Compiling native parallel-computing version of Wannier90 in Windows using Microsoft MPI, Gfortran from MINGW (MSYS2)

©QuantumNerd
GitHub: https://github.com/quantumNerd
YouTube: https://www.youtube.com/channel/UCgQPek4ZSo_yL7wEjIhxvfA
Twitter: https://twitter.com/realquantumnerd
Date: 2020.Sep.19, Wannier90 version 3.1.0

README

The goal of this document is to provide a guidance of compiling Wannier90 code on Windows and achieve high efficiency via native compilation and parallel computing on multiple CPU cores. An alternative to the procedure described below would be to use Cygwin, but it is not native compilation (Cygwin provides a Unix-like environment in Windows) and there is no easy multi-core parallel computation support to my knowledge, which makes the efficiency low. Consequently, we choose the procedure below.

Notice that like all other programs in Windows, the compiled version most likely works in any Windows machine. Therefore, **instead of trying to replicate the recipe below yourself, you could simply go to my GitHub page (https://github.com/quantumNerd/Wannier90_for_Windows) and download the compiled version (the result of this recipe) which is ready-to-use in Windows.**

MSYS2

MSYS2 is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software.

1. Download and install MSYS2: https://www.msys2.org/
2. We need GNU compiler v9
   - ➔ v10 does not work because the grammar is too strict, will give the following error (`Error: BOZ literal constant`), which can be solved by passing a flag to gfortran, but will then give another error (`Error: Type mismatch between actual argument at (1) and actual argument at (2) (INTEGER(4)/COMPLEX(8))`)
   - ➔ However, the current default of GNU compilers in MSYS2 is v10
   - ➔ MSYS2 does not have automatic version roll-back of compilers, so you need to do it manually, as described below
3. Install GNU compiler v9
   - ➔ Go to MSYS
   - ➔ Uninstall all current versions manually (including all dependencies). Just try first with

"pacman -R mingw-w64-i686-gcc" and it will indicate corresponding dependencies to uninstall.

- ◆ (32-bit, not necessarily as below)

```
pacman -R mingw-w64-i686-gcc-ada
pacman -R mingw-w64-i686-msmpi
pacman -R mingw-w64-i686-gcc-fortran
pacman -R mingw-w64-i686-openmp
pacman -R mingw-w64-i686-uasm
pacman -R mingw-w64-i686-gcc-objc
pacman -R mingw-w64-i686-gcc
```

- ◆ (64-bit, not necessarily as below)

```
pacman -R mingw-w64-x86_64-msmpi
pacman -R mingw-w64-x86_64-gcc-fortran
pacman -R mingw-w64-x86_64-gcc
pacman -R mingw-w64-x86_64-lapack
pacman -R mingw-w64-x86_64-openblas
pacman -R mingw-w64-x86_64-gcc-libgfortran
```

➔ Download older packages from the website http://repo.msys2.org/mingw/. 64 bit would be http://repo.msys2.org/mingw/x86_64/, while 32-bit would be http://repo.msys2.org/mingw/i686/. The package list is shown below.

➔ Install packages using the following commands. There will be instructions in the command line if the dependency is not met, and just follow them to resolve the dependency necessary.

- ◆ (32-bit, not necessarily as below)

```
pacman -U mingw-w64-i686-gcc-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-i686-gcc-ada-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-i686-gcc-fortran-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-i686-gcc-libgfortran-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-i686-gcc-libs-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-i686-gcc-objc-9.3.0-2-any.pkg.tar.xz
```

- ◆ (64-bit, not necessarily as below)

```
pacman -U mingw-w64-x86_64-gcc-libs-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-x86_64-gcc-libgfortran-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-x86_64-gcc-9.3.0-2-any.pkg.tar.xz
pacman -U mingw-w64-x86_64-gcc-fortran-9.3.0-2-any.pkg.tar.xz
```

4. Regularly install other packages (Microsft MPI (MSMPI), LAPACK, OpenBLAS, 64 bit for example):

```
pacman -S mingw-w64-i86_64-msmpi
pacman -S mingw-w64-i86_64-lapack
pacman -S mingw-w64-i86_64-openblas
```

Microsoft MPI (MSMPI):

Microsoft MPI (MS-MPI) is a Microsoft implementation of the Message Passing Interface standard for developing and running parallel applications on the Windows platform. The reference of the following part regarding MSMPI is https://www.math.ucla.edu/~wotaoyin/windows_coding.html

1. Download Microsoft MPI. v10.0 has been tested.
   https://www.microsoft.com/en-us/download/details.aspx?id=57467
   - ➔ For gendef used below, you need to run "pacman -S mingw-w64-x86_64-tools-git" in MSYS2
2. Install both msmpisdk.msi and MSMpiSetup.exe
3. Execute C:\msys64\mingw64.exe and locate the environment variables WINDIR, MSMPI_INC, and MSMPI_LIB64 by running:

```
printenv | grep "WIN\|MSMPI"
```

   - ➔ If you don't see them, then your Windows environment variables are not passed to MSYS2-MINGW64; you need to correct this before proceeding to the next step.

4. Add/create the header and library files for MS-MPI for later use:

```
mkdir ~/msmpi                          # create a temporary folder under your home directory
cd ~/msmpi                             # enter the folder
cp "$MSMPI_LIB64/msmpi.lib" .          # copy msmpi.lib to ~/msmpi/; the import library, which is a placeholder
for dll
cp "$WINDIR/system32/msmpi.dll" .      # copy msmpi.dll to ~/msmpi/; the runtime library
gendef msmpi.dll                       # generate msmpi.def. For 32-bit, use: gendef -a msmpi.dll, which
                                         specifies the stdcall format
dlltool -d msmpi.def -D msmpi.dll -l libmsmpi.a      # generate the (static) library file libmsmpi.a
cp libmsmpi.a /mingw64/lib             # copy this library file to where g++ looks for them;
                                       # try "g++ --print-search-dirs"
cp "$MSMPI_INC/mpi.h" .                # copy the header file mpi.h to ~/msmpi/
cp mpi.h /mingw64/include              # copy the header file to the default include folder
```

5. Now you can delete the folder ~/msmpi
6. Copy all files in C:\Program Files (x86)\Microsoft SDKs\MPI\Include and C:\Program Files (x86)\Microsoft SDKs\MPI\Include\x64 into some folder, let's say "C:\msys64\mingw64\include" in windows which would be "/mingw64/include" in MSYS2 language used later in the configuration file. If you do not do this, during the compilation Wannier90 might complain that it cannot find some header files like "mpi.f90".

Configure Wannier90 compilation

1. Now go to MINGW64 for 64-bit case, or MINGW32 for 32-bit case. Below only 64-bit is demonstrated, but the 32-bit case is the same.
   - ➔ Use "gfortran -v" to check version, and it should be 9.3.0 for our case. If it is newer (v10), please go to the previous instruction about how to downgrade it to v9, otherwise the compilation will not work.
   - ➔ Unzip wannier90-3.1.0 and go inside the folder
     - ◆ tar –zxvf wannier90-3.1.0.tar.gz
     - ◆ cd wannier90-3.1.0

- ➔ Copy the “make.inc.gfort” configuration file from “config” subfolder to the base directory and rename it as “make.inc”
- ➔ Change to the following lines in “make.inc” (you should NOT use MPIF90 = mpif90, because that might link against other MPI library causing error in runtime with multiple cores (while “mpiexec -np 1” case is ok), which is tricky to find out):

```
F90 = gfortran

COMMS    = mpi
MPIF90 = gfortran

FCOPTS = -O3 -fno-range-check -I/mingw64/include
LIBS = -llapack -lblas -lmsmpi
```

Install Wannier90

- ➔ If you have made anything before, use “make veryclean” to clean everything
- ➔ Use “make” (or “make all”) to compile. You should see lines like “gfortran -DMPI -O3 -fno-range-check -I/mingw64/include -c ../io.F90” while compiling and no warning or error

Test Wannier90 (in MINGW64 terminal)

1. Still in MINGW64 terminal
2. Run “export PATH="C:\Program Files\Microsoft MPI\Bin":$PATH” so that mpiexec works. Otherwise you will see the error “bash: mpiexec: command not found”
3. Go to “~/wannier90-3.1.0/examples/example16-noqe”
   - ➔ Serial: “../../wannier90.x Si.win” -> 2.578s
   - ➔ Parallel with one core (effectively serial, just to check mpiexec works) “mpiexec -np 1 ../../wannier90.x Si.win” -> 2.578s
   - ➔ Parallel with 6 cores “mpiexec -np 6 ../../wannier90.x Si.win” -> “Running in parallel on   6 CPUs” -> 0.953s
4. Check the timing in “Si.wout”, which on my computer is shown above. The multi-core version must work and must work much more efficiently, which means everything is successful.
5. Compare the result with reference output file, which should be consistent.

Test using test-suite (optional, will only partially work)

Wannier90 come with a test-suite that automatically tests the program in different calculations. In Linux this should work fine, but on Windows it does not work originally. First, it only works within MINGW terminal (not from CMD) because it needs “make” command from Linux. Second, several lines of python code need to be adapted. In the end, only some of the tests will pass while those that requires previous calculation results will fail because the “make” command in the MINGW terminal is not working fully as expected. **Consequently, the recommended way will be running**

tests manually (discussed above) rather than following the steps below in this section.

➔ Install python3 in MSYS2

➔ Install python3-pip

➔ Go to MINGW terminal:

- pip install configparser
- Change L169 in "run_tests" to be:

```
command_pieces = ["python"]+[
            testcode_exe] + verbose_cmd + [
            "--category={}".format(the_category),
            "--jobconfig={}".format(jobconfig),
            "--userconfig={}".format(userconfig)
            ]
```

- ./run_tests
- Change L73 in "clean_tests" to be:

```
errcode = subprocess.call(['git', '-C', test_folder, 'rev-parse', '--git-dir'],shell=True)
```

Use Wannier90 in CMD (outside of MINGW64)

We not only want to run Wannier90 through MSYS2 and MINGW64 terminal, but also through CMD which will be available on all Windows machines. We need to collect some supporting .dll files and put them in the same folder of Wannier90 program as shown in the steps below.

1. Run "ldd wannier90.x" in MINGW64 terminal , it will show a list of files that this program depends on

```
        ntdll.dll => /c/WINDOWS/SYSTEM32/ntdll.dll (0x7ffdc8b60000)
        KERNEL32.DLL => /c/WINDOWS/System32/KERNEL32.DLL (0x7ffdc77b0000)
        KERNELBASE.dll => /c/WINDOWS/System32/KERNELBASE.dll (0x7ffdc5b60000)
        msvcrt.dll => /c/WINDOWS/System32/msvcrt.dll (0x7ffdc6cc0000)
        msmpi.dll => /c/WINDOWS/SYSTEM32/msmpi.dll (0x7ffd726f0000)
        ADVAPI32.dll => /c/WINDOWS/System32/ADVAPI32.dll (0x7ffdc71a0000)
        sechost.dll => /c/WINDOWS/System32/sechost.dll (0x7ffdc8990000)
        RPCRT4.dll => /c/WINDOWS/System32/RPCRT4.dll (0x7ffdc6d90000)
        libblas.dll => /mingw64/bin/libblas.dll (0x66600000)
        liblapack.dll => /mingw64/bin/liblapack.dll (0x61780000)
        libgfortran-5.dll => /mingw64/bin/libgfortran-5.dll (0x65440000)
        WS2_32.dll => /c/WINDOWS/System32/WS2_32.dll (0x7ffdc7d20000)
        libgcc_s_seh-1.dll => /mingw64/bin/libgcc_s_seh-1.dll (0x61440000)
        libgcc_s_seh-1.dll => /mingw64/bin/libgcc_s_seh-1.dll (0x150000)
        libquadmath-0.dll => /mingw64/bin/libquadmath-0.dll (0x6cf00000)
        libwinpthread-1.dll => /mingw64/bin/libwinpthread-1.dll (0x64940000)
        MSWSOCK.dll => /c/WINDOWS/SYSTEM32/MSWSOCK.dll (0x7ffdc5270000)
        NTDSAPI.dll => /c/WINDOWS/SYSTEM32/NTDSAPI.dll (0x7ffda9600000)
        AUTHZ.dll => /c/WINDOWS/SYSTEM32/AUTHZ.dll (0x7ffdc4bb0000)
        ucrtbase.dll => /c/WINDOWS/System32/ucrtbase.dll (0x7ffdc5e10000)
        CRYPTBASE.DLL => /c/WINDOWS/SYSTEM32/CRYPTBASE.DLL (0x7ffdc5440000)
        bcryptPrimitives.dll => /c/WINDOWS/System32/bcryptPrimitives.dll (0x7ffdc5ae0000)
        DSPARSE.DLL => /c/WINDOWS/SYSTEM32/DSPARSE.DLL (0x7ffdb04d0000)
```

2. Copy all that is not in the system32 folder and distribute with wannier90.x code (same folder). Do the same for postw90.x code (using command “ldd postw90.x”). Check again, and everything should either exist in the same folder or in SYSTEM32.
3. Set environment variable (add “C:\Program Files\Microsoft MPI\Bin” to path for windows)
4. To test, go to CMD and go to folder “C:\msys64\home\anonymous\wannier90-3.1.0\examples\example16-noqe”, try the same command “mpiexec -np 6 ../../wannier90.x Si.win”, should work. In case it says some .dll missing, just find out the position and copy that to wannier90-3.1.0 root directory
    - ➔ Benchmark timing results:

Cores	Serial	-np 1	-np 2	-np 3	-np 6	-np 12
Total time/s	2.641	2.594	1.411	1.109	0.891	8.5

    - ➔ I have 6 CPU cores in my computer, so running in parallel with 6 cores is most efficient (shortest time). When the number of jobs exceeds 6, the efficiency drastically goes down as expected. Consequently, the parallel computing version of Wannier90 has been successfully installed.