Table of Contents

Information about the batch system (SLURM) **2**
- Overview 2
- Available Partitions 2
- Remark about environment 2
- An introductory example 2
  - From a shell on a node 3
  - Running directly from the front ends 3
  - Batch script 4
- Heterogeneous jobs 4
  - Heterogeneous jobs with MPI communication across modules 5
- Information on past jobs and accounting 6
- FAQ 7
  - Is there a cheat sheet for all main Slurm commands? 7
  - Why's my job not running? 7
  - How can I check which jobs are running in the machine? 7
  - How do I do chain jobs with dependencies? 7
  - How can check the status of partitions and nodes? 7
  - Can I join stderr and stdout like it was done with -joe in Torque? 8
  - What is the default binding/pinning behaviour on DEEP? 8
  - How do I use SMT on the DEEP CPUs? 9

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-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 -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`).

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 -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 -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 -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 -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 --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

#SBATCH --job-name=imb
##SBATCH --account=cdeep
#SBATCH --output=IMB-%j.out
#SBATCH --error=IMB-%j.err
#SBATCH --time=00:05:00
#SBATCH --gw_num=1
#SBATCH --gw_binary=/opt/parastation/bin/psgwd.extoll
#SBATCH --gw_psgwd_per_node=1
```

```
#SBATCH --partition=dp-cn
#SBATCH --nodes=1
#SBATCH --ntasks=1

#SBATCH packjob

#SBATCH --partition=dp-dam-ext
#SBATCH --nodes=1
#SBATCH --ntasks=1

echo "DEBUG: SLURM_JOB_NODELIST=$SLURM_JOB_NODELIST"
echo "DEBUG: SLURM_NNODES=$SLURM_NNODES"
echo "DEBUG: SLURM_TASKS_PER_NODE=$SLURM_TASKS_PER_NODE"

# Execute
srun hostname : hostname
srun module_dp-cn.sh : module_dp-dam-ext.sh
```

It uses two execution scripts for loading the correct environment and starting the IMB on the CM and the DAM node (this approach can also be used to start different programs, e.g. one could think of a master and worker use case). The execution scripts could look like:

```
#!/bin/bash
# Script for the CN using InfiniBand

module load Intel ParaStationMPI pscom

# Execution
EXEC=$PWD/mpi-benchmarks/IMB-MPI1
LD_LIBRARY_PATH=/opt/parastation/lib64:$LD_LIBRARY_PATH PSP_OPENIB_HCA=mlx5_0 ${EXEC} PingPong
```

```
#!/bin/bash
# Script for the DAM using Extoll

module load Intel ParaStationMPI pscom extoll

# Execution
EXEC=$PWD/mpi-benchmarks/IMB-MPI1
LD_LIBRARY_PATH=/opt/parastation/lib64:/opt/extoll/x86_64/lib:$LD_LIBRARY_PATH PSP_DEBUG=3 PSP_EXTOLL=1 PSP_VELO=1 PSP_REN
```

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 09/01/2020, the DEEP system has 1 gateway node. In the next weeks at least one additional gateway node will be installed.
- 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.
- The `xenv` utility (necessary on JURECA to load modules for different architectures - Haswell and KNL) is needed on DEEP only to load the `extoll` module on the DAM and ESB nodes (the `extoll` module is not available on the CM. Trying to load it there will produce an error and cause the job to fail). All the other modules can be loaded via the usual `module load` or `ml` command on the batch script before the `srun` command. If desired, `xenv` can still be used to load different set of modules for different sub-jobs of a heterogeneous jobs.

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
```

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     16   idle deeper-sdv[01-16]
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
dam             up   20:00:00      1  down* protodam01
dam             up   20:00:00      3   idle protodam[02-04]
extoll          up   20:00:00     16   idle deeper-sdv[01-16]
ml-gpu          up   20:00:00      1   idle ml-gpu01
dp-cn           up   20:00:00      1  drain dp-cn49
dp-cn           up   20:00:00      2  alloc dp-cn[01,50]
dp-cn           up   20:00:00     47   idle dp-cn[02-48]
```

```
dp-dam          up   20:00:00      1 drain* dp-dam01
dp-dam          up   20:00:00      1  drain dp-dam02
dp-dam          up   20:00:00     14   down dp-dam[03-16]
dp-sdv-esb      up   20:00:00      2   idle dp-sdv-esb[01-02]
psgw-cluster    up   20:00:00      1  down* nfgw01
psgw-booster    up   20:00:00      1  down* nfgw02
debug           up   20:00:00      1 drain* dp-dam01
debug           up   20:00:00      1  down* protodam01
debug           up   20:00:00      3  drain dp-cn49,dp-dam02,knl01
debug           up   20:00:00     14   down dp-dam[03-16]
debug           up   20:00:00      2  alloc dp-cn[01,50]
debug           up   20:00:00     69   idle deeper-sdv[01-16],dp-cn[02-48],knl[04-06],protodam[02-04]
```

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
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????
? Machine (191GB total)
?
? ????????????????????????????????????????????????????????????????????????????  ????????????????????????????????????????????
? ? ????????????????????????????????????????????????????????????????????????? ?  ? ????????????????????????????????????????????
? ? ? NUMANode P#0 (95GB)                                                      ? ?  ? ? NUMANode P#1 (96GB)
? ? ????????????????????????????????????????????????????????????????????????? ?  ? ????????????????????????????????????????????
? ?                                                                              ?  ?
? ? ????????????????????????????????????????????????????????????????????????? ?  ? ????????????????????????????????????????????
? ? ? Package P#0                                                              ? ?  ? ? Package P#1
? ? ?                                                                          ? ?  ? ?
? ? ? ???????????????  ???????????????  ???????????????  ??????????????? ? ?  ? ? ???????????????  ???????????????  ??????
? ? ? ? Core P#0    ?  ? Core P#1    ?  ? Core P#2    ?  ? Core P#3    ? ? ?  ? ? ? Core P#0    ?  ? Core P#3    ?  ? Core
? ? ? ?             ?  ?             ?  ?             ?  ?             ? ? ?  ? ? ?             ?  ?             ?  ?
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ? PU P#0  ? ?  ? ? PU P#1  ? ?  ? ? PU P#2  ? ?  ? ? PU P#3  ? ? ?  ? ? ? ? PU P#12 ? ?  ? ? PU P#13 ? ?  ? ? PU
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ? PU P#24 ? ?  ? ? PU P#25 ? ?  ? ? PU P#26 ? ?  ? ? PU P#27 ? ? ?  ? ? ? ? PU P#36 ? ?  ? ? PU P#37 ? ?  ? ? PU
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ???????????????  ???????????????  ???????????????  ??????????????? ? ?  ? ? ???????????????  ???????????????  ??????
? ? ?                                                                          ? ?  ? ?
? ? ? ???????????????  ???????????????  ???????????????  ??????????????? ? ?  ? ? ???????????????  ???????????????  ??????
? ? ? ? Core P#4    ?  ? Core P#9    ?  ? Core P#10   ?  ? Core P#16   ? ? ?  ? ? ? Core P#9    ?  ? Core P#10   ?  ? Core
? ? ? ?             ?  ?             ?  ?             ?  ?             ? ? ?  ? ? ?             ?  ?             ?  ?
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
```

```
? ? ? ? ? PU P#4  ? ?  ? ? PU P#5  ? ?  ? ? PU P#6  ? ?  ? ? PU P#7  ? ? ? ?  ? ? ? ? PU P#16 ? ?  ? ? PU P#17 ? ?  ? ? PU
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ? PU P#28 ? ?  ? ? PU P#29 ? ?  ? ? PU P#30 ? ?  ? ? PU P#31 ? ? ? ?  ? ? ? ? PU P#40 ? ?  ? ? PU P#41 ? ?  ? ? PU
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ???????????????  ???????????????  ???????????????  ???????????????? ? ?  ? ? ???????????????  ???????????????  ??????
? ? ?                                                                  ? ?  ? ?
? ? ? ???????????????  ???????????????  ???????????????  ???????????????? ? ?  ? ? ???????????????  ???????????????  ??????
? ? ? ? Core P#18   ?  ? Core P#19   ?  ? Core P#25   ?  ? Core P#26   ? ? ?  ? ? ? Core P#17   ?  ? Core P#18   ?  ? Core
? ? ? ?             ?  ?             ?  ?             ?  ?             ? ? ?  ? ? ?             ?  ?             ?  ?
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ? PU P#8  ? ?  ? ? PU P#9  ? ?  ? ? PU P#10 ? ?  ? ? PU P#11 ? ? ? ?  ? ? ? ? PU P#20 ? ?  ? ? PU P#21 ? ?  ? ? PU
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ? ? PU P#32 ? ?  ? ? PU P#33 ? ?  ? ? PU P#34 ? ?  ? ? PU P#35 ? ? ? ?  ? ? ? ? PU P#44 ? ?  ? ? PU P#45 ? ?  ? ? PU
? ? ? ? ??????????? ?  ? ??????????? ?  ? ??????????? ?  ? ??????????? ? ? ?  ? ? ? ??????????? ?  ? ??????????? ?  ? ????
? ? ? ???????????????  ???????????????  ???????????????  ???????????????? ? ?  ? ? ???????????????  ???????????????  ??????
? ? ??????????????????????????????????????????????????????????????????????????????????????? ?  ? ??????????????????????????????????????????????
? ???????????????????????????????????????????????????????????????????????????????????????  ????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????
? Host: dp-cn50
?
? Indexes: physical
?
? Date: Thu 21 Nov 2019 15:22:31 CET
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????
```

The `PU P#X` are the Processing Units numbers exposed by the OS.

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.