VASP Licensing and Compilation Guide for HPC Clusters

Last updated 1 May, 2025 • 2 min read

VASP licensing

Teams that would like to run VASP would need to purchase and manage their own VASP license.

VASP licensing is tied to each team that purchases a license and the license is not allowed to be shared among different teams, unless a team authorizes other users within their designated license.

> ⓘ HPC does not manage the VASP licensing.

After the VASP license is purchased, a team will be granted/assigned the installation files for the version of VASP the license supports.

Teams can place the VASP installation files within their designated team shared directories on HPC.

More information for the licensing is available here: FAQs .

VASP license registration form is available here:

Get a license

Compiling VASP on the cluster

Reading the comments in the VASP makefiles, we can compile VASP with GCC, the Intel compiler, the AMD AOCC compiler, or NVIDIA's NVHPC compiler. We will use the Intel compiler and VASP 6.3.2 in this example:

```
# Create a directory for our VASP project.
mkdir /path/to/shared/directory/location/vasp
cd /path/to/shared/directory/location/vasp
mkdir vasp-6.3.2
cd /path/to/shared/directory/location/vasp/vasp-6.3.2
mkdir buildfiles
cd buildfiles


# Transfer the associated Linux VASP tarball for 6.3.2 to the
# Use any file transfer program or means to transfer the tarb
#FileZilla, WinSCP, a direct copy on HPC using cp or rsync, e

rsync -a --progress vasp.6.3.2.tgz /path/to/shared/directory/

# Unpack the source.
tar -xvf vasp.6.3.2.tgz


# Load the Intel compiler and hdf5/1.13.2-ics version depende
# The libraries can change depending on the compiler being us
# and if dependencies were install locally under a team's sha
#
# List available Intel compiler versions:
module avail intelics
# Get rid of any other modules that might interfere with our

module purge
module load intel/oneapi/2022.3 hdf5/1.13.2-ics

# Compile the VASP program.
cd vasp.6.3.2

# Compile by copying over a pre-set makefile.include file fro
cp arch/makefile.include.intel_omp .

# Edit the makefile.include file that got copied with needed
nano or vi makefile.include.intel_omp


```

Here is an example of the contents that can be used with an Intel compiler build of VASP, settings may change.

```
# Default precompiler options
CPP_OPTIONS = -DHOST=\"LinuxIFC\" \
              -DMPI -DMPI_BLOCK=8000 -Duse_collective \
              -DscaLAPACK \
              -DCACHE_SIZE=4000 \
              -Davoidalloc \
              -Dvasp6 \

```

```
              -Duse_bse_te \
              -Dtbdyn \
              -Dfock_dblbuf

CPP         = fpp -f_com=no -free -w0 -march=core-avx2 $*$(FU

FC          = mpiifort
FCL         = mpiifort -qmkl=sequential

FREE        = -free -names lowercase

FFLAGS      = -assume byterecl -w

OFLAG       = -O2
OFLAG_IN    = $(OFLAG)
DEBUG       = -O0

OBJECTS     = fftmpiw.o fftmpi_map.o fftw3d.o fft3dlib.o /gpf
OBJECTS_O1 += fftw3d.o fftmpi.o fftmpiw.o
OBJECTS_O2 += fft3dlib.o

# For what used to be vasp.5.lib
CPP_LIB     = $(CPP)
FC_LIB      = $(FC)
CC_LIB      = icc
CFLAGS_LIB  = -O
FFLAGS_LIB  = -O1
FREE_LIB    = $(FREE)

OBJECTS_LIB = linpack_double.o

# For the parser library
CXX_PARS    = icpc
LLIBS       = -lstdc++

##
## Customize as of this point! Of course you may change the p
## part of this file as well if you like, but it should rarel
## necessary ...
##

# When compiling on the target machine itself, change this to
# relevant target when cross-compiling for another architectu
VASP_TARGET_CPU ?= -march=core-avx2
FFLAGS     += $(VASP_TARGET_CPU)

# Intel MKL (FFTW, BLAS, LAPACK, and scaLAPACK)
# (Note: for Intel Parallel Studio's MKL use -mkl instead of
FCL        += -qmkl=sequential
MKLROOT    ?= /gpfs/sharedfs1/admin/hpc2.0/apps/intel/oneapi/
LLIBS      += -L$(MKLROOT)/lib/intel64 -lmkl_scalapack_lp64 -
INCS        =-I$(MKLROOT)/include/fftw

# HDF5-support (optional but strongly recommended)
CPP_OPTIONS+= -DVASP_HDF5
HDF5_ROOT  ?= /gpfs/sharedfs1/admin/hpc2.0/apps/hdf5/1.13.2-i
LLIBS      += -L$(HDF5_ROOT)/lib -lhdf5_fortran
INCS       += -I$(HDF5_ROOT)/include
```

```
66
67 # For the VASP-2-Wannier90 interface (optional)
68 #CPP_OPTIONS    += -DVASP2WANNIER90
69 #WANNIER90_ROOT ?= /path/to/your/wannier90/installation
70 #LLIBS          += -L$(WANNIER90_ROOT)/lib -lwannier
```

After the makefile.include file has the build configuration set, save the file.

Submit an interactive SLURM job to a compute node to perform the build:

```
srun -N 1 -n 126 --partition=general --pty bash
```

Wait for a node to assign to the job, once assigned, perform the build by running the **make** command.

Example make command to build with 12 cores and specify to build all of the VASP executables.

```
make -j12 all
```

More information on the make command for the VASP build available on VASP's Wiki page here:

Installing VASP.6.X.X - VASP Wiki

If there are no errors, VASP should build successfully.

After building, if the vasp_std, vasp_gam, and vasp_ncl executables are not available, run the following make command:

```
make install
```

Creating a loadable VASP module on the cluster

Create a module file for VASP that so that we can conveniently load VASP and it's dependencies. The name that you choose for your module file is important as that is what module uses to reference it. We will make our name different by adding the "-mine" suffix to help separate it from the system installed vasp.

```
mkdir -p /path/to/shared/directory/location/mod/vasp
cd /path/to/shared/directory/location/mod/vasp
vi 6.3.2
```

```
#%Module1.0
## vasp modulefile
##
proc ModulesHelp { } {

        puts stderr "\tAdds vasp/6.3.2 to your environment"
}

module-whatis   "Adds vasp/6.3.2 to your environment"


# Throw an error if any of these modules are loaded.
module load pre-module

module load intel/oneapi/2022.3
module load zlib/1.2.12-ics
module load hdf5/1.13.2-ics

conflict vasp

setenv          MOD_APP         vasp
setenv          MOD_VER         6.3.2
set             prefix          /gpfs/sharedfs1/path/to/VASP/
prepend-path    PATH            $prefix/locationWhereVaspBina

module          load post-module
```

If you are interested, in learning about module files you can read `man modulefile`

Finally, make sure that `module` knows to look in your `~/mod` directory for your module files by setting the `MODULEPATH` environmental variable:

```
nano ~/.bashrc  # Add the lines below.
```

```
1 # My modules
2 source /etc/profile.d/modules.sh
3 MODULEPATH=/path/to/shared/directory/location/mod:${MODULEF
4
5
6
7
8
```

Reload your ~/.bashrc file in your current shell:

```
source ~/.bashrc
# Finally Now we can load and run our VASP module
module load vasp/6.3.2
which vasp
vasp -h

```

Using VASP on the cluster

All jobs on the cluster must be submitted through the SLURM scheduler using `sbatch`. Please read the SLURM Guide for more details. The preferred way to run `VASP` jobs is by specifying your associated VASP INCAR file to the vasp executable that a team installs within their team's shared directory. Please note that if your job uses many cores or a lot of memory it is better to submit to reserve a whole compute node (126 cores)

Multithread job

To submit a job that uses 126 computational threads on one node, create a submission script vaspMP.sh:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 126
#SBATCH -p general

module load vaspmodule
export UCX_TLS=tcp,self,sysv,posix
source /gpfs/sharedfs1/admin/hpc2.0/apps/intel/oneapi/2022.3/

mpirun vasp_std <restofcommandhere>


```

Then submit the script by:

```
 sbatch vaspMP.sh

```

GPU job

If you are running on a GPU node, feel free to reduce the cores and allocate resources as needed.

> Do not forget to allocate a GPU card to the job submission by using the #SBATCH --gres=gpu:X SLURM submission header

You can replace the X above for a GPU job submission with the number of GPU cards.

Here is an example to request 1 GPU card and 62 cores to a GPU node.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 62
#SBATCH -p general-gpu
#SBATCH --gres=gpu:1

```