Table of Contents

Information about the batch system (SLURM)

Please confer /etc/slurm/README.

The documentation of Slurm can be found ?here.

Overview

Slurm offers interactive and batch jobs (scripts submitted into the system). The relevant commands are `srun` and `sbatch`. The `srun` command can be used to spawn processes (**please do not use mpiexec**), both from the frontend and from within a batch script. You can also get a shell on a node to work locally there (e.g. to compile your application natively for a special platform).

Available Partitions

Please note that there is no default partition configured. In order to run a job, you have to specify one of the following partitions, using the `--partition=...` switch:

Name	Description
dp-cn	dp-cn[01-50], DEEP-EST Cluster nodes (Xeon Skylake)
dp-dam	dp-dam[01-16], DEEP-EST Dam nodes (Xeon Cascadelake + 1 V100 + 1 Stratix 10)
dp-dam-ext	dp-dam[09-16], DEEP-EST Dam nodes connected with Extoll Tourmalet
dp-esb	dp-esb[01-25], DEEP-EST Esb nodes (Xeon Cascadelake + 1 V100)
dp-sdv-esb	dp-sdv-esb[01-02], DEEP-EST ESB Test nodes (Xeon Cascadelake + 1 V100)
ml-gpu	ml-gpu[01-03], GPU test nodes for ML applications (up to 4 V100 cards)
sdv	deeper-sdv[01-16], cluster test nodes with Xeon Haswell CPU
extoll	deeper-sdv[01-16] (these nodes use an Extoll Tourmalet fabric)
knl	knl[01,04-06], KNL nodes
knl256	knl[01,05], KNL nodes with 64 cores
knl272	knl[04,06], KNL nodes with 68 cores
snc4	knl[05], KNL node in snc4 memory mode
psgw-cluster	gateway test node
psgw-booster	gateway test node
debug	all compute nodes (no gateways)

Anytime, you can list the state of the partitions with the `sinfo` command. The properties of a partition can be seen using

```
scontrol show partition <partition>
```

Remark about environment

By default, Slurm passes the environment from your job submission session directly to the execution environment. Please be aware of this when running jobs with `srun` or when submitting scripts with `sbatch`. This behavior can be controlled via the `--export` option. Please refer to the ?Slurm documentation to get more information about this.

In particular, when submitting job scripts, it is recommended to load the necessary modules within the script and submit the script from a clean environment.

An introductory example

Suppose you have an mpi executable named `hello_mpi`. There are three ways to start the binary.

From a shell on a node

First, start a shell on a node. You would like to run your mpi task on 4 machines with 2 tasks per machine:

```
[kreutz1@deepv /p/project/cdeep/kreutz1/Temp]$ srun -A deep -p dp-cn -N 4 -n 8 -t 00:30:00 --pty /bin/bash -i
[kreutz1@dp-cn01 /p/project/cdeep/kreutz1/Temp]$
```

The environment is transported to the remote shell, no `.profile`, `.bashrc`, … are sourced (especially not the modules default from `/etc/profile.d/modules.sh`). As of March 2020, an account has to be specified using the `--account` (short `-A`) option, which is "deep" for the project members. For people not included in the DEEP-EST project, please use the "Budget" name you received along with your account creation.

Once you get to the compute node, start your application using `srun`. Note that the number of tasks used is the same as specified in the initial `srun` command above (4 nodes with two tasks each):

```
[kreutz1@deepv Temp]$ srun -A deep -p dp-cn -N 4 -n 8 -t 00:30:00 --pty /bin/bash -i
[kreutz1@dp-cn01 Temp]$ srun ./MPI_HelloWorld
Hello World from rank 3 of 8 on dp-cn02
Hello World from rank 7 of 8 on dp-cn04
Hello World from rank 2 of 8 on dp-cn02
Hello World from rank 6 of 8 on dp-cn04
Hello World from rank 0 of 8 on dp-cn01
Hello World from rank 4 of 8 on dp-cn03
Hello World from rank 1 of 8 on dp-cn01
Hello World from rank 5 of 8 on dp-cn03
```

You can ignore potential warnings about the cpu binding. ParaStation will pin your processes.

Running directly from the front ends

You can run the application directly from the frontend, bypassing the shell:

```
[kreutz1@deepv Temp]$ srun -A deep -p dp-cn -N 4 -n 8 -t 00:30:00 ./MPI_HelloWorld
Hello World from rank 7 of 8 on dp-cn04
Hello World from rank 3 of 8 on dp-cn02
Hello World from rank 6 of 8 on dp-cn04
Hello World from rank 2 of 8 on dp-cn02
Hello World from rank 4 of 8 on dp-cn03
Hello World from rank 0 of 8 on dp-cn01
Hello World from rank 1 of 8 on dp-cn01
Hello World from rank 5 of 8 on dp-cn03
```

In this case, it can be useful to create an allocation which you can use for several runs of your job:

```
[kreutz1@deepv Temp]$ salloc -A deep -p dp-cn -N 4 -n 8 -t 00:30:00
salloc: Granted job allocation 69263
[kreutz1@deepv Temp]$ srun ./MPI_HelloWorld
Hello World from rank 7 of 8 on dp-cn04
Hello World from rank 3 of 8 on dp-cn02
Hello World from rank 6 of 8 on dp-cn04
Hello World from rank 2 of 8 on dp-cn02
Hello World from rank 5 of 8 on dp-cn03
Hello World from rank 1 of 8 on dp-cn01
Hello World from rank 4 of 8 on dp-cn03
Hello World from rank 0 of 8 on dp-cn01
...
# several more runs
```

```
...
[kreutz1@deepv Temp]$ exit
exit
salloc: Relinquishing job allocation 69263
```

Batch script

Given the following script `hello_cluster.sh`:

```
#!/bin/bash

#SBATCH --partition=dp-cn
#SBATCH -A deep
#SBATCH -N 4
#SBATCH -n 8
#SBATCH -o /p/project/cdeep/kreutz1/hello_cluster-%j.out
#SBATCH -e /p/project/cdeep/kreutz1/hello_cluster-%j.err
#SBATCH --time=00:10:00

srun ./MPI_HelloWorld
```

This script requests 4 nodes with 8 tasks, specifies the stdout and stderr files, and asks for 10 minutes of walltime. Submit:

```
[kreutz1@deepv Temp]$ sbatch hello_cluster.sh
Submitted batch job 69264
```

Check what it's doing:

```
[kreutz1@deepv Temp]$ squeue -u $USER
            JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            69264     dp-cn hello_cl  kreutz1 CG       0:04      4 dp-cn[01-04]
```

Check the result:

```
[kreutz1@deepv Temp]$ cat /p/project/cdeep/kreutz1/hello_cluster-69264.out
Hello World from rank 6 of 8 on dp-cn04
Hello World from rank 3 of 8 on dp-cn02
Hello World from rank 0 of 8 on dp-cn01
Hello World from rank 4 of 8 on dp-cn03
Hello World from rank 2 of 8 on dp-cn02
Hello World from rank 7 of 8 on dp-cn04
Hello World from rank 5 of 8 on dp-cn03
Hello World from rank 1 of 8 on dp-cn01
```

Heterogeneous jobs

As of version 17.11 of Slurm, heterogeneous jobs are supported. For example, the user can run:

```
srun --account=deep --partition=dp-cn -N 1 -n 1 hostname : --partition=dp-dam -N 1 -n 1 hostname
dp-cn01
dp-dam01
```

Please notice the `:` separating the definitions for each sub-job of the heterogeneous job. Also, please be aware that it is possible to have more than two sub-jobs in a heterogeneous job.

The user can also request several sets of nodes in a heterogeneous allocation using `salloc`. For example:

```
salloc --partition=dp-cn -N 2 : --partition=dp-dam -N 4
```

In order to submit a heterogeneous job via `sbatch`, the user needs to set the batch script similar to the following one:

```
#!/bin/bash

#SBATCH --job-name=imb_execute_1
#SBATCH --account=deep
#SBATCH --mail-user=
#SBATCH --mail-type=ALL
#SBATCH --output=job.out
#SBATCH --error=job.err
#SBATCH --time=00:02:00

#SBATCH --partition=dp-cn
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=1

#SBATCH packjob

#SBATCH --partition=dp-dam
#SBATCH --constraint=
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=1

srun ./app_cn : ./app_dam
```

Here the `packjob` keyword allows to define Slurm parameters for each sub-job of the heterogeneous job. Some Slurm options can be defined once at the beginning of the script and are automatically propagated to all sub-jobs of the heterogeneous job, while some others (i.e. `--nodes` or `--ntasks`) must be defined for each sub-job. You can find a list of the propagated options on the ?Slurm documentation.

When submitting a heterogeneous job with this colon notation using ParaStationMPI, a unique `MPI_COMM_WORLD` is created, spanning across the two partitions. If this is not desired, one can use the `--pack-group` key to submit independent job steps to the different node-groups of a heterogeneous allocation:

```
srun --pack-group=0 ./app_cn ; srun --pack-group=1 ./app_dam
```

Using this configuration implies that inter-communication must be established manually by the applications during run time, if needed.

For more information about heterogeneous jobs please refer to the ?relevant page of the Slurm documentation.

Heterogeneous jobs with MPI communication across modules

In order to establish MPI communication across modules using different interconnect technologies, some special Gateway nodes must be used. On the DEEP-EST system, MPI communication across gateways is needed only between Infiniband and Extoll interconnects.

Attention: Only ParaStation MPI supports MPI communication across gateway nodes.

This is an example job script for setting up an Intel MPI benchmark between a Cluster and a DAM node using a IB ↔ Extoll gateway for MPI communication:

```
#!/bin/bash

# Script to launch IMB PingPong between DAM-CN using 1 Gateway
# Use the gateway allocation provided by SLURM
# Use the packjob feature to launch separately CM and DAM executable


# General configuration of the job
```

```
#SBATCH --job-name=modular-imb
#SBATCH --account=deep
#SBATCH --time=00:10:00
#SBATCH --output=modular-imb-%j.out
#SBATCH --error=modular-imb-%j.err

# Configure the gateway daemon
#SBATCH --gw_num=1
#SBATCH --gw_psgwd_per_node=1

# Configure node and process count on the CM
#SBATCH --partition=dp-cn
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1

#SBATCH packjob

# Configure node and process count on the DAM
#SBATCH --partition=dp-dam-ext
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1

# Echo job configuration
echo "DEBUG: SLURM_JOB_NODELIST=$SLURM_JOB_NODELIST"
echo "DEBUG: SLURM_NNODES=$SLURM_NNODES"
echo "DEBUG: SLURM_TASKS_PER_NODE=$SLURM_TASKS_PER_NODE"


# Set the environment to use PS-MPI
module --force purge
module use $OTHERSTAGES
module load Stages/Devel-2019a
module load Intel
module load ParaStationMPI

# Show the hosts we are running on
srun hostname : hostname

# Execute
APP="./IMB-MPI1 Uniband"
srun ${APP}  : ${APP}
```

Attention: During the first part of 2020, only the DAM nodes will have Extoll interconnect, while the CM and the ESB nodes will be connected via Infiniband. This will change later during the course of the project (expected Summer 2020), when the ESB will be equipped with Extoll connectivity (Infiniband will be removed from the ESB and left only for the CM).

A general description of how the user can request and use gateway nodes is provided at ?this section of the JURECA documentation.

Attention: some information provided on the JURECA documentation do not apply for the DEEP system. In particular:

- as of 31/03/2020, the DEEP system has 2 gateway nodes.
- As of 09/01/2020 the gateway nodes are exclusive to the job requesting them. Given the limited number of gateway nodes available on the system, this may change in the future.
- As of 09/04/2020 the `xenv` utility (necessary on JURECA to load modules for different architectures - Haswell and KNL) is not needed any more on DEEP when using the latest version of ParaStationMPI (currently available in the `Devel-2019a` stage and soon available on the default production stage).

Workflows

The new version of the installed slurm now supports workflows. The idea is to have an overlap between the dependent jobs so that they can communicate the data over the network instead of writing and reading it on storage. To enable the workflows, we have introduced a new switch `delay`

to `sbatch` command. Here is a simple example script.

```
[huda1@deepv scripts]$ cat test.sh
#!/bin/sh

NAME=$(hostname)
echo "$NAME: Going to sleep for $1 seconds"
sleep $1
echo "$NAME: Awake"

[huda1@deepv scripts]$ cat batch_workflow.sh
#!/bin/bash
#SBATCH -p sdv -N2 -t3

#SBATCH packjob

#SBATCH -p sdv -N1 -t3 --delay 2

srun test.sh 175

[huda1@deepv scripts]$
```

In the above `sbatch` script, the usage of `--delay` can be seen. It takes thee values in minutes. The idea is to delay the corresponding job of a heterogeneous job by the provided number of minutes from the start of the first job in the job pack. After submission of this job pack, slurm divides it into separate jobs at the time of the resource reservation. So you can see multiple jobs in the output of `squeue` command. Here is the example execution of this script.

```
[huda1@deepv scripts]$ sbatch batch_workflow.sh
Submitted batch job 81458
[huda1@deepv scripts]$ squeue -u huda1
            JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            81458       sdv batch_wo    huda1 CF       0:01      2 deeper-sdv[02-03]
            81459       sdv batch_wo    huda1 PD       0:00      1 (Reservation)

[huda1@deepv scripts]$
```

Here the second job(81458) will start 2 minutes after the start of the first job(81459). Similarly, the output files will be different for each separated job in the job pack. the final outputs are:

```
[huda1@deepv scripts]$ cat slurm-81458.out
deeper-sdv02: Going to sleep for 175 seconds
deeper-sdv03: Going to sleep for 175 seconds
deeper-sdv02: Awake
deeper-sdv03: Awake

[huda1@deepv scripts]$ cat slurm-81459.out
deeper-sdv01: Going to sleep for 175 seconds
deeper-sdv01: Awake

[huda1@deepv scripts]$
```

Another feature to note is that if there are multiple jobs in a job pack and any number of consecutive jobs have the same `delay` values, they are combined into a new heterogeneous job. Here is an example of such a script:

```
[huda1@deepv scripts]$ cat batch_workflow_complex.sh
#!/bin/bash

#SBATCH -p sdv -N 2 -t 3
#SBATCH -J first
```

```
#SBATCH packjob

#SBATCH -p sdv -N 1 -t 3 --delay 2
#SBATCH -J second

#SBATCH packjob

#SBATCH -p sdv -N 1 -t 2 --delay 2
#SBATCH -J second

#SBATCH packjob

#SBATCH -p sdv -N 2 -t 3 --delay 4
#SBATCH -J third

if [ "$SLURM_JOB_NAME" == "first" ]
then
       srun ./test.sh 150

elif [ "$SLURM_JOB_NAME" == "second" ]
then
       srun ./test.sh 150 : ./test.sh 115

elif [ "$SLURM_JOB_NAME" == "third" ]
then
       srun ./test.sh 155

fi

[huda1@deepv scripts]$
```

Note the `delay` values for the second and third job in the script are equal. Also, note the usage of the environment variable `SLURM_JOB_NAME` in the script to decide which command to run in which job. The example execution leads to the following:

```
[huda1@deepv scripts]$ sbatch batch_workflow_complex.sh
Submitted batch job 81460

[huda1@deepv scripts]$ squeue -u huda1
            JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          81461+0       sdv   second    huda1 PD       0:00      1 (Resources)
          81461+1       sdv   second    huda1 PD       0:00      1 (Resources)
            81463       sdv    third    huda1 PD       0:00      2 (Resources)
            81460       sdv    first    huda1 PD       0:00      2 (Resources)

[huda1@deepv scripts]$
```

Note that the submitted heterogeneous job has been divided into a single job (81460), a job pack (81461) and again a single job (81643). Similarly, three different output files will be generated, one for each new job.

```
[huda1@deepv scripts]$ cat slurm-81460.out
deeper-sdv03: Going to sleep for 150 seconds
deeper-sdv04: Going to sleep for 150 seconds
deeper-sdv03: Awake
deeper-sdv04: Awake

[huda1@deepv scripts]$ cat slurm-81461.out
deeper-sdv01: Going to sleep for 150 seconds
deeper-sdv02: Going to sleep for 115 seconds
deeper-sdv02: Awake
deeper-sdv01: Awake
```

```
[huda1@deepv scripts]$ cat slurm-81463.out
deeper-sdv01: Going to sleep for 155 seconds
deeper-sdv02: Going to sleep for 155 seconds
deeper-sdv01: Awake
deeper-sdv02: Awake

[huda1@deepv scripts]$
```

If a job exits earlier than the allocated time asked by the user, the corresponding reservation for this job is deleted automatically and the resources become available for the other jobs. However, users should be careful with the requested time when submitting workflows as the larger time values can delay the scheduling of the workflows depending on the situation of the resources.

Information on past jobs and accounting

The `sacct` command can be used to enquire the Slurm database about a past job.

```
[kreutz1@deepv Temp]$ sacct -j 69268
      JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
69268+0            bash      dp-cn deepest-a+         96  COMPLETED      0:0
69268+0.0    MPI_Hello+            deepest-a+          2  COMPLETED      0:0
69268+1            bash     dp-dam deepest-a+        384  COMPLETED      0:0
```

On the Cluster (CM) nodes it's possible to query the consumed energy for a certain job:

```
[kreutz1@deepv kreutz1]$ sacct -o ConsumedEnergy,JobName,JobID,CPUTime,AllocNodes -j 69326
ConsumedEnergy    JobName        JobID    CPUTime AllocNodes
-------------- ---------- ------------ ---------- ----------
      496.70K hpl_MKL_O+ 69326          16:28:48          1
            0      batch 69326.batch    16:28:48          1
      496.70K xlinpack_+ 69326.0        08:10:24          1
```

This feature will also be for the ESB nodes.

FAQ

Is there a cheat sheet for all main Slurm commands?

Yes, it is available ?here.

Why's my job not running?

You can check the state of your job with

```
scontrol show job <job id>
```

In the output, look for the `Reason` field.

You can check the existing reservations using

```
scontrol show res
```

How can I check which jobs are running in the machine?

Please use the `squeue` command (the "-u $USER" option to only list jobs belonging to your user id).

How do I do chain jobs with dependencies?

Please confer the `sbatch`/`srun` man page, especially the

```
-d, --dependency=<dependency_list>
```

entry.

Also, jobs can be chained after they have been submitted using the `scontrol` command by updating their `Dependency` field.

How can check the status of partitions and nodes?

The main command to use is `sinfo`. By default, when called alone, `sinfo` will list the available partitions and the number of nodes in each partition in a given status. For example:

```
[deamicis1@deepv hybridhello]$ sinfo
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
sdv             up   20:00:00     11   idle deeper-sdv[06-16]
knl             up   20:00:00      1  drain knl01
knl             up   20:00:00      3   idle knl[04-06]
knl256          up   20:00:00      1  drain knl01
knl256          up   20:00:00      1   idle knl05
knl272          up   20:00:00      2   idle knl[04,06]
snc4            up   20:00:00      1   idle knl05
extoll          up   20:00:00     11   idle deeper-sdv[06-16]
ml-gpu          up   20:00:00      3   idle ml-gpu[01-03]
dp-cn           up   20:00:00      1  drain dp-cn33
dp-cn           up   20:00:00      5   resv dp-cn[09-10,25,49-50]
dp-cn           up   20:00:00     44   idle dp-cn[01-08,11-24,26-32,34-48]
dp-dam          up   20:00:00      1 drain* dp-dam08
dp-dam          up   20:00:00      2  drain dp-dam[03,07]
dp-dam          up   20:00:00      3   resv dp-dam[05,09-10]
dp-dam          up   20:00:00      2  alloc dp-dam[01,04]
dp-dam          up   20:00:00      8   idle dp-dam[02,06,11-16]
dp-dam-ext      up   20:00:00      2   resv dp-dam[09-10]
dp-dam-ext      up   20:00:00      6   idle dp-dam[11-16]
dp-esb          up   20:00:00     51 drain* dp-esb[11,26-75]
dp-esb          up   20:00:00      2  drain dp-esb[08,23]
dp-esb          up   20:00:00      2  alloc dp-esb[09-10]
dp-esb          up   20:00:00     20   idle dp-esb[01-07,12-22,24-25]
dp-sdv-esb      up   20:00:00      2   resv dp-sdv-esb[01-02]
psgw-cluster    up   20:00:00      1   idle nfgw01
psgw-booster    up   20:00:00      1   idle nfgw02
debug           up   20:00:00      1 drain* dp-dam08
debug           up   20:00:00      4  drain dp-cn33,dp-dam[03,07],knl01
debug           up   20:00:00     10   resv dp-cn[09-10,25,49-50],dp-dam[05,09-10],dp-sdv-esb[01-02]
debug           up   20:00:00      2  alloc dp-dam[01,04]
debug           up   20:00:00     69   idle deeper-sdv[06-16],dp-cn[01-08,11-24,26-32,34-48],dp-dam[02,06,11-16],knl[04-06
```

Please refer to the man page for `sinfo` for more information.

Can I join stderr and stdout like it was done with `-joe` in Torque?

Not directly. In your batch script, redirect stdout and stderr to the same file:

```
...
#SBATCH -o /point/to/the/common/logfile-%j.log
#SBATCH -e /point/to/the/common/logfile-%j.log
...
```

(The `%j` will place the job id in the output file). N.B. It might be more efficient to redirect the output of your script's commands to a dedicated file.

What is the default binding/pinning behaviour on DEEP?

DEEP uses a ParTec-modified version of Slurm called psslurm. In psslurm, the options concerning binding and pinning are different from the ones provided in Vanilla Slurm. By default, psslurm will use a *by rank* pinning strategy, assigning each Slurm task to a different physical thread on the node starting from OS processor 0. For example:

```
[deamicis1@deepv hybridhello]$ OMP_NUM_THREADS=1 srun -N 1 -n 4 -p dp-cn ./HybridHello | sort -k9n -k11n
Hello from node dp-cn50, core 0; AKA rank 0, thread 0
Hello from node dp-cn50, core 1; AKA rank 1, thread 0
Hello from node dp-cn50, core 2; AKA rank 2, thread 0
Hello from node dp-cn50, core 3; AKA rank 3, thread 0
```

Attention: please be aware that the psslurm affinity settings only affect the tasks spawned by Slurm. When using threaded applications, the thread affinity will be inherited from the task affinity of the process originally spawned by Slurm. For example, for a hybrid MPI-OpenMP application:

```
[deamicis1@deepv hybridhello]$ OMP_NUM_THREADS=4 srun -N 1 -n 4 -c 4 -p dp-dam ./HybridHello | sort -k9n -k11n
Hello from node dp-dam01, core 0-3; AKA rank 0, thread 0
Hello from node dp-dam01, core 0-3; AKA rank 0, thread 1
Hello from node dp-dam01, core 0-3; AKA rank 0, thread 2
Hello from node dp-dam01, core 0-3; AKA rank 0, thread 3
Hello from node dp-dam01, core 4-7; AKA rank 1, thread 0
Hello from node dp-dam01, core 4-7; AKA rank 1, thread 1
Hello from node dp-dam01, core 4-7; AKA rank 1, thread 2
Hello from node dp-dam01, core 4-7; AKA rank 1, thread 3
Hello from node dp-dam01, core 8-11; AKA rank 2, thread 0
Hello from node dp-dam01, core 8-11; AKA rank 2, thread 1
Hello from node dp-dam01, core 8-11; AKA rank 2, thread 2
Hello from node dp-dam01, core 8-11; AKA rank 2, thread 3
Hello from node dp-dam01, core 12-15; AKA rank 3, thread 0
Hello from node dp-dam01, core 12-15; AKA rank 3, thread 1
Hello from node dp-dam01, core 12-15; AKA rank 3, thread 2
Hello from node dp-dam01, core 12-15; AKA rank 3, thread 3
```

Be sure to explicitly set the thread affinity settings in your script (e.g. exporting environment variables) or directly in your code. Taking the previous example:

```
[deamicis1@deepv hybridhello]$ OMP_NUM_THREADS=4 OMP_PROC_BIND=close srun -N 1 -n 4 -c 4 -p dp-dam ./HybridHello | sort -k
Hello from node dp-dam01, core 0; AKA rank 0, thread 0
Hello from node dp-dam01, core 1; AKA rank 0, thread 1
Hello from node dp-dam01, core 2; AKA rank 0, thread 2
Hello from node dp-dam01, core 3; AKA rank 0, thread 3
Hello from node dp-dam01, core 4; AKA rank 1, thread 0
Hello from node dp-dam01, core 5; AKA rank 1, thread 1
Hello from node dp-dam01, core 6; AKA rank 1, thread 2
Hello from node dp-dam01, core 7; AKA rank 1, thread 3
Hello from node dp-dam01, core 8; AKA rank 2, thread 0
Hello from node dp-dam01, core 9; AKA rank 2, thread 1
Hello from node dp-dam01, core 10; AKA rank 2, thread 2
Hello from node dp-dam01, core 11; AKA rank 2, thread 3
Hello from node dp-dam01, core 12; AKA rank 3, thread 0
Hello from node dp-dam01, core 13; AKA rank 3, thread 1
Hello from node dp-dam01, core 14; AKA rank 3, thread 2
Hello from node dp-dam01, core 15; AKA rank 3, thread 3
```

Please refer to the ?following page on the JURECA documentation for more information about how to affect affinity on the DEEP system using psslurm options. Please be aware that different partitions on DEEP have different number of sockets per node and cores/threads per socket with respect to JURECA. Please refer to the System_overview or run the `lstopo-no-graphics` on the compute nodes to get more information about the hardware configuration on the different modules.

How do I use SMT on the DEEP CPUs?

On DEEP, SMT is enabled by default on all nodes. Please be aware that on all JSC systems (including DEEP), each hardware thread is exposed by the OS as a separate CPU. For a *n*-core node, with *m* hardware threads per core, the OS cores from *0* to *n-1* will correspond to the first hardware thread of

all hardware cores (from all sockets), the OS cores from *n* to *2n-1* to the second hardware thread of the hardware cores, and so on.

For instance, on a Cluster node (with two sockets with 12 cores each, with 2 hardware threads per core):

```
[deamicis1@deepv hybridhello]$ srun -N 1 -n 1 -p dp-cn lstopo-no-graphics --no-caches --no-io --no-bridges --of ascii
????????????
```

To exploit SMT, simply run a job using a number of tasks*threads_per_task higher than the number of physical cores available on a node. Please refer to the ?relevant page of the JURECA documentation for more information on how to use SMT on the DEEP nodes.

Attention: currently the only way to assign Slurm tasks to hardware threads belonging to the same hardware core is to use the `--cpu-bind` option of psslurm using `mask_cpu` to provide affinity masks for each task. For example:

```
[deamicis1@deepv hybridhello]$ OMP_NUM_THREADS=2 OMP_PROC_BIND=close OMP_PLACES=threads srun -N 1 -n 2 -p dp-dam --cpu-bin
Hello from node dp-dam01, core 0; AKA rank 0, thread 0
Hello from node dp-dam01, core 48; AKA rank 0, thread 1
Hello from node dp-dam01, core 1; AKA rank 1, thread 0
Hello from node dp-dam01, core 49; AKA rank 1, thread 1
```

This can be cumbersome for jobs using a large number of tasks per node. In such cases, a tool like ?hwloc (currently available on the compute nodes, but not on the login node!) can be used to calculate the affinity masks to be passed to psslurm.