Batch system

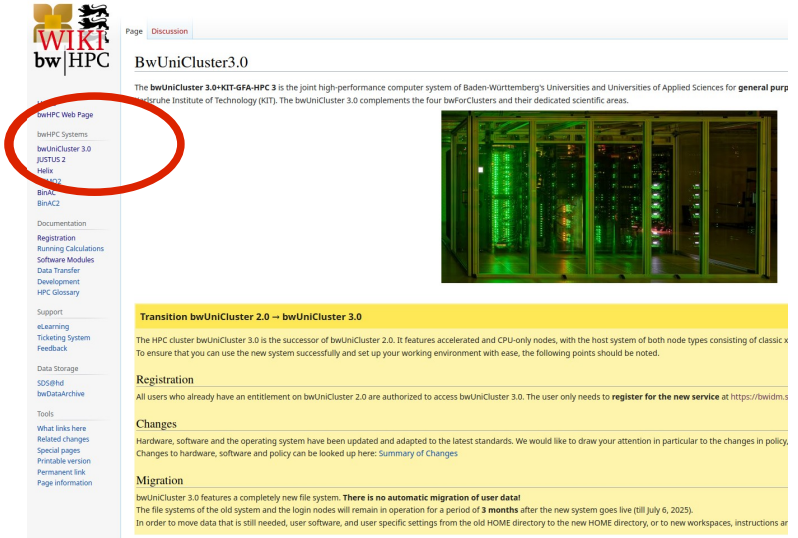
- Best Practices & Running parallel jobs -

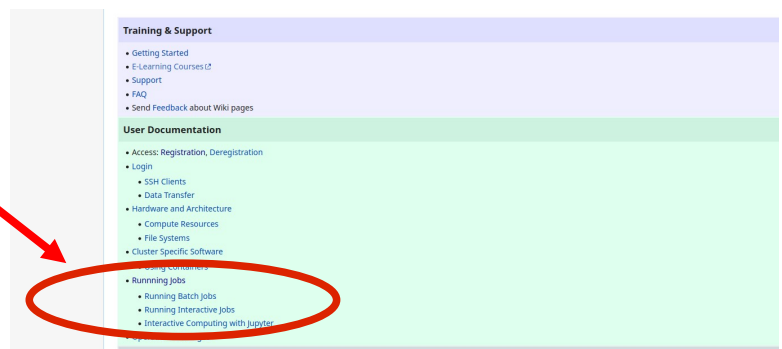
Andreas Baer, KIT, SCC

Baden-Württemberg
Ministerium für Wissenschaft,
Forschung und Kunst

Reference: bwHPC Wiki

- Most information given by this talk can be found at https://wiki.bwhpc.de
  - Select cluster
  - Select “Batch System” or “Running Batch Jobs”



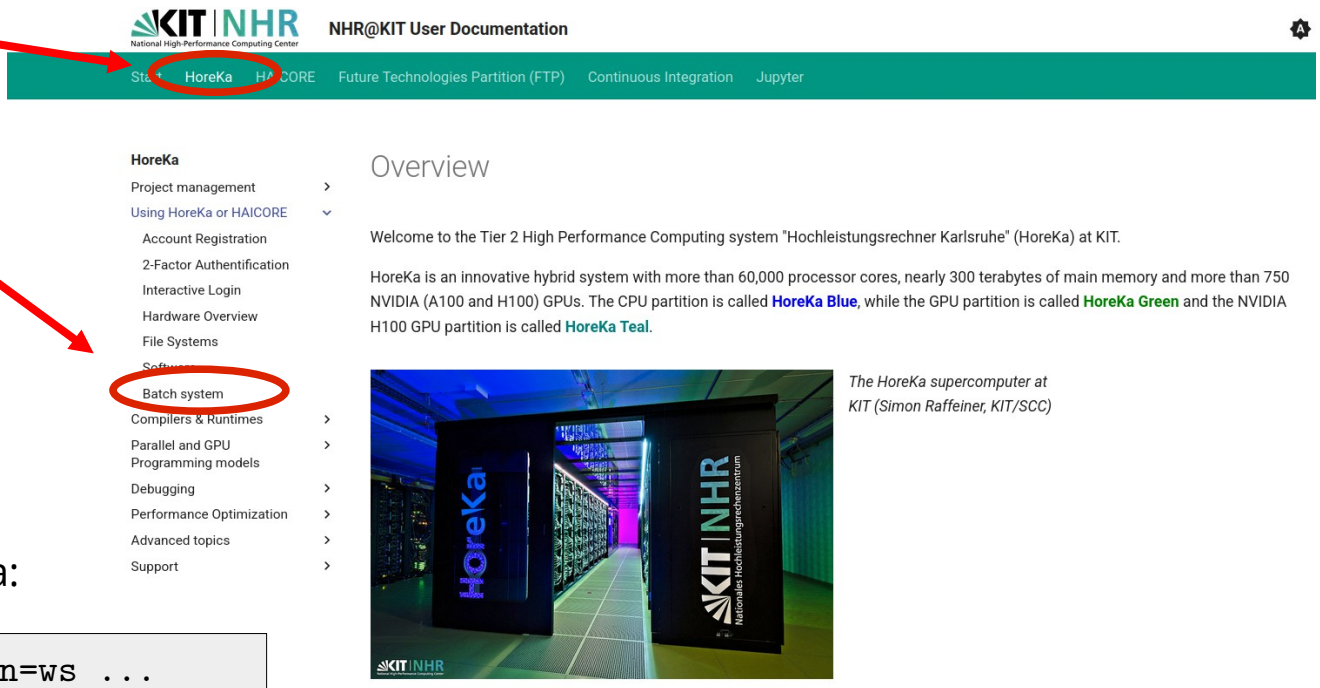


- Workshop reservation bwUniCluster3.0:

```
sbatch -p cpu --reservation=ws ...
```

Reference: NHR@KIT User Documentation

- Most information given by this talk
  can be found at https://www.nhr.kit.edu/userdocs
  - Select cluster
  - Select
    "Using HoreKa or Haicore"
    → "Batch System"



- Workshop reservation HoreKa:

```
sbatch -p cpuonly --reservation=ws ...
```

Material: Slides & Scripts

- https://indico.kit.edu/event/5464/
- BwUniCluster 3.0: /opt/bwhpc/common/workshops/2026-04-15/
- HoreKa: /software/all/workshop/2026-04-15/

How to read the following slides

Abbreviation	Full meaning
`$ command -option value`	$ = prompt of the interactive shell The full prompt may look like: `user@machine:path$` The command has been entered in the interactive shell session
`<integer>` `<string>`	<> = Placeholder for integer, string etc
`foo, bar`	Metasyntactic variables

Outline

- Best Practices
- Parallel jobs
  - OpenMP
  - MPI
  - Hybrid (MPI + OpenMP)

Best practices

Best practices for job submission

- SLURM long options for the batch script, short options for the command line
- Set default values in the script, overwrite in command line if needed
- Pass arguments to batch job on submission via command line: `$ sbatch job.sh -x argument`
- When using shared nodes:
  - Only requested resources available (e.g. memory; defaults apply though)
  - SSD storage is shared **without** resource control!
- Environment and data paths
  - Always ensure a well-defined environment

```
module purge
module load foo bar
```

```
#SBATCH --export=NONE # do not export env
```

  - Change to absolute path, Slurm submit directory or similar

Job script templates

- Many software packages provide help description, example cases or job templates
- How to get it? => Search in description for example directory, example Turbomole

```
$ module show chem/turbomole 2>&1 | grep "EXA_DIR"
/opt/bwhpc/common/chem/turbomole/TmoleX2025/bwhpc-examples
```

bwHPC_turbomole_single-node_tmpdir_example.sh

```
#!/bin/bash

## Purpose: Turbomole JOB example script for bwHPC, such as
bw{For,Uni}Cluster
##          for   S I N G L E   N O D E   runs   O N L Y
...
```

Job monitoring

- Do NOT run script that submits **every second** commands like:
  - `squeue`
  - `scontrol show job <JOB_ID>`
  - `tail -f  <Global_file_system>/<file>`
    - Change to „`tail -f -s 10"` etc.

- How to follow **live** the job progress on compute node?
  - `srun --jobid=<JOB_ID> [--overlap] --pty /usr/bin/bash`
  - `htop`
  - monitor local storage

Job monitoring – job report

- Check out resource usage for optimization
  - Wallclock time
  - CPU
  - Memory
  - Energy
- Further monitoring
  => Talk later

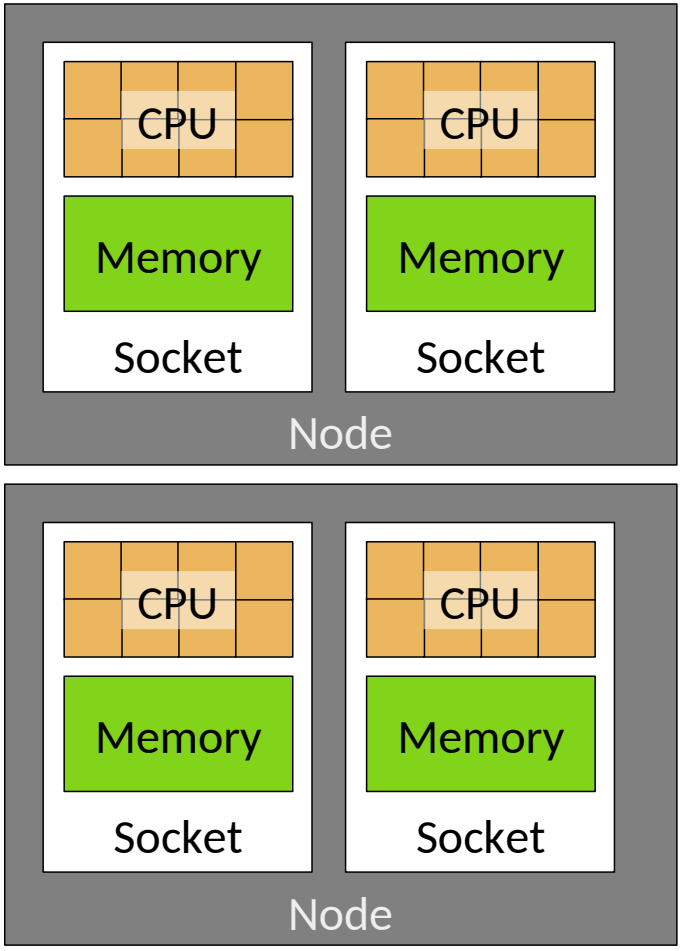
```
============================== JOB FEEDBACK ==============

Job ID: 23482310
Cluster: hk
User/Group: ab1234/my-project
Account: my-project
State: COMPLETED (exit code 0)
Partition: cpuonly
Nodes: 4
Cores per node: 152
Nodelist: hkn[0201-0202,1603-1604]
CPU Utilized: 00:21:29
CPU Efficiency: 3.59% of 09:57:52 core-walltime
Job Wall-clock time: 00:00:59
Starttime: Mon Mar 25 13:35:31 2024
Endtime: Mon Mar 25 13:36:30 2024
Memory Utilized: 25.92 GB
Memory Efficiency: 2.73% of 950.00 GB
Energy Consumed: 68710 Joule / 19.0861111111111 Watthours
Average node power draw: 1164.57627118644 Watt
```

Parallel Jobs

Resource specifications

- Computing resources are clustered in several layers:
  - node = compute server
    (sharing main memory, mostly 2 sockets per node)
  - socket = set of cpu unit (with several cores) + main memory
  - cpu = CPU Core (with two threads each)
  - task = instance of your program
  - thread = sequence for instructions
- Hyperthreading:
  - Use of both threads per core
  - Only with OpenMP, not MPI
  - Only in some cases beneficial => test if it improves performance



How to do the exercises?

- Login to cluster & Generate workspace „bwhpc-course“

```
$ ws_allocate bwhpc-course 30
Workspace created. Duration is 720 hours.
Further extensions available: 3
/pfs/work2/workspace/scratch/xy1234-bwhpc-course-0
```

- Copy examples to your workspace

```
$ WORKSHOP=/opt/bwhpc/common/workshops/2026-04-15   # bwUniCluster
$ WROKSHOP=/software/all/workshops/2026-04-15       # HoreKa
$ cd $(ws_find bwhpc-course)
$ mkdir -v 2026-04-15; cd 2026-04-15
$ cp -pr ${WORKSHOP}/exercises/01 ./
```

- Submit jobs from your workspace

```
$ cd $(ws_find bwhpc-course)/2026-04-15/01
$ sbatch -p {single|cpuonly} --reservation=ws [--exclusive] <jobscript>
```

Compile executables

- We need executables „**pi_omp**”, “**parmmul**” and “**parmmul_omp**”
- Open files “**omp.README**” and “**parmmul.README**” => execute commands in these files
- Hints:
  - You can use `cat` for an easy copy paste

```
$ cat omp.README
module load compiler/intel
...
# pi_omp
# first compile seconds.c
icc -DFTNLINKSUFFIX -O -c seconds.c -o seconds.o
ifort -O -qopenmp pi.f90 seconds.o -o pi_omp
```

  - You can parse the whole file with `bash` to execute all commands subsequently

```
$ bash omp.README
$ bash parmmul.README
```

OpenMP

- OpenMP = Open Multi-Processing
  - Compiler directives, library routines, environment variables to enable multi-threading on **shared-memory** multiprocessor platforms (e.g. on single node)
  - https://www.openmp.org

- Single task is split into multiple threads for parallel execution
  - E.g. `for loop` for a matrix multiplication
- Typical issues:
  - Number of spawned threads should match resources: typically 1 thread per core
  - Proper thread binding and mapping

OpenMP parallel example

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --time=00:05:00


# Set executable name to variable
exe=./pi_omp
# Load modules
module load compiler/intel
# Setup OpenMP environment variable
export OMP_NUM_THREADS=${SLURM_NPROCS}
# Printout number of threads
echo "No.threads = ${OMP_NUM_THREADS}"
# Execute program
${exe}
```

Example:

`${WORKSHOP}/exercises/01/omp.sh`

- Shared memory is restricted to 1 node.
- Use the precompiled pi_omp or store the executable in pi_omp
- Do not define number of threads explicitely. Use environment variables.

OpenMP parallel example: thread pinning

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=10
#SBATCH --time=00:01:00
# Set executable name to variable
exe=./pi_omp
module load compiler/intel
# Setup OpenMP environment variable
export OMP_NUM_THREADS=${SLURM_NPROCS}
# Use different pinning: none, scatter, compact
export KMP_AFFINITY=verbose,scatter
# Printout number of threads
echo "No.threads = ${OMP_NUM_THREADS}"
# Execute program
${exe}
```

Using Intel OpenMP Thread Affinity for Pinning Threads differently

```
${WORKSHOP}/exercises/01/omp_v2.sh
```

- Submit script with different pinnings:
  - *none*
  - *scatter*
  - *compact*
  - *compact,1,0*
- Compare results

MPI

- MPI = Message Passing Interface
  - Open standard for parallelization on **distributed memory** systems, e.g. multiple nodes
  - Program is started by wrapper (e.g. `mpirun`, `srun`, ...), spawning several `tasks` (on multiple nodes)
    - Each task executes the same binary
    - Differences between the tasks execution depend on the rank (task id) and total number of tasks
      => relevant for the programmer
  - Standard on https://www.mpi-forum.org
- Variants on the clusters
  - Intel-MPI
  - OpenMPI

  => different versions, depending on different compilers
- Typical issues: task assignment and binding

MPI process binding

- Compute-bound job
  - One MPI task per available core
- Memory-bound job
  - One/few MPI tasks per socket or node
- Hybrid jobs (MPI + OpenMP)
  - One MPI task per socket or node
  - OpenMP multithreading over the whole socket/node

MPI parallel jobs: compute-bound

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=20
#SBATCH --time=00:05:00

# Set executable name to variable
exe=./parmmul
# Load modules
module purge
module load compiler/intel
module load mpi/openmpi
# Printout number of tasks
echo "No.MPI tasks = ${SLURM_NPROCS}"
# Spawn for each core 1 MPI task
mpirun --bind-to core --report-bindings ${exe}
```

Example:

```
${WORKSHOP}/exercises/01/openmpi.sh
```

- Use parallel multiplication binary
- Load Intel compiler module and OpenMPI module
- Use mpirun to execute the binary, print details of task pinning.

MPI parallel jobs: memory-bound

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --time=00:05:00
# Set executable name to variable
exe=./parmmul
# Load modules
module purge
module load compiler/intel
module load mpi/openmpi
# Printout number of MPI tasks
echo "No.MPI tasks = ${SLURM_NTASKS}"
# Spawn only one MPI task per socket
mpirun --bind-to core --map-by socket \
       --report=bindings ${exe}
```

Example:

`${WORKSHOP}/exercises/04/openmpi_v2.sh`

- 2 tasks per node
  → each task can consume 45 GB (120 GB on HoreKa)
- Loading the needed environment
- Map each MPI task to one socket

MPI + OpenMP hybrid job

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=20
#SBATCH --time=00:05:00
exe=./parmmul_omp
module load compiler/intel mpi/openmpi
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export KMP_AFFINITY=verbose,scatter
# Printout number of nodes = MPI tasks
echo "No. MPI tasks (nodes) = ${SLURM_NTASKS}"
echo "No. threads per node  = ${OMP_NUM_THREADS}"
# Spawn only one MPI task per socket
mpirun -n ${SLURM_NTASKS} --bind-to core --map-by socket:PE=${OMP_NUM_THREADS} \
       --report-bindings ${exe}
```

`${WORKSHOP}/exercises/04/hybrid_openmpi_omp.sh`

Spawn 1 MPI task per socket and
20 OpenMP threads per MPI task

Set KMP_AFFINITY
to bind and map
threads to cores!

Thank you for your attention.

Questions?