## Submit Jobs (jsub)

(Last update: Jul 11, 2025)

You can submit your jobs with "jsub" command. Input files (and jobscript) should be placed on ccfep beforehand using scp or sftp etc. (For file transfer using scp and sftp, please check quick start guide page.)

For Gaussian jobs, there are special commands g16sub and g09sub. Please consider g16sub/g09sub first, although jsub can be used.

For ORCA, we have also prepared a dedicated command osub. (Registration is required to use ORCA installed in /apl/orca.)


- Basic Usage
- Prepare Jobscript
    - sample jobs for applications installed by RCCS => Sample Jobs
    - jobscript header samples => Tips about job submission
    - 1 - Choose Interpreter (shebang; required)
    - 2 - Request Resources (select; required)
    - 3 - Max Walltime of Job (walltime; required)
    - 4 - Optional Parameters
- Job Dependencies (-W depend=afterok:(job ID), -W depend=afterany:(job ID))
- Stepwise Executioin of Jobs (step jobs; --step or --stepany)
- Specify Values of Variables upon Job Submission (-v VARNAME=VALUE)
- Wait until Job Termination (-W block=true)
- Special Usage
    - Need Large Memory But not CPU Cores
    - Do Many Calculations in Single Job


### Basic Usage

You need to prepare a job script (named "job.sh" here; the detail is described below) first. You can submit a jobs using "jsub" command and "job.sh" file as follows. (Don't type leading $.) If the job is successfully accepted by the job server, job ID and server name will be shown.

```
$ jsub job.sh
6169323.ccpbs1
$
```

The "jsub" command itself finishes immediately. However, submitted job is not always executed immediately. If there are no available resources (CPU or GPU), the job needs to wait for a while. Submitted jobs remain in the system even when you log out.

The displayed number (6169323 in the example above) is the ID of the job (job ID). This is used when you delete the job. However, it is not necessary to always remember the displayed job ID. This is because job IDs can be

listed with "jobinfo" command after the job submission.

Although you don't need to specify queue name, you can specify the queue name like "jsub -q H job.sh".

Optional parameters of "jsub" command can be listed with "jsub --help" command. Please use "jobinfo" command to check the status of submitted jobs. If you want to delete or cancel jobs, please use "jdel" command.

## Prepare Jobscript

For applications installed by RCCS, there are samples of jobscript and input files. For the standard location of those files in RCCS, please check Sample Jobs page. These sample jobscripts can be used as a template file for your own jobs.

We here show an example jobscript. This request 64 CPU cores, 32 MPI processes and 2 OpenMP threads. The maximum wall time of this job is 24 hours. This will run "my-program" installed in the home directory. "my-program" uses Open MPI 4.1.6 installed by RCCS. (NOTE: annotation part beginning with  in the example shouldn't be present in the actual job script file. It will cause errors.)

```
#!/bin/sh   ? 1 Choose Interpreter (shebang)
#PBS -l select=1:ncpus=64:mpiprocs=32:ompthreads=2  ? 2 Request Resources
#PBS -l walltime=24:00:00  ? 3 Max Walltime of Job
                    ? 4 Optional Parameters can be added here

# chdir to the working directory when you submit job (recommended; possible to skip)
# When job starts on computation node, the working directory is usually your home directory.
# If you skip this operation, you may have trouble with input and output file paths for example.
cd ${PBS_O_WORKDIR}

# do module setting if necessary
module -s purge
module -s load openmpi/4.1.6

# you can set parameters for your application here
export PATH="/home/users/${USER}/bin:$PATH"

INPUT=myinp.inp
OUTPUT=myout.out

# run program
mpirun -np 32 my-program -i ${INPUT} -o ${OUTPUT}
```

1 - Choose Interpreter (shebang; required)

Choose interpreter of the script (shebang). Normally, one of the following is used (corresponding to sh/bash, csh/tcsh, or zsh).

- #!/bin/sh
- #!/bin/csh -f
- #!/bin/zsh

This interpreter definition is necessary.

2 - Request Resources (select; required)

Queue information and queue factors (CPU points definition) can be found at this page.
Some samples of this section are also available at this page.

Numbers of CPU cores, MPI processes, OpenMP threads, and GPUs are specified here ("#PBS -l" at the beginning of the line is also necessary). Basic format is summarized as follows.

```
#PBS -l select=(# of nodes):ncpus=(# of CPU cores):mpiprocs=(# of MPI processes):ompthread
```

For number of nodes

- If ncpus=1-63 and no GPUs (jobtype=core), the number of nodes is always 1.
- If ncpus=64 or 128 (jobtype=vnode), the number of nodes is 1 or more. The total number of CPU cores is (# of nodes)x(# of CPU cores; ncpus).
- If ngpus >= 1 (jobtype=gpu), the number of nodes is 1 or more. The total number of CPU cores is (# of nodes)x(# of CPU cores; ncpus). The total number of GPUs is (# of nodes)x(# of GPUs; ngpus).

For information other than the number of nodes, these numbers are per node values. The value of mpiprocs is related to machine file (host file) or MPI. The value of ompthreads corresponds to "OMP_NUM_THREADS" environment variable.

- ncpus= specifies number of CPU cores per node. Valid values are 1-64 and 128.
- mpiprocs= specifies number of MPI processes per node. Usually mpiprocs * ompthreads is equal to ncpus.
- ompthreads= specifies number of OpenMP threads per process. Usually mpiprocs * ompthreads is equal to ncpus. For non-OpenMP thread parallelism, this value is simply ignored.
- The available memory amount is proportional to ncpus= value. For jobtype=largemem, available memory amount is 7.875 GB/core. For other types, that is 1.875 GB/core. Please also check queue information page.

Depending on how the MPI library is built, it may not properly respect machine file (host file) provided by the queuing system. In that case, you need to manually give the host file (defined in PBS_NODEFILE environment variable) to mpirun or mpiexec command. MPI libraries installed by RCCS can implicitly respect the host file provided by the queuing system.

- Number of GPUs is specified with ngpus= keyword, where the number CPU cores per GPU (ncpus/ngpus) must be less than or equal to 16. The valid values of ngpus is 1-8, since one TypeG node has 8 GPUs.

```
# 32 CPU cores and 2 GPUs example
#PBS -l select=1:ncpus=32:mpiprocs=2:ompthreads=16:ngpus=2
```

- You need to add jobtype=largemem to use TypeF node which has 1 TB/node memory. It is to be noted that there are only 14 TypeF nodes in total. Please try to use the regular TypeC nodes as much as possible. TypeF nodes are not equipped with GPUs. Therefore, ncpus is 64 or 128 and ngpus is not available in the case of jobtype=largemem.

```
# jobtype=largemem sample (2 nodes, 256 CPU cores in total)
#PBS -l select=2:ncpus=128:mpiprocs=64:ompthreads=2:jobtype=largemem
```

3 - Max Walltime of Job (walltime; required)

The format of wall time is:

```
#PBS -l walltime=(hours):(minutes):(seconds)
```

If you write "walltime=30:00", this is considered 30 minutes, not 30 hours. Please beware.

CPU points of jobs are not calculated from this walltime specification. CPU points are calculated from job duration time (elapsed time). Therefore, you can request a somewhat longer time. However, please do not specify a too long time. Very long jobs sometimes are disadvantaged especially during busy times.

If the requested time is longer than the remaining time until the next maintenance, that job won't run until the end of the next maintenance (jsub would issue a warning). In other words, those long jobs will begin to run immediately after the maintenance.

4 - Optional Parameters

Following optional parameters are available.

```
#PBS -m abe
```

Mail notice: email will be sent you when the job begins(a), ends(e), and aborts(a) for the "PBS -m abe" case. No mails will be sent if "#PBS -m n" is specified. The default is to send only when abort (#PBS -m a). You can use "-m ae" instead of "-m abe" to send mail when the job ends (normally or abnormally). Note: please don't add "b" or "e" when you submit many jobs.

```
#PBS -m n
```

```
#PBS -r n
```

Suppress restarting: when the job crashes due to the system trouble, the queuing system tries to restart the job from the beginning. You can suppress this restart mechanism by adding this line in the jobscript. This is useful when the job is not able to restart (due to the temporary files for example). Note: CPU points are not consumed for aborted run. If the job is restarted after the aborting, CPU points of that job are calculated from the elapsed time of restarted run.

```
#PBS -j oe
```

Merge outputs: normally, stdout and stderr of the job are saved in different files. If you add this option, stderr output is merged into stdout file. (If you add "-j eo", stdout output is merged into stderr.)

```
#PBS -N (arbitrary job name)
```

Job name: you can add name to that job. For example, you can add name "myjob1" by adding "#PBS -N myjob1". Command line option "-N" of jsub has the same functionality. If you don't add the name, the file

name of the jobscript will be the name of the job.

For example, following header specification is possible. Send email at the beginning and end of the job execution. This won't restart when the job crashes due to the system trouble. Stderr output is merged into stdout file and the job name is "myhob1".

```
#!/bin/sh
#PBS -l select=1:ncpus=64:mpiprocs=32:ompthreads=2
#PBS -l walltime=24:00:00
#PBS -m be
#PBS -N myhob1
#PBS -j oe

cd ${PBS_O_WORKDIR}

module -s purge
module -s load openmpi/4.1.6

export PATH=/home/users/${USER}/bin

mpirun -np 32 -npernode 32 -x OMP_NUM_THREADS=2 -x PATH -x LD_LIBRARY_PATH ./a.out
```

Job Dependencies (-W depend=afterok, -W depend=afterany)

Please also check stepwise jobs below. Stepwise jobs are easier to use than this.

Stepwise Execution of Jobs (step jobs; --step or --stepany)

You can add dependencies on jobs. For example, a job defined in "job2.sh" file would run only if job 61693323 finishes without error. (If 61693323 returns error, job2.sh won't be executed.)

You can run jobs stepwise manner by adding "--step" or "--stepany".

```
$ jsub -W depend=afterok:61693323 job2.sh
```

If you want job2.sh to run regardless of the exit status of 61693323, please use "afterany" keyword instead.

```
$ jsub -W depend=afterany:61693323 job2.sh
```

Specify Values of Variables upon Job Submission (-v VARNAME=VALUE)

You can define values of variables with -v command line option of jsub. The following example, with the lines INPUT=myinp.inp and OUTPUT=myout.out removed from the original sample, illustrates the operation.

```
#PBS -v INPUT=myinp.inp,OUTPUT=myout.out
```

In the following example, you can run job1.sh, job2.sh, job3.sh, and job4.sh sequentially. If one of the jobs return an error, following jobs won't run. ("afterok" dependency is internally used.)

```
$ jsub --step job1.sh job2.sh job3.sh job4.sh
```

Job submission finishes immediately after submitting a job. If you add "-W block=true", jsub waits until the job termination. (-W block=true)

```
#PBS -W block=true
```

## Special Usage

Need Large Memory But not CPU Cores

The available memory amount is proportional to number of requested CPU cores on RCCS system. If you need a single CPU core and 180 GB of memory for the computation, you need to request 32 CPU cores (approximately 60 GB of memory available). (In terms of CPU points, points are calculated from requested number of cores.) 32 cores job on this system. However, there are some limitations for such job. This is because existence of too many jobs place a heavy load on the queuing system. You may need to merge jobs into single job. For example, instead of submitting 32 jobs (or more) job submitted a long series of step jobs. (e.g. jsub --step -m n (job1) (job2) ... (jobN))

In such case, job script like below is efficient and safe.

If you need more memory, please consider to use jobtype=largemem nodes. (You can choose from

```
#!/bin/sh
#PBS -l select=1:ncpus=32:mpiprocs=1:ompthreads=1
#PBS -l walltime=24:00:00

cd ${PBS_O_WORKDIR}

sh job1.sh &
sh job2.sh &
sh job3.sh &
... (skipped)
sh job31.sh &
sh job32.sh &

wait # wait until all the background jobs finish
```

The last line, wait, is the key points of this script. You can wait for the termination of the 32 background processes by wait command. (If "wait" does not exist, the job script finishes immediately and the job itself is

also terminated. Running background processes are killed by the system even when they are still running.) The "wait" command here is the builtin function of bash, csh/tcsh also has "wait" command just like this.

You can also use ncpus=64 or ncpus=128 instead of 32 in the example above. If you want to run thousands or more jobs, please use ncpus=64 or 128 (jobtype=vnode). (Note: it is tedious to use multiple nodes in this way.)