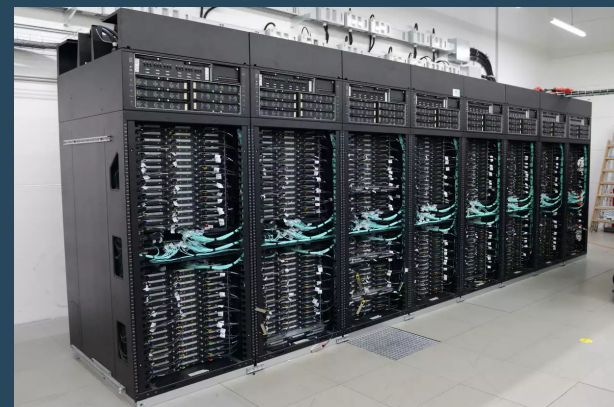
FRITZ-HABER-INSTITUT
MAX-PLANCK-GESELLSCHAFT

VIPER SUPERCOMPUTER AT MPCDF

FIRST STEPS ON THE SLURM CLUSTER



Simeon Beinlich
beinlich@fhi.mpg.de
PP&B Computer Support Group
Fritz-Haber-Institut der Max-Planck-Gesellschaft
Image: mpcdf.mpg.de

PREFACE

The Viper Supercomputer is a shared resource between all Max Planck Institutes provided by the Max Planck Computing and Data Facility (MPCDF).

The FHI heavily relies on these resources and on a good relation to the MPCDF. Members of the FHI essentially use the resources of the FHI Theory department by Prof. Dr. Karsten Reuter.

Please use these resources responsibly and adhere to the rules specified in the wiki FHI Viper Wiki. (in addition to the Usage conditions that you accepted during registration for an MPCDF account)

PREFACE

The Viper Supercomputer is a shared resource between all Max Planck Institutes provided by the Max Planck Computing and Data Facility (MPCDF).

The FHI heavily relies on these resources and on a good relation to the MPCDF. Members of the FHI essentially use the resources of the FHI Theory department by Prof. Dr. Karsten Reuter.

Please use these resources responsibly and adhere to the rules specified in the wiki FHI Viper Wiki.
(in addition to the Usage conditions that you accepted during registration for an MPCDF account)

Accounts of users that do not adhere to these rules will be closed.

If you are in doubt about anything, please contact us before submitting any jobs.

PREFACE

The Viper Supercomputer is a shared resource between all Max Planck Institutes provided by the Max Planck Computing and Data Facility (MPCDF).

The FHI heavily relies on these resources and on a good relation to the MPCDF. Members of the FHI essentially use the resources of the FHI Theory department by Prof. Dr. Karsten Reuter.

Please use these resources responsibly and adhere to the rules specified in the wiki FHI Viper Wiki.
(in addition to the Usage conditions that you accepted during registration for an MPCDF account)

Accounts of users that do not adhere to these rules will be closed.

If you are in doubt about anything, please contact us before submitting any jobs.

Good News: We will set up everything individually *with you!*
Your Software, your job script, your parallelization, your file directory, etc. and we will make sure that your are good to go!

PREFACE

The Viper Supercomputer is a shared resource between all Max Planck Institutes provided by the Max Planck Computing and Data Facility (MPCDF).

The FHI heavily relies on these resources and on a good relation to the MPCDF. Members of the FHI essentially use the resources of the FHI Theory department by Prof. Dr. Karsten Reuter.

Please use these resources responsibly and adhere to the rules specified in the wiki FHI Viper Wiki.
(in addition to the Usage conditions that you accepted during registration for an MPCDF account)

Accounts of users that do not adhere to these rules will be closed.

If you are in doubt about anything, please contact us before submitting any jobs.

Good News: We will set up everything individually *with you!*
Your Software, your job script, your parallelization, your file directory, etc. and we will make sure that your are good to go!

PS: *Please keep an eye on your cpuh consumption here: FHI Compute Monitor.*
As a rule of thumb: 100k cpuh per month is fine (orange lines).

REQUIREMENTS, REGISTRATION & ACCESS

REQUIREMENTS

- Accessible for all members of the FHI
- Requires prior consultation (PP&B) required:
  - Use case (software)
  - Required computation time
  - Basic introduction
- **Adherence to Viper Usage Rules: FHI Viper Wiki | Usage**

Computing time limits per user *in used CPU hours (cpuh)*

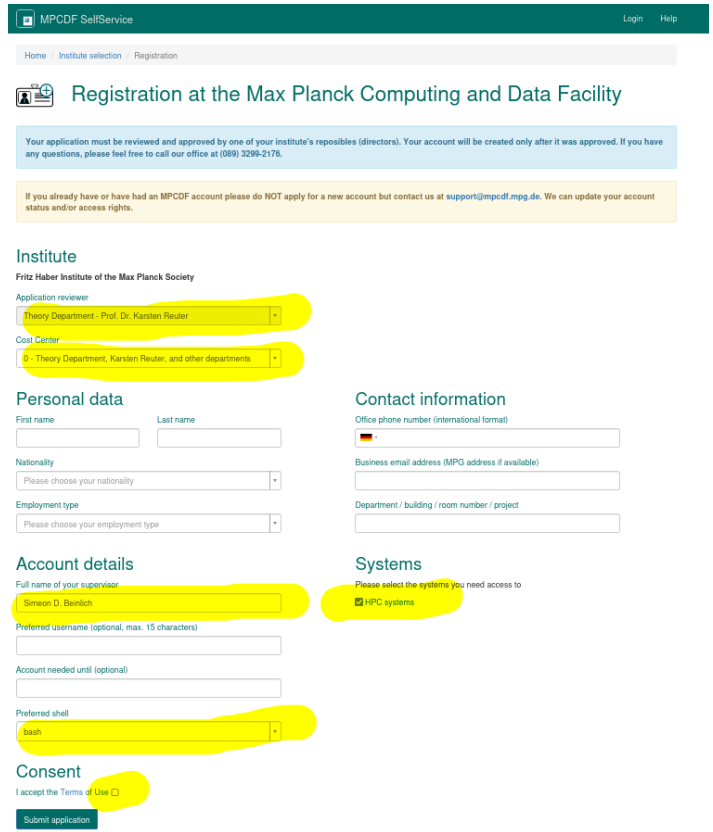
- **Soft limit per user:** 100k cpuh / month ; 1.2M cpuh / year
- **Hard limits are discussed for each individual use case**

REGISTRATION

- Usage bound to MPCDF account
- Separate registration required (PPB 'acts as supervisor'):
  FHI Viper Wiki | Account Creation

ACCESS

- Terminal via SSH only (Password + OTP), see FHI Viper Wiki | Access



SCHEDULED SYSTEMS VS DIRECT EXECUTION

LAPTOP / NORMAL COMPUTER (DIRECT EXECUTION)

Just run your software with your input file.

CLUSTER / SUPERCOMPUTER (SCHEDULED)

Create a TODO for the supercomputer and it will at some point do it.

SCHEDULED SYSTEMS VS DIRECT EXECUTION

LAPTOP / NORMAL COMPUTER (DIRECT EXECUTION)

Just run your software with your input file.

CLUSTER / SUPERCOMPUTER (SCHEDULED)

Create a TODO for the supercomputer and it will at some point do it.

The supercomputer keeps an internal TODO-list and will use its resources most efficient to run all TODOs as fast as possible.

We have to tell the supercomputer:

- What to do (as a shell script)
- How much resources the TODO needs: number of cores, memory, ... (as a special header in the shell script)
- Which software (and which prerequisites) we need

In addition:

- Upload all files required for the job (inputfile, pseudopotentials, ...)
- Download all results (job files on the supercomputer get deleted after some time).

EXAMPLE JOB SCRIPT (TODO FILE) - GAUSSIAN

```
#!/bin/bash -l

#### Specify the resources / meta information #####
# The Job Name:
#SBATCH -J test_gaussian
#
# Number of MPI Tasks, e.g. 16:
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16   # <--------- change this for more/less cores
# Memory usage [MB] of the job is required, e.g. 4000 MB per task:
#SBATCH --mem=64000          # <--------- change this for more/less memory (max. 4GB/core)
#

# Standard output and standard error:
#SBATCH -o ./job.out.%j
#SBATCH -e ./job.err.%j

# Initial working directory: (./ == where we submit the job)
#SBATCH -D ./

# Wall clock limit (max. is 24 hours):
#SBATCH --time=24:00:00      # <--------- change this to ~150% expeceted runtime
```

EXAMPLE JOB SCRIPT (TODO FILE) - GAUSSIAN

```
#### Define required variables ####
# Define your own variables and/or the variables required by your software
export INPUT_FILE=test.gjf
export OUTPUT_FILE=test.out
export g16root="/u/$(whoami)/Software/Gaussian/G16/C0.1_with_gcc-14.0"
export GAUSS_SCRDIR="/ptmp/$(whoami)/g16-scratch"
export PROGRAM=/u/$(whoami)/Software/Gaussian/G16/C0.1_with_gcc-14.0/g16/g16
```

EXAMPLE JOB SCRIPT (TODO FILE) - GAUSSIAN

```
#### Other required Steps ####
# Do some other required steps (e.g. create required directories)
echo $GAUSS_SCRDIR
mkdir $GAUSS_SCRDIR -p

#### Load the Modules ####
# Load the required modules (prerequisites, like libraries etc) and environment files
module load gcc/14
source $g16root/g16/bsd/g16.profile

#### Execute the program: ####
echo "Starting `${PROG_NAME} in $`( pwd ) on `date`"
`${PROGRAM} < $`{INPUT_FILE} > ${OUTPUT_FILE}

# Done!
```

A SHORT NOTICE ON PARALLELIZATION

```
...
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16   # <--------- change this for more/less cores
# Memory usage [MB] of the job is required, e.g. 4000 MB per task:
#SBATCH --mem=64000          # <--------- change this for more/less memory (max. 4GB/core)
...
/some/path/myprogram < input_file > output_file
```

MULTITHREADING (OPENMP):

- One process, many threads (e.g. 16): `--ntasks=1 --cpus-per-task=16 --mem=64000`

MULTIPROCESSING (MPI)

- Many processes (e.g. 16), each one thread `--ntasks=16 --cpus-per-task=1 --mem=64000`

HYBRID (MPI + OPENMP)

- Many Processes, many threads (e.g. 4 processes, 4 threads) `--ntasks=4 --cpus-per-task=4 --mem=64000`

The parallelization depends on your software, we will find out together which is required!

SHARED-, SINGLE-, & MULTI-NODE JOBS

```
...
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16   # <--------- change this for more/less cores
# Memory usage [MB] of the job is required, e.g. 4000 MB per task:
#SBATCH --mem=64000          # <--------- change this for more/less memory (max. 4GB/core)
...

/some/path/myprogram < input_file > output_file
```

SHARED-NODE

- Use less than a full node (e.g. only 16 out of 128 cores) `--ntasks=1 --cpus-per-task=16 --mem=64000`

SINGLE-NODE

- Use a full node (128 cores), add: `--nodes=1 --ntasks=128 --cpus-per-task=1`
- Start the program, e.g., an MPI program, with `srun ...`

MULTI-NODE

- Many Nodes (e.g. 5x128 cores), add: `--nodes=5 --tasks-per-node=128 --cpus-per-task=1`
- Start the program, e.g., an MPI program, with `srun ...`

Again, this depends on your software (and your input), we will find out together what makes sense!

EXAMPLE MULTI-NODE QUANTUM ESPRESSO JOB

```
#!/bin/bash

#SBATCH --job-name=my-job
#SBATCH --no-requeue
#SBATCH --nodes=5
#SBATCH --ntasks-per-node=128
#SBATCH --time=24:00:00
#SBATCH --output=std.out
#SBATCH --error=std.err

...

srun /some/path/bin/pw.x -in inputfile > outputfile
```

This will use 5x128 = 640 cores!

Multi-node processing requires to tweak your software accordingly, e.g. in QE:

- parallelize over images (efficient)
- parallelize over k-points (efficient)
- parallelize over tasks (inefficient)
- ...

Finding out what makes sense often requires benchmarking - we will do that with you if required!

EXAMPLE JOB SCRIPT (TODO FILE) - QUANTUM ESPRESSO

```
#!/bin/bash

#SBATCH --job-name=my-job
#SBATCH --no-requeue
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --time=24:00:00
#SBATCH --output=std.out
#SBATCH --error=std.err

QE_root="/u/sbeinlich/Software/QE/qe-7.3.1/intel_standard"
inputfile="inputfile.in"
outputfile="outputfile.out"

QE_flags=""
#QE_flags="-nk 1"

srun_flags=""
#srun_flags="-n 128"

module purge

unset LD_LIBRARY_PATH

module load intel/2024.0
module load impi/2021.11
module load mkl/2024.0
```

EXAMPLE JOB SCRIPT (TODO FILE) - QUANTUM ESPRESSO

```
source "$MKLROOT/env/vars.sh"

# This should ideally not be needed, but QE had trouble linking these libraries correctly...

export LIBS="-lmkl_intel_lp64 -lmkl_sequential -lmkl_core"

# magic environment variable settings similar to raven (might not be needed but unlikely to do harm)

srun `$srun_flags $`QE_root/bin/pw.x `$QE_flags -in $`inputfile > $outputfile
```

COMMON WORKFLOW

1) LOGIN TO THE SUPERCOMPUTE AND CREATE A JOB FOLDER

2) CREATE AND TRANSFER ALL REQUIRED FILES

via WinSCP, SCP, Rsync, ... **to /ptmp/username/some/folder**

- Input Files, Job Script, ...

3) SUBMIT THE JOB

- Run `sbatch the_job_script.sh`
- Note down the job ID

4) CHECK WHETHER IT IS DONE

- Run `squeue -u <your username>` and look for the state of your job: *PD=pending, R=running*
- If it is not in the queue anymore, it is done - successful or not ;) ...

5) TRANSFER THE RESULTS BACK TO YOUR PC

via WinSCP, SCP, Rsync, ... **to /ptmp/username/some/folder**

- **any relevant file that you want to keep!** (job files on `ptmp` get deleted after ~12 weeks)

PS: *Create a new job folder for very job instead of running multiple jobs within the same folder!*

STORAGE ON VIPER

HOME

`/u/<username>` or `/viper/u/<username>`

- **long-term files**
- Software
- Configurations
- Job scripts
- Important results.
- **No system backups are performed.**

It is not allowed to run jobs here, use PTMP.

PTMP

`/ptmp/<username>` or `/viper/ptmp/<username>`:

- **job files**
- Parallel network storage
  (identical on every compute node)
- No need to copy anything back and forth (unlike Q cluster)
- The size of PTMP is 12 PB
  for **all users of all Max Planck Institutes**.
- **No system backups are performed.**

PTMP is a temporary file system -- Copy any important results to your computer after the job is finished.

- All files that have not been accessed for more than 12 weeks will be removed automatically.
- The period of 12 weeks may be reduced if necessary.

SLURM USAGE

SLURM COMMANDS

the most important ones

- **Submit a job** (note down the jobid!):
  `sbatch some_job.sh`
- **Cancel / Kill a job**:
  `scancel <jobid>`
- **View queuing / running jobs**:
  `squeue -u <your MPCDF username>`
- **View details about your finished job**:
  `sacct -j <jobid>`
- **View your currently queuing jobs and additional information:**
  `squeue --format="%15j %7i %10u %8P %4t %16V %16S %9M %7Q %8l %5D %10R" -u <YOUR USER NAME>`

Slurm is the most common scheduler on scientific HPC systems.

SLURM JOB SCRIPTS

- **General collection of job scripts:**
  MPCDF Viper Slurm Example Batch Scripts.
- **Software-specific jobs scripts:**
  FHI Viper Wiki | Software.
- **A good starting point for a normal MPI job:**
  FHI Viper Wiki | Quantum Espresso.

Remember to start you jobs from PTMP (at least let them run there), and then only copy the relevant results to HOME.

AVAILABLE SOFTWARE

A (rather) up-to-date list can be found here: FHI Viper Wiki | Software.

PREINSTALLED

- ABINIT
- CP2K
- DFTB+
- FHI-aims
- GROMACS
- LAMMPS
- ORCA
- PLUMED
- TURBOMOLE
- VASP
- ...

BASICS

- Python / Anaconda (conda-forge)
- (Tensorflow, pytorch, etc. on GPU partition)
- Matlab / Octave
- ...

CUSTOM

- Gaussian
- Dirac
- Molpro
- Quantum Espresso

COMPILERS

- GNU (GCC, GFORTRAN, G++, ...)
- OPENMPI
- INTEL (+Intel MPI)
- ...

Organized in a huge module tree!

NOT YET THERE?

Anything can be compiled:

- Hardware compatible (x86)
- Unix compatible
  SUSE Linux Enterprise Server

OTHER / ADVANCED TOPICS (DON'TS)

LOGIN NODES

- Use them for setting up software, configs, jobs, and managing your data.

Do NOT run any software or processing on the login nodes!

AUTOMATIC JOB SUBMISSION

- One can use scripts or ready-made frameworks to automatize job submission.

NEVER run automatized job submission scripts without discussing with us.

JOB FARMING

- Instead of submitting 100 small jobs, submit 1 large job that executes each small job.
- Good for trivially parallel tasks.

DEPENDENCY JOBS

- Start a job only after another job has finished
- E.g. if Job-2 requires outputs from Job-1

Do NOT run any Job Farming or Dependency Jobs without discussing with us!

We will figure out with you whether something of it makes sense and how we can do that efficiently!

FURTHER TOPICS & LINKS

MORE TRAINING MATERIAL

Visit root.compute.fhi.mpg.de/training

PPB WIKI | HPC

General Information.

FHI GITLAB | HPC DOCUMENTATION

VIPER Wiki | Jupyterhub Wiki | Virtual Desktop Wiki | Storage Wiki

Questions, issues, needing help?
Please mail: beinlich@fhi.mpg.de
or call: +49 30 8431 ***5270***