bw|HPC

Batch System – Best Practices

Robert Barthel, SCC, KIT

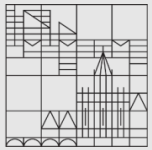
UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

UNI FREIBURG

UNIVERSITÄT HOHENHEIM 1818

Hochschule Esslingen
University of Applied Sciences

Universität Konstanz



UNIVERSITÄT MANNHEIM

Universität Stuttgart

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

KIT Karlsruher Institut für Technologie

ulm university universität uulm

Funding:

Baden-Württemberg
MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST

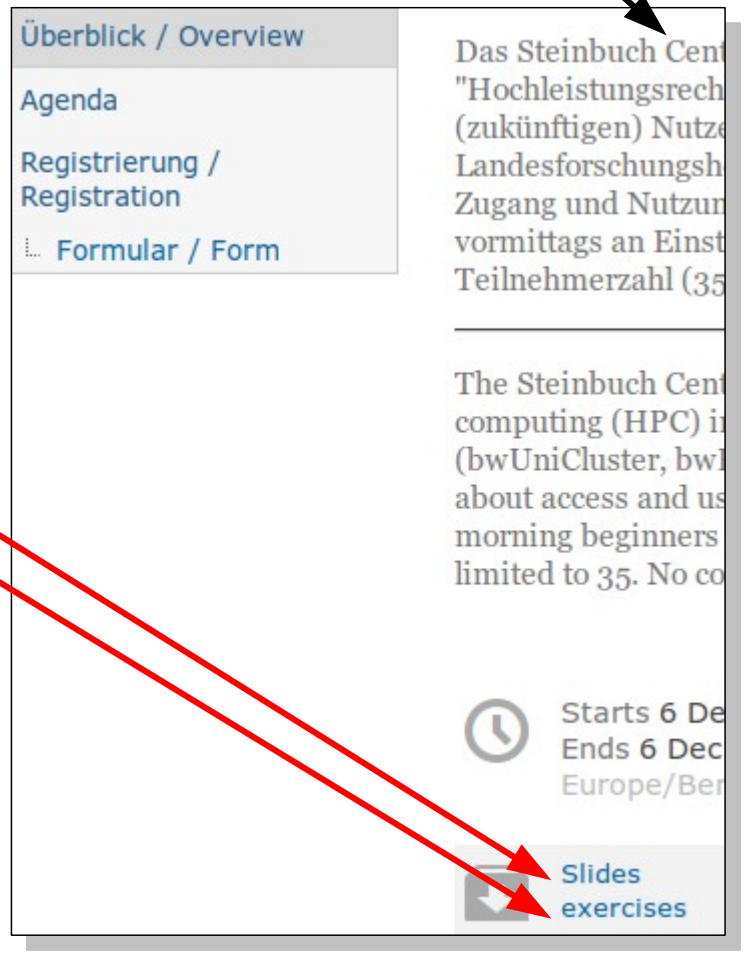
How to read the following slides

Abbreviation/Colour code	Full meaning
`$ command -opt value`	`$` = prompt of the interactive shell The full prompt may look like: `user@machine:path $` The command has been entered in the interactive shell session
`<integer>` `<string>`	<> = Placeholder for integer, string etc
`foo, bar`	Metasyntactic variables
`${WORKSHOP}`	`/pfs/data1/software_uc1/bwhpc/kit/workshop/2019-04-10`

- MOAB: more info can be found at http://bwhpc.de/wiki in article:
  - Batch_Jobs
- SLURM: more info can be found at http://wiki.scc.kit.edu/hpc in article:
  - Batch_Jobs

Where to get the slides/exercises/reservation?

- https://indico.scc.kit.edu/e/bwhpc_course_2019-04-10 or
  bwUniCluster: /pfs/data1/software_uc1/bwhpc/kit/workshop/2019-04-10
  - Slides
  - Exercises



- Workshop reservation:
  single node:

```
msub -A workshop
-l advres=bwhpc-workshop_single.97
```

- multi node:

```
msub -A workshop
-l advres=bwhpc-workshop_multi.98
```

How to do excerices?

- 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=/pfs/data1/software_uc1/bwhpc/kit/workshop/2019-04-10
$ cd $(ws_find bwhpc-course)
$ mkdir -v 2019-04-10; cd 2019-04-10
$ cp -pr ${WORKSHOP}/exercises/04 ./
```

- Submit jobs from your workspace

```
$ cd $(ws_find bwhpc-course)/2019-04-10/04
$ msub <jobscript>
```

Job Preparation

Msub Directives (1)

- Write msub defaults in your script, overwrite time interactively via CLI, e.g.: **msub -N newname <script>**

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -q develop
#MSUB -N serial-test
#MSUB -m bea
#MSUB -M my_email_address

## to access today's standing reservation
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop.75

printenv
```

Msub Directives (2)

- Use alternative directive prefixes to store multiple „defaults“ in your run script and use them via: **msub -C '#Prefix'**
  - Pitfall 1: undefined msub options in #Prefix will be filled with defined #MSUB options

```
...
## Default
#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -N default
#MSUB -v runtype=1
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_single.89

## Alternative msub directives, to use via: msub -C '#MALT'
#MALT -l nodes=1:ppn=1
#MALT -l walltime=00:02:00
#MALT -l mem=200mb
#MALT -N alternative
#MALT -v runtype=2

if [[ ${runtype:-0} -eq 1 ]] ; then
   echo "Run default"; printenv
else
   echo "Run alternative"; printenv
fi
```

${WORKSHOP}/exercises/04/01_msubdirectives.sh

Attention!!!
#MALT uses from #MSUB:
-A workshop & -l advres=...

Sbatch Directives

- Write SBATCH defaults in your script, overwrite time interactively via CLI, e.g.: **sbatch -J newname <script>**

```
#!/bin/bash

#SBATCH -N 1
#SBATCH -t 00:01:00
#SBATCH --mem=50mb
#SBATCH -p singlenode
#SBATCH -J serial-test
#SBATCH --mail-type=BEGIN,FAIL,END
#SBATCH --mail-user=<my_email_address>

## to access today's standing reservation
#SBATCH -A workshop
#SBATCH --reservation=<name>

printenv
```

Ressource specifications (1)

- @ MOAB
  - Node

    = computer server
  - Proc

    = „processors“ → Cores
  - *Task*

    = atomic collection of resources, such as processors, memory, or local disk, which must be found on the same node

Examples:

1) Addressing 2 cores on 1 node:

```
msub -l nodes=1:ppn=2
```

2) two cores on nodes 1,2 + four cores on node 3:

```
msub -l nodes=2:ppn=2+1:ppn=4
```

- @ OpenMP
  - Thread

    = smallest sequence of instructions managed independently by a OS scheduler

    → Normally 1 thread is pinned to 1 core
- @ MPI
  - Task

    = is 1 process excuted by the OS

    → smallest useful unit: 1 MPI task per node

Ressource specifications (2)

- @ MOAB
  - **mem**

    = Working memory for total job
  - **pmem**

    = Working memory per cores

Example:

Addressing 4GB for 2cores on 1 node:

```
msub -l nodes=1:ppn=2,pmem=2gb
or
msub -l nodes=1:ppn=2,mem=4gb
```

- Node access policies

  bwUniCluster: **shared** *(with other users' jobs)*
  how to get whole node? → e.g.:

  ```
  msub -l nodes=1:ppn=16   or
  msub -l naccesspolicy=singlejob
  ```

  bwForClusters: shared, **singleuser** *(=shared with other own jobs),*
  singlejob *(=exclusively for your job)*

Ressource specifications (3)

- @ SLURM (ForHLR I/II)
  - `node` = computer server
  - `task` = number of „instances“ of your command to be executed, default: max tasks per node is equal max cores (exception: overcommit)
  - `cpu` = *CPU Cores*Threads* (e.g. hyperthreading)
  - `thread` = number of processing units on each core (not number of application tasks launched per core; default: on per *CPU core*

Examples:

1) Addressing 2 CPU cores (@1 task) on 1 node:

```
#SBATCH --nodes=1 --ntasks=1 --cpus-per-task=2
```

2) 40 Tasks on 1 node:

```
#SBATCH -N1 --ntasks=40 --overcommit
```

Ressource specifications (2)

- @ SLURM (ForHLR I/II)
  - `--mem` = Memory per node (Check defaults: „scontrol show config“)
  - `--mem-per-cpu` = Minimum memory per allocated cpu,
    if exceeding MaxMemPerCPU, do workaround with mem,
    ntasks and cpus-per-task
  - Node access policies

    ForHLR I/II: **singlejob** *(=exclusively for your job)*

Job Script: Templates

- Use templates for job scripts
  - Provided by many installed software packages
    - cf. help description of SW

Example:

- How to get? Search in description for example directory, or e.g. Turbomole

```
$ module show chem/turbomole 2>&1 | grep "EXA_DIR"
→ shows path:
/opt/bwhpc/common/chem/turbomole/7.2.1_tmolex432/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
...
...
```

Best Practises – Job setup (1)

- Directory:
  - Running your code/application/job in ${HOME} is not permitted

    Valid destinations are:

    1. Workspaces (`ws_allocate`)
    2. $TMPDIR (not suitable for multinode jobs)

  - Issues:
    - Workspaces:
      - Does not handle well codes producing Tbyte of scratch files and more then 10000 files. Solution: Change your application code, Apply for Tiger Team Support.
    - $TMPDIR:
      - Requires job script setup to handle data transfer

```
#!/bin/bash
#MSUB ...
cp -pr ${MOAB_SUBMITDIR}/<file> ${TMPDIR}
...
cp -pr ${TMPDIR}/<results>* ${MOAB_SUBMITDIR}
```

Best Practises – Job setup (2)

- Directory: (cont.)
  - Potential problems:
    - During job run, binaries/inputfiles etc not found

      → give full path to binaries/inputfiles

      → change DIR in jobscript

```
#!/bin/bash
#MSUB ...
# Change to job submission directory
cd ${MOAB_SUBMITDIR}
```

- Record resource usage to optimise resource requests
  - Walltime:

```
#!/bin/bash
#MSUB ...

# Record runtime of executed program
SECONDS=0
time ./program
...
echo $SECONDS
```

Job submission

Job Submit

- MOAB: Not allowed:

```
$ msub job.sh -x argument
```

→ msub will interprete -x as an own option

- SLURM: allowed:

```
$ sbatch job.sh -x argument
```

→ sbatch assumes that everything after stating the script name are user's arguments

- Solution:

(A) Submit wrapper script:

```
#!/bin/bash
your_script -x argument
```

(B) Export your script options and arguments to environment variable; read in that variable during runtime of script, cf. wiki

```
if [ -n "${SCRIPT_FLAGS}" ] ; then
   if [ -z "${*}"  ] ; then
      set -- ${SCRIPT_FLAGS}
   fi
fi
```

(C) Use msub wrapper via:

```
$ module load system/msub_addon/1.0
$ msub <options> job.sh
```

Job execution

MOAB: Best Practices – Job „Observation“

- Do NOT run script that submits every second commands like:
  - **`checkjob`**
  - **`showq -n`**
  - **`tail -f  <Global_file_system>/<file>`**
    - Change to „**`tail -f -s 10"`** etc.

- How to follow **live** the job progress on compute node?
  - **`ssh`** on given compute node does
    - **NOT** working on bwUniCluster
    - Work on bwForCluster JUSTUS
    - ...

  → cf. bwHPC Wiki

Parallel Jobs

OpenMP parallel jobs (1)

- OpenMP = Open Multi-Processing)

  → compiler directives, lib routines, environment variables to enable multithreading on shared-memory multiprocessor platforms

  - https://www.openmp.org

- Coures:
  - @KIT: OpenMP (10.07.19) as part of lecture „Parallelrechner & Parallelprogrammierung“
  - @Uni Stuttgart: Oct 14-18 2019, https://www.hlrs.de/training/2019/PAR

- Typical issues:
  - Number of threads not matching given ressources

    → Normally 1 thread is to be pinned to 1 core
  - Process binding

OpenMP parallel jobs (2)

- Example: `${WORKSHOP}/exercises/04/parallel/omp.sh`

```
#!/bin/bash
#MSUB -l nodes=1:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_single.97

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

- Shared memory is restricted to 1 node.
- Do not define number of threads explicitely. Use environment variables.

OpenMP parallel jobs: Pinning

- Using Intel OpenMP Thread Affinity for Pinning Threads differently

${WORKSHOP}/exercises/04/parallel/omp_v2.sh

```
#!/bin/bash
#MSUB -l nodes=1:ppn=4
#MSUB -l walltime=00:01:00
#MSUM -l pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_single.97

# Set executable name to variable
exe=./hello_omp
# Load modules
module load compiler/intel

# Setup OpenMP environment variable
export OMP_NUM_THREADS=${MOAB_PROCCOUNT}
# Use different pinning: none, scatter, compact
export KMP_AFFINITY=verbose,scatter

# Printout number of threads
echo "No.threads = ${OMP_NUM_THREADS}"

# Execute program
${exe}
```

TASK/ToDo: 5 min

- Submit script with different pinnings
- Compare results

MPI parallel jobs (1)

- MPI = Message Passing Interface
  - → To enable programs parallelly running on a distributed memory system
  - MPI tutorial from Livermore Computing Center (https://computing.llnl.gov/tutorials/mpi/)
  - MPI Standards on http://www.mpi-forum.org
- Variants @ Clusters
  - Intel-MPI (impi → modules: mpi/impi/<version>)
  - OpenMPI (openmpi → modules: mpi/openmpi/<version>)
  - Dependencies?
    - **Versions depend on compilers!**
- Coures:
  - @KIT: OpenMP (10.07.19) as part of lecture „Parallelrechner & Parallelprogrammierung“
  - @Uni Stuttgart: Oct 14-18 2019, https://www.hlrs.de/training/2019/PAR

MPI process binding

- Compute-bound MPI
  - As many MPI tasks per node as cores available
- Memory-bound MPI
  - One MPI task per socket/node
- Hybrid MPI + OpenMP
  - One MPI task per „domain“, multithreaded process over the whole domain

Compute-bound: Intel-MPI parallel jobs

- Example: `${WORKSHOP}/exercises/04/parallel/impi.sh`

```
#!/bin/bash
#MSUB -l nodes=1:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_single.97

# Set executable name to variable
exe=./hello_impi
# Load modules
module load mpi/impi/2017-intel-17.0

# Printout number of tasks
echo "No.MPI tasks = ${MOAB_PROCCOUNT}"
# Spawn for each core 1 MPI task
mpirun -print-rank-map ${exe}
```

- For computations on more than 1 node use **bwhpc-workshop_multi.94**
- For testing example, copy the executable to submit directory
- Adds automatically the corresponding Intel compiler module
- Use mpirun to execute the binary, print details of task pinning.

Compute-bound: OpenMPI parallel jobs

- Difference to OpenMPI?

  → compare: `${WORKSHOP}/exercises/04/parallel/openmpi.sh`

```
#!/bin/bash

#MSUB -l nodes=1:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_single.97

# Set executable name to variable
exe=./hello_openmpi
# Load modules
module load mpi/openmpi/3.1-intel-17.0

# Printout number of tasks
echo "No.MPI tasks = ${MOAB_PROCCOUNT}"

# Spawn for each core 1 MPI task
mpirun -display-map ${exe}
```

Memory-bound: Intel-MPI parallel jobs

- Spawning **only 1 MPI task per node**
  - Pitfalls: a) `mpirun -n ${MOAB_NODECOUNT}` b) `mpirun -perhost 1` c) ...

```
#!/bin/bash
#MSUB -l nodes=2:ppn=28
#MSUB -l walltime=00:01:00
#MSUM -l pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_multi.98
# Set executable name to variable
exe=./hello_impi
# Load modules
module load compiler/intel mpi/impi


# Printout number of nodes = MPI tasks
echo "No.MPI tasks = ${MOAB_NODECOUNT}"
# Spawn only one MPI task per node
export I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0
mpirun -perhost 1 -binding domain=node -print-rank-map ${exe}
```

`${WORKSHOP}/exercises/04/parallel/impi_v2.sh`

- Preset of MOAB must be overwriten by: `I_MPI_JOB….=0`
- Assign to each MPI task the full node as its domain

Memory-bound: OpenMPI parallel jobs

- Spawning only 1 MPI task per node
  - Difference to Intel-MPI?

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

```
#!/bin/bash

#MSUB -l nodes=2:ppn=28
#MSUB -l walltime=00:01:00
#MSUM -l pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_multi.98

# Set executable name to variable
exe=./hello_openmpi
# Load modules
module load mpi/openmpi/3.1-intel-17.0

# Define domain size of each MPI task
domain_size=$((${MOAB_PROCCOUNT}/${MOAB_NODECOUNT}))
# Printout number of MPI tasks and domain size
echo "No.MPI tasks = ${MOAB_NODECOUNT}"
echo "Domain size  = ${domain_size}"

# Spawn only one MPI task per node
mpirun --map-by ppr:1:node:pe=${domain_size} -display-map ${exe}
```

- Autoloads corresponding Intel compiler
- Assign to each MPI task the full node as its domain

Hybrid parallel jobs: Intel-MPI + OpenMP

- Spawning only 1 MPI task per node, and n OpenMP thread per MPI task

${WORKSHOP}/exercises/04/parallel/hybrid_impi_omp.sh

```
#!/bin/bash

#MSUB -l nodes=2:ppn=28
#MSUB -l walltime=00:01:00
#MSUM -l pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_multi.98

# Set executable name to variable
exe=./hello_impi_omp
# Load modules
module load compiler/intel mpi/impi

# Define domain size of each MPI task
domain_size=$((${MOAB_PROCCOUNT}/${MOAB_NODECOUNT}))
# Printout number of MPI tasks and domain size
echo "No.MPI tasks = ${MOAB_NODECOUNT}"
echo "Domain size  = ${domain_size}"

# Spawn only one MPI task per node; set variables
export OMP_NUM_THREADS=${domain_size}
export I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0
mpirun -genvall -perhost 1 -binding domain=node -print-rank-map ${exe}
```

Hybrid parallel jobs: OpenMPI + OpenMP

- Spawning only 1 MPI task per node, and n OpenMP thread per MPI task
  - Difference to Intel-MPI?

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

```
#!/bin/bash
#MSUB -l nodes=2:ppn=28,walltime=00:01:00,pmem=200mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop_multi.98

# Set executable name to variable
exe=./hello_openmpi_omp
# Load modules
module load compiler/intel mpi/openmpi/3.1-intel-17.0



# Setup OpenMP env variable
export OMP_NUM_THREADS=$((${MOAB_PROCCOUNT}/${MOAB_NODECOUNT}))
# Printout number of nodes = MPI tasks
echo "No. MPI tasks (nodes) = ${MOAB_NODECOUNT}"
echo "No. threads per node  = ${OMP_NUM_THREADS}"
# Spawn only one MPI task per node
mpirun --map-by ppr:1:node:pe=${OMP_NUM_THREADS} -report-bindings ${exe}
```

Thank you for your attention!

Questions?