Slurm Basics
Slurm documentation

- **NHR@FAU**
  - General: [https://hpc.fau.de/systems-services/systems-documentation-instructions/batch-processing/](https://hpc.fau.de/systems-services/systems-documentation-instructions/batch-processing/)
  - Cluster-specific: [https://hpc.fau.de/systems-services/systems-documentation-instructions/clusters/](https://hpc.fau.de/systems-services/systems-documentation-instructions/clusters/)

- **Official Slurm documentation**
  - Separate documentation for every command and the available options: [https://slurm.schedmd.com/man_index.html](https://slurm.schedmd.com/man_index.html)
  - Slurm commands and their counterparts in different batch systems: [https://slurm.schedmd.com/rosetta.pdf](https://slurm.schedmd.com/rosetta.pdf)
  - Slurm tutorials: [https://slurm.schedmd.com/tutorials.html](https://slurm.schedmd.com/tutorials.html)
Terminology

- **Job**: allocation of resources assigned to a user for a specified amount of time
- **Partition**: set of nodes grouped by specific property (e.g. hardware); can have constraints on job size, time limit, permitted users, etc. → queues
- **Task**: how many instances of your command are executed; normally corresponds to number of MPI processes
- **Jobstep**: set of tasks within a job; a job can contain multiple job steps executing sequentially or in parallel
- **QoS (Quality-of-Service)**: limits set on a per-group-basis (walltime, #GPUs, running jobs per group,...)
- **GRES**: generic resources, here: GPUs
- **CPU**: equivalent to hyperthread if configured; otherwise equivalent to core
Ways to get a job allocation

- **sbatch**: submit a job script for later execution; script will contain (**srun**) commands to execute jobsteps

- **salloc**: allocate resources in real time and spawn a shell when resources are available → interactive job

- **srun**: initiate a job step (run an application) in real time, either interactively or within a job script; if not issued within an allocation, a new allocation will be created automatically → interactive job

On TinyGPU and TinyFat: use command-wraper for all commands, e.g. `sbatch.tinygpu/sbatch.tinyfat, alloc.tinygpu/salloc.tinyfat, ...`
Ways to get a job allocation

$ sbatch [options] jobscript

$ salloc [options]

- c | --cpus-per-task  Number of logical CPUs (hardware threads) per task
---gres               Request nodes with e.g. GPUs
-J | --job-name         Name of job
--mail-user           Mail address for notifications
--mail-type           When to send mail notifications (BEGIN, END, FAIL, ALL)
-N | --nodes            Number of compute nodes
-n | --ntasks           Number of tasks (MPI processes)
--ntasks-per-node     Number of tasks per node
-p | --partition        Partition to be used for job
-t | --time             Max. wall-clock time for job
Job script - general structure

#!/bin/bash -l
#
#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --job-name=myJob
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./application [options]
#!/bin/bash -l
#
#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --job-name=myJob
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./application [options]
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --job-name=myJob
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./application [options]
Job script - general structure

```bash
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --job-name=myJob
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./application [options]
```
Job script - general structure

#!/bin/bash -l
#
#SBATCH --nodes=2
#SBATCH --ntasks=20
#SBATCH --time=01:00:00
#SBATCH --job-name=myJob
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./application [options]

Load necessary modules

Execute parallel application with srun
Environment export

- Environment of submitting shell is by default propagated via `sbatch/salloc` to job
  - Includes all loaded modules and other environment settings from frontend!
  - can lead to unexpected behavior of jobs/applications, which is difficult to reproduce
  - use `sbatch` option `--export=none` to prevent export

- **Caveat**: environment of job script (e.g. loaded modules) has to be propagated to jobstep
  - use `export SLURM_EXPORT_ENV=ALL` inside job script to enable export again

- Currently only available for `sbatch`, not for interactive jobs via `salloc`!
  - take care what modules and environment are loaded in your submitting shell!
  - `module purge`
Managing jobs

- **squeue**: information about jobs in scheduling queue (only your own jobs)
- **sinfo**: reports the state of partitions and nodes
- **scancel**: cancel a pending or running job
- **sattach**: attach standard input, output, and error plus signal capabilities to a currently running job
- **scontrol**: mostly administrator tool, but can be used as a user for e.g. `scontrol show job=<jobId>`
- **sacct**: report job accounting information about active and completed jobs of user
- **sstat**: get information about resource utilization of **running** jobs
Converting PBS/Torque to Slurm

qsub \rightarrow sbatch
qsub -I \rightarrow salloc
qstat \rightarrow squeue
qstat -f JOBID \rightarrow scontrol show job=JOBID
qdel \rightarrow scancel

$PBS_O_WORKDIR \rightarrow $SLURM_SUBMIT_DIR
$PBS_JOBID \rightarrow $SLURM_JOB_ID
cat $PBS_NODEFILE \rightarrow scontrol show hostnames $SLURM_JOB_NODELIST

Running jobs - Best practices and examples
Best practices

- Use interactive jobs for debugging/testing.
- Use batch jobs with job scripts for production work.
- Use `#SBATCH` in the jobscript instead of specifying `sbatch` options on command line for better reproducibility.
- Be as concise as possible with resource allocation and do not over-specify.
- Even for exclusive nodes or automatically allocated cores, you have to specify `--ntasks`, otherwise `ntasks=1` by default.
Node sharing

- Some clusters (TinyX, Alex) allow sharing of nodes among jobs; GPUs are always exclusive to one job.
- User processes are confined to the respective resources via cgroups.
- Performance may be impacted due to shared infrastructure of node (network, SSD,...).

- TinyGPU/Alex: share is based on number of GPUs that are requested; respective share of host CPUs/memory is allocated automatically.
- TinyFat: either request amount of main memory (\(--\text{mem}=\ldots\) in MB) or number of CPUs; share of other resource is allocated automatically.

- See cluster documentation for amount of resources per GPU!
Interactive jobs

- Use for interactive debugging or testing of your application

$ salloc --nodes=1 --time=01:00:00

$ salloc.tinygpu --gres=gpu:1 --time=01:00:00

- When resources are available, this will open an interactive Shell on first node of the allocation.
- Start application/jobsteps with `srun` and corresponding options; also a subset of allocated resources can be used.
- Job will be canceled when interactive shell is closed/disconnected.

Number of concurrent interactive jobs and/or runtime of interactive jobs may be limited on some clusters.
MPI jobs

- We recommend using `srun` and not `mpirun` to start the parallel application!

- Two ways to request resources:
  - `--ntasks`: works well if you want a multiple of the available cores per node (→ full nodes)
  - `--nodes` and `--ntasks-per-node`: Slurm will set `--ntasks` automatically if not specified; can also be used to only partially allocate nodes

Do not use `--ntasks-per-socket`: unpredictable behavior.

Do not use `--ntasks-per-board`: not functional.
MPI jobs - Job script

```bash
#!/bin/bash -l
#
#SBATCH --ntasks=80   # 4 full nodes on meggie
#SBATCH --time=08:00:00
#SBATCH --job-name=TestJobMPI
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./mpi_application [options]
```
MPI jobs - Job script

```bash
#!/bin/bash -l
#
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=72 # full node on fritz
#SBATCH --time=08:00:00
#SBATCH --job-name=TestJobMPI
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./mpi_application [options]
```
MPI jobs - \texttt{mpirun}

- For pure MPI jobs, \texttt{mpirun} usually works without problem.
- Slurm instructs \texttt{mpirun} about number of processes and node hostnames for both IntelMPI and OpenMPI.

Do NOT add options like \texttt{-n <number_of_processes>} or any other option defining the number of processes or nodes to \texttt{mpirun}!

This will mess with the automatic affinity settings of the processes!
MPI jobs - Process placement

- Optimal placement of MPI processes is dependent on the application. For optimal performance, you might need to adjust the automatic binding.
- Automatic binding behavior can differ between type of MPI (IntelMPI vs. OpenMPI), version of the MPI library and Slurm version.
- Resulting distribution of processes may differ between `srun` and `mpirun`.

To check process binding use
  - `--cpu-bind=verbose` for `srun`
  - `--report-bindings` for OpenMPI-`mpirun`
  - `export I_MPI_DEBUG=5` for IntelMPI-`mpirun`

Further information: [https://hpc-wiki.info/hpc/Binding/Pinning](https://hpc-wiki.info/hpc/Binding/Pinning)
Two cases have to be distinguished:

- Full nodes: all available cores are used by jobstep
  → Process binding is done correctly and automatically by `srun` and `mpirun`

- Partially-used nodes: some (automatically) allocated cores are not used by jobstep
  → Process binding is not done automatically by `srun`
  → Use `srun --cpu-bind=cores ./mpi_application`
Shared-memory jobs

- Slurm is not OpenMP aware → $OMP_NUM_THREADS$ must be set manually

- For correct resource allocation in Slurm, use --cpus-per-task to define the number of OMP threads

- If your application does not use OpenMP but other shared-memory parallelization, please consult the application manual on how to specify number of threads.
Shared memory/OpenMP jobs - Thread pinning

- Slurm will not pin (OpenMP) threads! This has to be done manually, e.g. by setting
  
  $OMP_PLACES=cores, $OMP_PROC_BIND=spread

- This is a solid starting point, but optimal pinning always depends on the application! It is good practice to perform some test runs and scaling tests!

Further information: [https://hpc-wiki.info/hpc/Binding/Pinning](https://hpc-wiki.info/hpc/Binding/Pinning)
Shared-memory/OpenMP jobs

```bash
#!/bin/bash

#SBATCH --nodes=1          # always single-node jobs
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64  # full node of TinyFat
#SBATCH --time=04:00:00
#SBATCH --job-name=TestJobMPI
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

module load <modules>

export OMP_PLACES=...; export OMP_PROC_BIND=...
./openmp_application
```
Hybrid MPI/OpenMP jobs

- Combination of shared-memory and MPI jobs as discussed previously:
  - Specify `--cpus-per-task`; set `$OMP_NUM_THREADS` to this value
  - Pinning of threads is necessary and not done automatically
  - Correct binding of MPI processes is especially important for hybrid applications

- Binding of MPI processes needs some manual intervention:
  - `srun`: option `--cpu-bind=cores` necessary!
  - OpenMPI with `mpirun`: automatic process binding is not correct for hybrid case!
    - Use options `--map-by socket:PE=${OMP_NUM_THREADS}` and `--bind-to core`

- Nodes with SMT enabled (TinyFat) use hyperthreads by default for hybrid/shared-memory applications; use `#SBATCH --hint=nomultithread` to prevent this
Hybrid MPI/OpenMP jobs

```bash
#!/bin/bash -l
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=10
#SBATCH --time=04:00:00
#SBATCH --job-name=TestJobHybrid
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

module load <modules>

export OMP_PLACES=... ; export OMP_PROC_BIND=...

srun --cpu-bind=cores ./hybrid_application
```
GPU jobs

- Previously discussed resource specifications are also applicable for GPU jobs
- Amount of host resources is determined by requested number of GPUs
- Share of host resources per GPU cannot be exceeded
- `--ntasks/--cpus-per-task` still have to be requested! Per default `ntasks=1`

How to request GPUs?

- `--gres=gpu:<number>` type is not important (only on clusters with `work/any` partition)
- `--gres=gpu:<type>:<number>` request specific type
- `--gres=gpu:<type>:<number> --partition=<type>` for V100/A100/A40 GPUs
#!/bin/bash

#SBATCH --ntasks=16
#SBATCH --time=06:00:00
#SBATCH --gres=gpu:a40:1
#SBATCH --partition=a40
#SBATCH --export=NONE

export SLURM_EXPORT_ENV=ALL

module load <modules>

srun ./mpi_cuda_application
Advanced usage and other tips
Why is my job not running?

- High workload on the cluster → check node status via `sinfo`
- You are running into a group/user resource limit → look at “Reason” in `squeue` output (`Resources`, `AssocGrpNodeLimit`, `AssocGrpGRES`, …)
- You and/or your group have used many resources over the last days and your fairshare is low → `sshare -1` (below 0.5 if usage > allocated share)

What to do?

- **Be patient!** When there is a high load on the cluster, it might simply take a few hours for your job to start.
- Check via `scontrol show job=JOBID` when your job is probably scheduled to start (no guarantee!).
- Check job priority → `sprio` (depends on time spend in queue/age, fairshare and job size)
Monitor your jobs

You can connect to nodes when job is running to check it interactively:

- **CPU-only jobs:**
  
  ```sh
  $ srn --jobid=<jobID> --overlap --pty /bin/bash -l
  ```

- **GPU jobs:**
  
  - Check on which node job is running with `squeue`.
  - `ssh <nodename>`
  - In case you have more than one job an a node, you will end up in the allocation with the most recently started jobstep; this currently cannot be changed.
  - (This should be obsolete with Slurm v22.05 and the above `srn` command should work for all jobs.)

- **ClusterCockpit:** [https://monitoring.nhr.fau.de/](https://monitoring.nhr.fau.de/) (currently only emmy, meggie, woody & fritz)
How to group work together

- Many jobs that only differ by some index → Array jobs
  - Jobs are differentiable by $SLURM_ARRAY_TASK_ID
  - Submit with #SBATCH --array=1-10

- Run several jobsteps in parallel
  - Every srun must only use subset of allocated resources, defined via options
  - Total requested resources of job can not be exceeded
    
    ```
    #SBATCH --ntasks=3
    srun -n 2 ./application1 &
    srun -n 1 ./application2 &
    wait
    ```

- Run several jobsteps sequentially via srun

- srun --multi-prog (see srun man page)
Job dependencies

- Can be useful for long-running sequences of jobs.
- Jobs will be set on hold until specified dependencies are satisfied.

```
#SBATCH -d <type>:<jobID>[::<jobID>]
```

Available types:

- **after**: job can begin execution after the specified jobs have begun execution.
- **afterany**: job can begin execution after the specified jobs have terminated.
- **afternotok**: job can begin execution after the specified jobs have terminated in some failed state (non-zero exit code, node failure, timed out, etc).
- **afterok**: job can begin execution after the specified jobs have successfully finished (zero exit code).
THANK YOU.

NHR@FAU

https://hpc.fau.de