GPUs:

```
#!/bin/bash
#-----Script SBATCH - NLHPC -----
#SBATCH -J GPU
#SBATCH -p gpus
#SBATCH -n 1
#SBATCH --gres=gpu:1
#SBATCH -c 1
#SBATCH --mem-per-cpu=4250
#SBATCH --mail-user=example@foo.bar
#SBATCH --mail-type=ALL
#SBATCH --array=1-100%10
#SBATCH -o GPU_%A_%a.out
#SBATCH -e GPU_%A_%a.err

#-----Toolchain-----
ml purge
ml fosscuda/2019b
# -----Modulos-----
ml NAMD/2.13-mpi
# -----Comando-----
srun namd2
```

Ejercicio 5

1. Descarga los siguientes archivos en tu directorio de trabajo: mulBy2.cu, [Makefile](#)
2. Teniendo en cuenta que es un código en Cuda y que tenemos un Makefile, mediante el comando **make** compila el código en tu directorio de trabajo
3. Crea un script de ejecución para lanzarlo con sbatch con las siguientes consideraciones:
 - a) La partición a lanzar es gpus.
 - b) Ejecuta 1 proceso.
 - c) Cada proceso debe de tener 1 CPU.
 - d) Cada CPU requiere 2300MB de memoria RAM.
 - e) El binario a ejecutar es mulBy2.
4. Ejecuta el script en el gestor de tareas y observa el archivo de salida.

Dependencias de trabajos

Las dependencias de trabajos se utilizan para aplazar el inicio de un trabajo hasta que se satisfagan las dependencias especificadas. Se especifican con la opción `--dependency` en el siguiente formato:

```
sbatch --dependency=<type:job_id[:job_id][,type:job_id[:job_id]]> ...
```

Los tipos de dependencias son las siguientes:

`after:jobid[:jobid...]`

el trabajo puede empezar después de que los trabajos especificados comiencen

`afterany:jobid[:jobid...]`

el trabajo puede empezar después de que los trabajos especificados terminen

`afternotok:jobid[:jobid...]`

el trabajo puede empezar después que los trabajos especificados terminan fallidamente

`afterok:jobid[:jobid...]`

el trabajo puede empezar después que los trabajos especificados terminan exitosamente

Ejemplo de dependencias de trabajos Ejemplo

La manera más simple de usar una dependencia del tipo *afterok*:

```
[prueba@leftrar1 ~]$ sbatch job1.sh
```

Submitted batch job 21363626

```
[prueba@leftrar1 ~]$ sbatch --dependency=afterok:21363626 job2.sh
```

Ahora cuando job1.sh finalice correctamente, el job2.sh entrará en ejecución. Si job1.sh termina con error, job2.sh no entrará en ejecución nunca pero sí quedará en cola (debe cancelarse manualmente el trabajo).

Programación de tareas usando scrontab

scrontab es el planificador de tareas de SLURM similar al planificador de tareas **crontab**.

La definición de una tarea se realiza de la siguiente manera:

```
[prueba@leftrarui ~]$ scrontab -e
```

```
#SCRON -J tarea
#SCRON -n 2
#SCRON -o logfile_%j.out
# .----- minuto (0 - 59)
# | .----- hora (0 - 23)
# | | .----- día del mes (1 - 31)
# | | | .----- mes (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- día de la semana(0 - 6) (Sunday=0 or 7)
# | | | | |
# * * * * * script a ejecutar
```

Programación de tareas usando scrontab

Editar nuestras tareas:

```
[prueba@leftrar1 ~]$ scrontab -e
```

```
#SCRON -J ejemplo_scrontab  
#SCRON -p slims  
#SCRON -n 1  
#SCRON -c 1  
#SCRON -o ejemplo_scrontab_%j.out  
*/5 * * * * miscript
```

Listar tareas definidas:

```
[prueba@leftrar1 ~]$ scrontab -l
```

Consideraciones de scrontab

Algunas consideraciones a tener presentes:

- El uso de **scrontab** hace uso de los recursos disponibles que tiene el servicio SLURM
 - No depende del nodo específico como es el caso de **crontab**
- La directriz **#SCRON** utiliza los mismos parámetros usados por **#SBATCH**
 - Los parámetros de salida como **#SCRON -o** o **#SCRON -e** deben incluir la ruta completa, de lo contrario quedarán en la carpeta *home* del usuario
- Cada script a ejecutar debe estar antecedido de los recursos a asignar
- Las tareas son **enviadas** a la cola de trabajo según la definición del tiempo indicado por el usuario
 - Si existen tareas encoladas, la nueva tarea se agregará a la lista de espera
 - **scrontab** no otorga prioridad sobre la cola de trabajo actual
- Se recomienda que el script con su trabajo contenga rutas absolutas
 - O bien que accedan a directorios mediante **cd /home/user/directorio/trabajo**
- Si una tarea enviada a través de **scrontab** es cancelada, dicha tarea no se volverá a ejecutar (quedará comentada)

Más información sobre **scrontab** en el siguiente enlace: <https://slurm.schedmd.com/scrontab.html>

Monitoreo de Simulaciones - htop

- Puede ingresar a través de ssh a un nodo en donde tenga una tarea en ejecución y ejecutar **htop**:

```
1 [|||||100.0%] 6 [ 0.0%] 11 [ 0.0%] 16 [ 0.0%]
2 [ 0.0%] 7 [ 0.0%] 12 [ 0.0%] 17 [ 0.0%]
3 [|||||100.0%] 8 [ 0.0%] 13 [ 0.0%] 18 [ 0.0%]
4 [ 0.0%] 9 [ 0.7%] 14 [ 0.0%] 19 [ 0.0%]
5 [ 0.0%] 10 [ 0.0%] 15 [ 0.0%] 20 [ 0.0%]

Mem[||||| 1.56G/62.7G] Tasks: 44, 39 thr: 3 running
Swp[ 0K/62.5G] Load average: 2.00 2.01 2.05
Uptime: 4 days, 00:36:11

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
18305 nperinet 20 0 19020 3768 892 R 100.0 0.0 2h52:34 ./LL RK4.x
18335 nperinet 20 0 17104 1584 892 R 100.0 0.0 2h47:30 ./LL RK4.x
18605 root 20 0 121M 2432 1476 R 0.0 0.0 0:00.05 htop
1 root 20 0 185M 5068 2384 S 0.0 0.0 0:11.06 /usr/lib/systemd/systemd --switched-root --system --de
669 root 20 0 112M 2000 1564 S 0.0 0.0 0:00.00 /bin/bash
671 root 20 0 36816 7236 6908 S 0.0 0.0 0:00.78 /usr/lib/systemd/systemd-journald
726 root 20 0 43808 2428 1268 S 0.0 0.0 0:00.83 /usr/lib/systemd/systemd-udev
1012 root 16 -4 51188 1620 1236 S 0.0 0.0 0:00.05 /sbin/auditd -n
1002 root 16 -4 51188 1620 1236 S 0.0 0.0 0:00.25 /sbin/auditd -n
1206 root 20 0 448M 10956 6968 S 0.0 0.0 0:00.00 /usr/sbin/NetworkManager --no-daemon
1209 root 20 0 448M 10956 6968 S 0.0 0.0 0:00.09 /usr/sbin/NetworkManager --no-daemon
1139 root 20 0 448M 10956 6968 S 0.0 0.0 0:02.34 /usr/sbin/NetworkManager --no-daemon
1144 avahi 20 0 30220 1564 1300 S 0.0 0.0 0:00.44 avahi-daemon: running [cnf004.local]
1148 dbus 20 0 28824 1772 1352 S 0.0 0.0 0:00.44 /bin/dbus-daemon --system --address=systemd: --nofork
1166 root 20 0 198M 1236 776 S 0.0 0.0 0:00.00 /usr/sbin/gssproxy -D
1167 root 20 0 198M 1236 776 S 0.0 0.0 0:00.00 /usr/sbin/gssproxy -D
1168 root 20 0 198M 1236 776 S 0.0 0.0 0:00.00 /usr/sbin/gssproxy -D
1169 root 20 0 198M 1236 776 S 0.0 0.0 0:00.00 /usr/sbin/gssproxy -D
1170 root 20 0 198M 1236 776 S 0.0 0.0 0:00.00 /usr/sbin/gssproxy -D
1164 root 20 0 198M 1236 776 S 0.0 0.0 0:00.30 /usr/sbin/gssproxy -D

F1Help F2Setup F3Search F4Filter F5Free F6SortBy F7Nice F8Nice F9Kill F10Quit
```


Monitoreo de Simulaciones - Métricas de uso

- Cuando lanza una tarea en ejecución, recibirá un enlace para ver el comportamiento de su tarea.





Monitoreo de Simulaciones - Métricas de uso


- Resumen de tarea




NLHPC
National Laboratory
for High Performance
Computing
Chile

Utilización de recursos job ID: 

Job Name: 

Usuario: 

Partition: slims

Script: sbatch -J pi -p slims -n 40 --mem-per-cpu=1000 -o pi_log.out 

Status: RUNNING

Nodos: cn[110,130-132]

MEM x CPU: 1000M

Tiempo transcurrido: 0:17:05

Max escritura disco: 0M

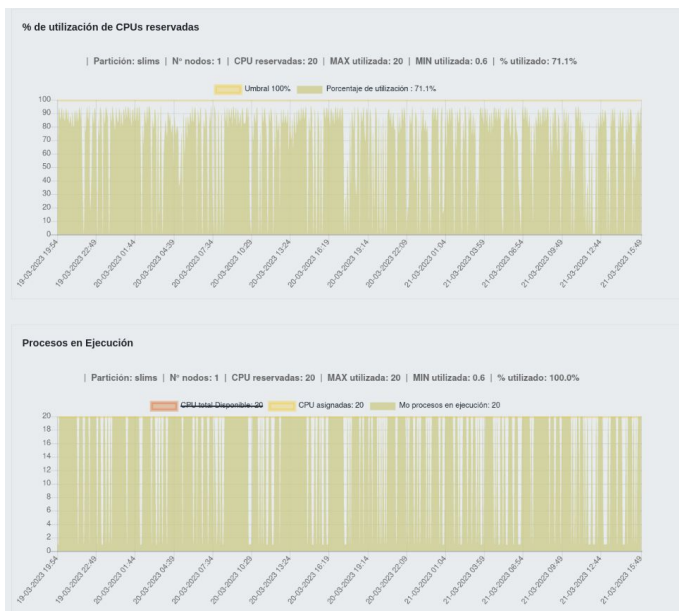
Max lectura disco: 0M

Consumo energía: 419.80K

Procesos en Ejecución: 45

Monitoreo de Simulaciones - Métricas de uso

- Uso de CPU



Monitoreo de Simulaciones - Métricas de uso

- Uso de Memoria



Almacenamiento utilizado

Para conocer el almacenamiento utilizado debe utilizar el siguiente script:

```
dbowman@leftraru2 ~ # usoDisco
```

```
Uso de disco del usuario: dbowman
```

```
Cuota      = 800G
```

```
Utilizado  = 466.05G
```

```
% de utilizacion = 58.3%
```

- El almacenamiento libre le permitirá que los resultados de sus simulaciones se almacenen.
- Si el almacenamiento llega a 100% o más, sus simulaciones se verán afectadas.

Instalación y Compilación de aplicaciones - Compiladores

- La compilación es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a lenguaje máquina
- Un compilador es un programa de computadora que “traduce” las instrucciones o código fuente desde un lenguaje a otro (usualmente código máquina)
- En ambientes HPC se busca sacar el máximo rendimiento al hardware con las aplicaciones, por lo que el 99% de los programas se encuentra compilado
- En nuestro caso, disponemos de los siguientes set de herramientas de compilación:
 - **Intel Toolchain**
 - **GNU Toolchain**
 - Estos 2 set de herramientas de compilación permiten compilar programas escritos en C, C++, Fortran
- Recomendamos siempre hacer uso de las herramientas de Intel, ya que los binarios generados suelen tener mayor rendimiento

Instalación y Compilación de aplicaciones - Flags

- Los flags de optimización se usan para:
 - Disminuir la cantidad de mensajes de depuración
 - Aumentar los niveles de aviso de errores
 - **Optimizar el código producido**
- Algunos de los flags usados en nuestras compilaciones con Intel son:
 - **-O3**: habilitar la optimización del compilador, donde 3 representa el nivel (máximo)
 - **-ipo**: la optimización interprocedural le permite al compilador analizar el código para determinar donde este puede beneficiarse de optimizaciones específicas
 - **-no-prec-div**: ofrece resultados levemente menos precisos en las divisiones de punto flotante
 - **-static-intel**: hace que las bibliotecas proporcionadas por intel se vinculen estáticamente
 - **-fPIC**: determina si el compilador genera código independiente de la posición
 - **-qopenmp**: Permite que el paralelizador genere código multiproceso basado en directivas OpenMP

Instalación y Compilación de aplicaciones - Ejemplos de compilación

- La compilación más básica de un programa en C sería la siguiente:
 - **icc hello_world.c -o hello_world**
 - Esta compilación generará un ejecutable llamado “hello_world” el cual puede ser ejecutado de la siguiente forma: **./hello_world**
- Si queremos compilar un programa en C que soporte MPI, la compilación sería la siguiente:
 - **mpiicc hello_world.c -o hello_world -O3 -axCORE-AVX512,AVX -ipo -no-prec-div -static-intel -fPIC -qopenmp**
- La opción **-axCORE-AVX512,AVX** la utilizamos para que nuestro ejecutable pueda ejecutarse en los distintos tipos de nodos que tenemos actualmente
- Compiladores disponibles: **gcc/icc/mpiicc** (C) **g++/icpc/mpiicpc** (C++), **gfortran/ifort/mpiifort** (Fortran)

Instalación de módulos en Python y R

- En el caso de que queramos instalar módulos de Python localmente en nuestro directorio utilizaremos el siguiente comando:
 - Cargar el módulo de Python: **m1 Python/3.8.3**
 - **pip install programa --no-binary :all: --user**
 - El parámetro **--no-binary :all:** intentará compilar el módulo a instalar
 - El parámetro **--user** instalará en nuestro directorio el módulo
- Para instalar paquetes de R localmente en nuestro directorio:
 - Cargamos el módulo de R: **m1 R/4.0.0**
 - **R CMD INSTALL -l /home/prueba/R/library programa**
 - Para decirle a R que utilice los paquetes instalados por mi: **library("paquete", lib.loc="/home/prueba/R/library")**
- Para instalaciones generales, debe contactar a soporte@nlhpc.cl

Problemas Frecuentes - Falta de memoria RAM

- Debemos tener en cuenta que la memoria RAM que SLURM reserva por defecto es de 1000MB (1GB). Un típico error de cancelación de tareas es por utilizar más de la memoria reservada:

```
/tmp/slurmd/job136839939/slurm_script: line 15: 23547 Killed                  ./programa.sh
slurmstepd: error: Detected 1 oom-kill event(s) in step 136839939.batch cgroup. Some of your processes
may have been killed by the cgroup out-of-memory handler.
```

- Si su tarea ocupa más de la memoria por defecto (1GB), se puede utilizar el siguiente parámetro en SLURM:
 - **#SBATCH --mem-per-cpu=2300**
- Esto hará que SLURM reserve 2300MB (2.2GB) por CPU asociado al trabajo
- Para más detalles de nuestra infraestructura visitar: [Hardware Disponible](#)

Problemas Frecuentes - Subutilización de CPU y RAM

- Nuestro objetivo en el cluster es velar por el uso óptimo de los recursos, por lo que estamos continuamente monitoreando el uso de los nodos.
- El procedimiento en estos casos es el siguiente:
 - Se envía una notificación a las 2 horas de ejecución de la tarea que subutilice recursos.
 - Si mantiene el comportamiento en las próximas 2 horas, se envía una notificación informando la cancelación de la tarea.
 - En el correo de cancelación se adjunta un link para ver el sistema de métricas y ver de manera sencilla el del uso de CPU y Memoria de su tarea, con el fin de que puedan realizar los cambios respectivos en las próximas tareas.

Problemas Frecuentes - Subutilización de CPU y RAM

Qué es la subutilización:

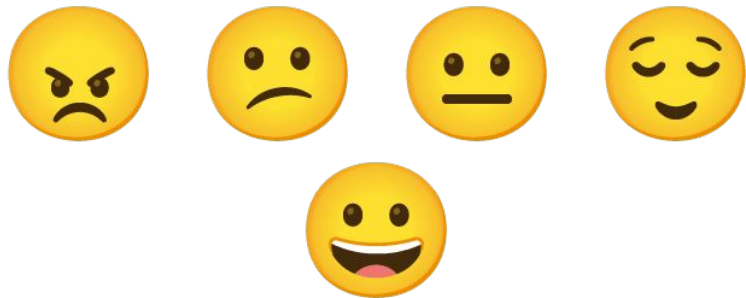
- Es subutilización si en las primeras 2 horas se utiliza un <50% de los recursos reservados
- Es subutilización en las siguientes horas según la siguiente tabla:

Partición	uso CPU	uso RAM
Slims	<75%	<70%
General	<70%	<75%
LargeMem	<80%	<75%
GPU	<80%	<75%

Queremos saber tu opinión

- Encuesta de satisfacción
- Correo de agradecimiento

¿Cómo evalúa el curso recibido?



¡Gracias por participar!

www.nlhpc.cl

2023

