Installing VASP

From Vaspwiki

Contents

Requirements

For the compilation of the parallel version of VASP the following software is mandatory:

- Fortran and C compilers.
- An implementation of MPI (Message Passing Interface).
- Numerical libraries like BLAS, LAPACK, ScaLAPACK, and FFTW.

Build system

The build system of VASP (as of versions >= 5.4.1) has the following structure:

```
            vasp.X.X.X (root directory)
                        |
     ---------------------------------------
    |             |          |             |
   arch           bin       build         src
                                            |
                                      ---------------
                                     |       |       |
                                    lib    parser   CUDA
```

root/
Holds the high-level makefile, and several subdirectories.

root/src
Holds the source files of VASP, and a low-level makefile.

root/src/lib
Holds the source of the VASP library (used to be vasp.X.lib), and a low-level makefile.

root/src/parser
Holds the source of the LOCPROJ parser (as of versions >= 5.4.4), and a low-level makefile.

root/src/CUDA
Holds the source of the cuda-code that will be executed on the GPU by the GPU port of VASP.

root/arch
Holds a collection of makefile.include.arch files.

root/build
The different versions of VASP, i.e., the standard, gamma-only, non-collinear version will be build in separate subdirectories of this directory.

root/bin
Here make will store the binaries.

How to make VASP

Copy one of the `makefile.include.arch` files in `root/arch` to `root/makefile.include`. Take one that most closely reflects your system (hopefully). For instance, on a linux box with the Intel Composer suite:

```
cp arch/makefile.include.linux_intel  ./makefile.include
```

In many cases these `makefile.include` files will have to be adapted to the particulars of your system (see below).

When you've finished setting up `makefile.include`, build VASP:

```
make all
```

This will build the standard, gamma-only, and non-collinear version of VASP one after the other. Alternatively on may build these versions individually:

```
make std
make gam
make ncl
```

To compile the GPU port of VASP:

```
cp arch/makefile.include.linux_intel_cuda ./makefile.include
```

and adapt it to the particulars of your system (see below), followed by:

```
make gpu
make gpu_ncl
```

to built the GPU ports of the standard and non-collinear versions, respectively.

N.B.: Unfortunately at this time we do not offer a GPU port of the gamma-only version yet.

Adapting makefile.include

Precompiler variables

CPP_OPTIONS

Specify the precompiler flags:

```
CPP_OPTIONS=[-Dflag1 [-Dflag2] ... ]
```

Take a lead from the `makefile.include.arch` files in **/arch**, and have a look at the description of the commonly used VASP prcompiler flags.

- N.B.I: -DNGZhalf, -DwNGZhalf, -DNGXhalf, -DwNGXhalf are deprecated options. Building the standard, gamma-only, or non-collinear version of the code is specified through an additional argument to the make command (see the make section).
- N.B.II: CPP_OPTIONS is only used in this file, where it should be added to CPP (see next item).

CPP

The command to invoke the precompiler you want to use, for instance:

- Using Intel's Fortran precompiler:

```
CPP=fpp -f_com=no -free -w0 $*$(FUFFIX) $*$(SUFFIX) $(CPP_OPTIONS)
```

- Using cpp:

```
CPP=/usr/bin/cpp -P -C -traditional $*$(FUFFIX) >$*$(SUFFIX) $(CPP_OPTIONS)
```

- N.B.: This variable has to include $(CPP_OPTIONS)! If not, CPP_OPTIONS will be ignored.

Compiler variables

The Fortran compiler will be invoked as:

```
$(FC) $(FREE) $(FFLAGS) $(OFLAG) $(INCS)
```

FREE

Specify the options that your Fortran compiler needs for it to accept free-form source layout, without line-length limitation. For instance:

- Using Intel's Fortran compiler:

```
FREE=-free -names lowercase
```

- Using gfortran:

```
FREE=-ffree-form -ffree-line-length-none
```

FC

The command to invoke your Fortran compiler (e.g. gfortran, ifort, mpif90, mpiifort, ...).

FCL

The command that invokes the linker. In most cases:

```
FCL=$(FC) [+ some options]
```

- Using the Intel composer suite (Fortan compiler + MKL libraries), typically:

```
FCL=$(FC) -mkl
```

OFLAG

The general level of optimization (default: OFLAG=-O2).

FFLAGS

Additional compiler flags.

- To enable debugging in vasp following line can be added:

```
FFLAGS+=-g
```

OFLAG_IN

(default: -O2) In the vast majority of makefiles this variable is set:

```
OFLAG_IN=$(OFLAG)
```

DEBUG

The optimization level with which the main program (main.F) will be compiled, usually:

```
DEBUG=-O0
```

INCS

Use this variable to specify objects to be included in the sense of:

```
INCS=-Idirectory-with-files-to-be-included
```

Linking against libraries

The linker will be invoked as:

```
$(FCL) -o vasp  ..all-objects.. $(LLIBS) $(LINK)
```

LLIBS

Specify libraries and/or objects to be linked against, in the usual ways:

```
LLIBS=[-Ldirectory -llibrary] [path/library.a] [path/object.o]
```

Usually one has to specify several numerical libraries (BLAS, LAPACK or scaLAPACK, etc). For instance using the Intel composer suite (and compiling with CPP_OPTIONS= .. -DscaLAPACK ..):

```
MKL_PATH   = $(MKLROOT)/lib/intel64
BLACS      = -lmkl_blacs_openmpi_lp64
SCALAPACK  = $(MKL_PATH)/libmkl_scalapack_lp64.a $(BLACS)
LLIBS      = $(SCALAPACK) $(LAPACK)
```

For other configurations please take a lead from the `makefile.include.arch` files under **/arch**, or look at the examples below.

Note on LAPACK 3.6.0 and newer

As of LAPACK >=3.6 the subroutine DGEGV is deprecated and replaced by DGGEV [[1] (http://www.netlib.org/lapack/lapack-3.6.0.html)]. Linking against LAPACK 3.6 or higher will result in following error message:

```
broyden.o: In function `__broyden_MOD_broyd':
broyden.f90:(.text+0x4bb0): undefined reference to `dgegv_'
dynbr.o: In function `brzero_':
dynbr.f90:(.text+0xe78): undefined reference to `dgegv_'
dynbr.f90:(.text+0x112c): undefined reference to `dgegv_'
```

The recommended solution to this problem is to add following line to the `makefile.include`

```
CPP_OPTIONS += -DLAPACK36
```

and following lines to `./src/symbol.inc`

```
! routines replaced in LAPACK >=3.6
#ifdef LAPACK36
#define DGEGV DGGEV
#endif
```

This will replace all calls of DGEGV by DGGEV before compilation.

The list of objects

The standard list of objects needed to compile VASP is given by the variable SOURCE in the `root/src/.objects` file that is part of the distribution.

Objects to be added to this list can be specified in `makefile.include` by means of:

```
OBJECTS= .. your list of objects ..
```

N.B.: Several objects will *have* to be added in this manner (see the following section on "Fast-Fourier-Transforms).

Fast-Fourier-Transforms

OBJECTS
: Add the objects to be compiled (or linked againts) that provide the FFTs (may include static libraries of objects .a).

INCS
: In case one compiles using the fftw-library, i.e.,

```
OBJECTS= .. fftw3d.o fftmpiw.o ..
```

then INCS can be set to the directory that holds `fftw3.f`:

```
INCS=-Idirectory-that-holds-fftw3f
```

(needed because `fftw3d.F` and `fftmpiw.F` include `fftw3.f`).

N.B.: If in the aformentioned case INCS is not set, then `fftw3.f` has to be present in **/src**.

Common choices are:

- To use Intel's MKL wrapper of fftw (and compiling with CPP_OPTIONS= .. -DMPI ..):

```
OBJECTS= fftmpiw.o fftmpi_map.o fftw3d.o fft3dlib.o \
         $(MKLROOT)/interfaces/fftw3xf/libfftw3xf_intel.a
INCS=-I$(MKLROOT)/include/fftw
```

- Or to use Juergen Furtmueller's FFT implementation (and -DMPI):

```

```

```
OBJECTS= fftmpi.o fftmpi_map.o fft3dfurth.o fft3dlib.o

INCS=
```

For other configurations please take lead from the `makefile.include.arch` files under **/arch** or look at the examples below.

Special rules

Special rules for the optimization level of FFT related objects

The makefiles of our old build systems contained a set of special rules for the optimization level allowed in the compilation of the FFT related objects. In the current build system these special rules can be duplicated by adding the following:

```
OBJECTS_O1 += fft3dfurth.o fftw3d.o fftmpi.o fftmpiw.o
OBJECTS_O2 += fft3dlib.o
```

Special rules in general

The current `src/makefile` contains a set of recipes to allow for the compilation of objects at different levels of optimization (other than the general level specified by OFLAG). These recipes replace the special rules section of the makefiles in our old build system.

In these recipes the compiler will be invoked as:

```
$(FC) $(FREE) $(FFLAGS_x) $(OFLAG_x) $(INCS_x)
```

where x stands for: 1, 2, 3, or IN.

FFLAGS_x
: Default: FFLAGS_x=$(FFLAGS), for x=1, 2, 3, and IN.

OFLAG_x
: Default: OFLAG_x=-Ox (for x=1, 2, 3), and OFLAG_IN=-O2

INCS_x
: Default: INCS_x=$(INCS), for x=1, 2, 3, and IN.

The objects to be compiled in accordance with these recipes have to be specified by means of the variables:

OBJECTS_O1, OBJECTS_O2, OBJECTS_O3, OBJECTS_IN

Several objects are compiled at -O1 and -O2 by default. These lists of objects are specified in the .objects file through the variables:

SOURCE_O1, SOURCE_O2, SOURCE_IN

and reflect the special rules as they were present in most of the makefiles of the old build system.

To completely overrule a default setting (for instance for the -O1 special rules) use the following construct:

```
SOURCE_O1=
OBJECTS_O1= .. your list of objects ..
```

For the VASP library (`lib`)

CPP_LIB

The command to invoke the precompiler. In most cases it will suffise to set:

```
CPP_LIB=$(CPP)
```

FC_LIB

The command to invoke your Fortran compiler. In most cases:

```
FC_LIB=$(FC)
```

N.B.: the library can be compiled without MPI support, i.e., when `FC=mpif90`, FC_LIB may specify a Fortran compiler without MPI supprt, e.g. `FC_LIB=ifort`.

FFLAGS_LIB

Fortran compiler flags, including a specification of the level of optimization. In most cases:

```
FFLAGS_LIB=-O1
```

FREE_LIB

Specify the options that your Fortran compiler needs for it to accept free-form source layout, without line-length limitation. In most cases it will suffise to set:

```
FREE_LIB=$(FREE)
```

CC_LIB

The command to invoke your C compiler (e.g. gcc, icc, ..).
N.B.: the library can be compiled without MPI support.

CFLAGS_LIB

C compiler flags, including a specification of the level of optimization. In most cases:

```
CFLAGS_LIB=-O
```

OBJECTS_LIB

List of "non-standard" objects to be added to the library. In most cases:

```
OBJECTS_LIB= linpack_double.o
```

When compiling VASP with `-Duse_shmem`, one has to add `getshmem.o` as well, i.e.,

```
OBJECTS_LIB= .. getshmem.o ..
```

For the LOCPROJ-parser (`parser`)

CXX_PARS
The command to invoke your C++ compiler (e.g. g++, icpc, ..).

And tell make that the parser needs to be compiled:

```
LIBS += parser
```

and needs to be linked against:

```
LLIBS += -Lparser -lparser -lstdc++
```

For the interface to Wannier90 (optional)

To include the interface to Wannier90 (-DVASP2WANNIER90 or -DVASP2WANNIER90v2), one needs to specify:

```
LLIBS += /your-wannier90-directory/libwannier.a
```

And one needs to download Wannier90 (http://www.wannier.org) and compile `libwannier.a`.

For `libbeef` (optional)

In case one wants to compile VASP with the BEEF van-der-Waals functionals (-Dlibbeef), one needs to add:

```
LLIBS += -Lyour-libbeef-directory -lbeef
```

And one needs to download and compile libbeef (https://github.com/vossjo/libbeef), of course.

For the GPU port

CUDA_ROOT
Location of CUDA toolkit install. For example:

```
CUDA_ROOT := /opt/cuda
```

CUDA_LIB
CUDA toolkit libraries to link to. Typically:

```
CUDA_LIB := -L$(CUDA_ROOT)/lib64 -lnvToolsExt -lcudart -lcuda -lcufft -lcublas
```

NVCC

Location of CUDA compiler and flags. Typically:

```
NVCC := $(CUDA_ROOT)/bin/nvcc -g
```

OBJECTS_GPU

Add the objects to be compiled (or linked againts) that provide the FFTs (may include static libraries of objects .a). For FFTW:

```
OBJECTS_GPU = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d_gpu.o fftmpiw_gpu.o
```

GENCODE_ARCH

CUDA compiler options to generate code for your particular GPU architecture.
For Kepler:

```
GENCODE_ARCH := -gencode=arch=compute_35,code=\"sm_35,compute_35\"
```

For Maxwell:

```
GENCODE_ARCH := -gencode=arch=compute_53,code=\"sm_53,compute_53\"
```

Multiple `-gencode` statements can be compiled to create cross-platform executables.
For details see the NVIDIA nvcc documentation (http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc).

MPI_INC

Path to MPI include files so the CUDA compiler can find them. For example:

```
MPI_INC := /opt/openmpi/include
```

These can often be found with `mpicc --show`.

CPP_GPU

Preprocessor options for GPU compilation.
Always include:

> `-DCUDA_GPU` to build cross-platform sources for GPU, `-DUSE_PINNED_MEMORY` to use pinned memory for transfer buffers, and `-DRPROMU_CPROJ_OVERLAP` to overlap communication and computation in RPROJ_MU.

Optional:

> Set `-DCUFFT_MIN=N` to intercept any FFT calls of size greater than N^3 and evaluate on GPU.

Experimental:

> Set `-DUSE_MAGMA` to use MAGMA for LAPACK-like calls on the GPU.

So typically:

```
CPP_GPU    = -DCUDA_GPU -DRPROMU_CPROJ_OVERLAP -DUSE_PINNED_MEMORY -DCUFFT_MIN=28
```

MAGMA_ROOT
If using the experimental MAGMA support, path to MAGMA 1.6. Typically:

```
MAGMA_ROOT := /opt/magma/lib
```

Examples

- makefile.include.linux_intel
- makefile.include.linux_gfortran
- makefile.include.linux_intel_cuda

- A Debian based installation of VASP
- A Ubuntu based installation of VASP
- A Fedora based installation of VASP
- A CentOS based installation of VASP

- Linking gfortran with Intel MKL

Patches

For vasp.5.4.1.24Jun15

Unfortunately several bugs were reported for vasp.5.4.1.24Jun15. To fix them download the patch (es) below:

- patch.5.4.1.08072015.gz (http://cms.mpi.univie.ac.at/patches/patch.5.4.1.08072015.gz)
- patch.5.4.1.27082015.gz (http://cms.mpi.univie.ac.at/patches/patch.5.4.1.27082015.gz)
- patch.5.4.1.06112015.gz (http://cms.mpi.univie.ac.at/patches/patch.5.4.1.06112015.gz)

To apply these patch(es) gunzip the patch file(s) and

```
patch -p1 < patch.5.4.1.ddmmyyyy
```

within your vasp.X.X.X root-directory.

For vasp.5.4.1.05Feb16 (with GPU support)

The following patch improves the mapping between MPI-ranks and GPUs on multi-node/multi-GPU systems (the issue is performance only, not a bugfix):

- patch.5.4.1.14032016.gz (http://cms.mpi.univie.ac.at/patches/patch.5.4.1.14032016.gz)

The following patch unfortunately does address several bugs:

1. For noncollinear calculations LOPTICS=.TRUE. didn't work correctly with Blöchl-smearing (ISMEAR≤-4).

2. The Zeroth-Order-Regular-Approximation (ZORA) that accounts for the relativistic mass correction in the Spin-Orbit-Coupling operator was not implemented correctly.

   N.B.: Unfortunately this bugfix affects the total energy. Effects are expected to be negligible except for heavy elements.

- patch.5.4.1.03082016.gz (http://cms.mpi.univie.ac.at/patches/patch.5.4.1.03082016.gz)

To apply these patches gunzip the patch files and

```
patch -p0 < patch.5.4.1.14032016
patch -p0 < patch.5.4.1.03082016
```

within your vasp.X.X.X root-directory.

For vasp.5.4.4.18Apr17-6-g9f103f2a35

The following patch addresses a few issues:

1. Fixes a bug in the stress term when using the SCAN functional in certain pathological cases.
2. Fixes a bug in the Thomas-Fermi potential.
3. Fixes a bug that affected the optB88 for some atoms and molecules.
4. Fixes some issues with the BSE at finite **q**.

and adds support for:

1. The CX13 vdW-DFT functional.

- patch.5.4.4.16052018.gz (http://cms.mpi.univie.ac.at/patches/patch.5.4.4.16052018.gz)

To apply this patch gunzip the patch file and

```
patch -p0 < patch.5.4.4.16052018
```

within your vasp.X.X.X root-directory.

Validation

We are currently constructing a suite of tests to be used to validate your VASP executables.

Related Sections

Precompiler flags, GPU port of VASP

Contents

Retrieved from "https://cms.mpi.univie.ac.at/wiki/index.php?title=Installing_VASP&oldid=9905"

Categories: VASP | Installation | Performance | GPU