NWChemEx Install Instructions

Setup

Typically, setup involves having to obtain prerequisite software to be able to compile the NWChemEx stack. This includes the libraries Blas/OpenBlas and Libint2. For Blas/OpenBlas, the software can be downloaded via a package manager that has a package available that provides the library. Usually when a package is installed via the main package manager of an operating system, those libraries end up in the `/usr/lib` directory. This folder is usually checked by default by our current build system, CMake, to find the prerequisite software.

Personally, I prefer putting libraries into `/home/jacob/Libraries` on my own system:

```
cd /home/jacob/Libraries
curl -L -O https://github.com/evaleev/libint/releases/download/v2.6.0/libint-2.6.0.tgz
tar xvf libint-2.6.0.tgz
cd libint-2.6.0
```

Knowing where the library is installed is important if you plan to download and compile the libraries from source.

Libint2 Build This tells `cmake` that the current source directory is the current file (-S), that we want the configuration and build to happen in the "build" directory (`cmake` will create this folder if it doesn't exist), and that we would like to set the `CMAKE_INSTALL_PREFIX` to `` `pwd`/install, `` which will resolve to '`/home/jacob/Libraries/libint-2.6.0/install`, meaning that the result of the build will be installed to that path.

```
cmake -S . -B build -DCMAKE_INSTALL_PREFIX=`pwd`/install
```

If needed, you can confirm that the `CMAKE_INSTALL_PREFIX` variable was set via `ccmake`, but using `ccmake` to configure and generate the build usually results in undefined behavior for the build, so I would just use `ccmake` to confirm that variables have been set correctly.

```
# OPTIONAL
ccmake build # Use 'q' to quit
```

Then the build begins! Specify that the build directory is build, the target is to install the library, and that we would like to use 4 cores to compile the code.

```
cmake --build build --target install --parallel 4
```

At the end of compilation, you should have a `install/` folder in the libint-2.6.0 folder, which will have `lib/`, `include/`, and `share/`. This is important to know for our `CMAKE_PREFIX_PATH` variable later for building NWChemEx.

Blas/OpenBlas This library is typically the easiest to install, I usually go through the main package manager of the operating system. This would be the following

Ubuntu/Debian

```
sudo apt isntall libopenblas-dev
```

As long as you are able to point to `libopenblas.so` via `CMAKE_PREFIX_PATH`, this should work fine.

Obtaining NWChem NWChemEx depends on a few executable programs, one of which is NWChem. You can obtain NWChem a multitude of ways.

Ubuntu/Debian

```
sudo apt install nwchem
```

Homebrew

```
brew install nwchem
```

Nix Package Manager

```
nix-env -iA nixpkgs.nwchem
```

The Homebrew and Nix versions of NWChem are build with OpenMPI, which requires it to be run via `mpirun nwchem -np <number-of-cores> nwchem input.nw`, but the implementation in the underlying framework QCEngine doesn't handle this well. If you install a version of NWChem that requires `mpirun`, you should use the following script and have it in your path:

```
#!/usr/bin/env bash
REAL_NWCHEM="<path-to-nwchem>" # eg /home/jacob/Applications/nwchem/bin/LINUX64/nwchem DO NOT JUST DO n
echo "Running NWChem located at: $REAL_NWCHEM"
if [[ "$0" == "$REAL_NWCHEM" ]]; then
    echo "Error: Wrapper is calling itself!" >&2
    exit 1
fi
exec mpirun "$REAL_NWCHEM" "$@"
```

This essentially tricks QCEngine into running NWChem with `mpirun`.

Python Environment I like to set up my python environments in `/home/jacob/Environments/`, but the path doesn't necessarily matter so long as your source it BEFORE you configure NWChemEx via `cmake` (otherwise the interpreter path for NWChemEx will be set to the incorrect path)

```
cd /home/jacob/Environments
python3 -m venv nwchemex-env
# I prefer to source from an absolue path /home/jacob/Environments/nwchemex-env/bin/activate, but
# this way is fine as well
source ./nwchemex-env/base/bin/activate
```

The necessary python packages that we need are `qcelemental`, `qcengine`, `networkx`, and `ase`:

```
# With the environment activated
pip install qcelemental qcengine networkx ase
```

NWChemEx Build Now we can go on to building NWChemEx! We typically use a `toolchain.cmake` file to specify certain variables to pass to cmake at configuration, here is mine:

```
# GCC Setup
set(CMAKE_C_COMPILER   gcc)
set(CMAKE_CXX_COMPILER g++)
set(MPI_C_COMPILER     mpicc)
set(MPI_CXX_COMPILER   mpic++)

# Options
set(CMAKE_POSITION_INDEPENDENT_CODE TRUE)
set(BUILD_SHARED_LIBS TRUE)
set(BUILD_TESTING TRUE)
set(CMAKE_EXPORT_COMPILE_COMMANDS TRUE) # Useful for LSPs like Clangd
set(CMAKE_POLICY_VERSION_MINIMUM 3.5) # Only needed if CMake version is too new
set(ENABLE_SIGMA ON)

# List directories for dependencies you have installed in non-standard
# locations. For example:
```

```
set(CMAKE_PREFIX_PATH "/home/jacob/Libraries/libint-2.9.0/install:/usr/lib") # /usr/lib should already
set(CMAKE_CXX_STANDARD 17)

# BLAS/LAPACK
set(ENABLE_SCALAPACK ON)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -DOMPI_SKIP_MPICXX")
```

Some things to note, if you need to install a version of ChemCache that has more basis sets available, you will most likely need to clone ChemCache and check out the `generated_data` branch:

```
# I like to keep outside projects in my /home/jacob/Projects folder
cd /home/jacob/Projects
git clone https://github.com/NWChemEx/ChemCache.git
cd chemcache
git checkout generated_data
```

You can confirm that the generated data is available if you go to the `src/chemcache/bases` folder and see more than just `sto_dash_3g` in the folder. Then you will want to add the following line to the `toolchain.cmake` file:

```
set(FETCHCONTENT_SOURCE_DIR_CHEMCACHE "/home/jacob/Projects/ChemCache")
```

Now we are ready to configure, confirm, and build NWChemEx. We can run the following command to configure the NWChemEx build (**MAKE SURE YOU HAVE SOURCED YOUR PYTHON ENVIRONMENT!**):

```
cmake -S . -B build -DCMAKE_INSTALL_PREFIX=`pwd`/install \
                    -DNWX_MODULE_DIR=`pwd`/install \
                    -DCMAKE_TOOLCHAIN_FILE=toolchain.cmake
```

CMake will configure the build and download any sources that it can't find. After a successful configuration, we will want to confirm that our variables were set via `ccmake`, you will want to confirm `CMAKE_INSTALL_PREFIX`, `CMAKE_PREFIX_PATH`, and `FETCHCONTENT_SOURCE_DIR_CHEMCACHE` if set.

Now, it is time to build the NWChemEx stack:

```
cmake --build build --target install --parallel 4
```

You can use more than 4 cores if needed, but if you are building the full ChemCache with the generated_data branch, this may cause the memory to be exceeded on your system. 4 cores is pretty safe. If you are building the full ChemCache with generated data, this build process will take a while.

Once the build is complete, you should have a folder named `install/` in the NWChemEx directory. We need to do a bit of clean up since the NWChemEx install process isn't fully working yet.

First, let's check out what is in the `install/` folder:

```
Permissions Size User  Date Modified Name
drwxr-xr-x     - jacob 10 Jun 17:59  bin
drwxr-xr-x     - jacob 10 Jun 17:59  friendzone
drwxr-xr-x     - jacob 10 Jun 17:59  include
drwxr-xr-x     - jacob 10 Jun 17:59  lib
drwxr-xr-x     - jacob 10 Jun 17:59  nwchemex
drwxr-xr-x     - jacob 10 Jun 17:59  share
.rwxr-xr-x  210k jacob 10 Jun 17:59  chemcache.so
.rwxr-xr-x  3.0M jacob 10 Jun 17:52  chemist.so
.rwxr-xr-x  210k jacob 10 Jun 17:52  integrals.so
.rwxr-xr-x  210k jacob 10 Jun 17:49  nux.so
.rwxr-xr-x  512k jacob 10 Jun 17:44  parallelzone.so
.rwxr-xr-x  914k jacob 10 Jun 17:45  pluginplay.so
```

```
.rwxr-xr-x  232k jacob 10 Jun 17:53  scf.so
.rwxr-xr-x  2.7M jacob 10 Jun 17:48  simde.so
.rwxr-xr-x  298k jacob 10 Jun 17:47  tensorwrapper.so
```

The result of setting `-DNWX_MODULE_DIRECTORY=`pwd`/install` is that the pybind11 python modules (denoted by `.so` suffix, referencing `parallelzone.so`, `chemcache.so`, etc.) as well as the `friendzone` and `nwchemex` python modules are installed here. When you plan to start interacting with the NWChemEx stack via `python`, this is the path where you will set `PYTHONPATH` environment variable. This will allow `python` to find these modules, so they can be imported like usual in a python setting (i.e. `import parallelzone`).

Now, let's go into the `lib/` folder:

```
Permissions Size User  Date Modified Name
drwxr-xr-x     - jacob 10 Jun 17:59  chemcache
drwxr-xr-x     - jacob 10 Jun 17:59  chemist
drwxr-xr-x     - jacob 10 Jun 17:59  cmake
drwxr-xr-x     - jacob 10 Jun 17:59  integrals
drwxr-xr-x     - jacob 10 Jun 17:59  parallelzone
drwxr-xr-x     - jacob 10 Jun 17:59  pkgconfig
drwxr-xr-x     - jacob 10 Jun 17:59  pluginplay
drwxr-xr-x     - jacob 10 Jun 17:59  sigma
drwxr-xr-x     - jacob 10 Jun 17:59  simde
drwxr-xr-x     - jacob 10 Jun 17:59  tensorwrapper
drwxr-xr-x     - jacob 10 Jun 17:59  utilities
.rwxr-xr-x  691k jacob 10 Jun 17:45  libexchcxx.so
lrwxrwxrwx     - jacob 10 Jun 17:59  libfort.so -> libfort.so.0.4
lrwxrwxrwx     - jacob 10 Jun 17:59  libfort.so.0.4 -> libfort.so.0.4.2
.rwxr-xr-x  118k jacob 10 Jun 17:43  libfort.so.0.4.2
.rw-r--r--  6.0M jacob 10 Jun 17:46  libgauxc.a
lrwxrwxrwx     - jacob 10 Jun 17:59  libspdlog.so -> libspdlog.so.1.11
lrwxrwxrwx     - jacob 10 Jun 17:59  libspdlog.so.1.11 -> libspdlog.so.1.11.0
.rwxr-xr-x  748k jacob 10 Jun 17:44  libspdlog.so.1.11.0
lrwxrwxrwx     - jacob 10 Jun 17:59  libxc.so -> libxc.so.12
.rwxr-xr-x   13M jacob 10 Jun 17:45  libxc.so.12
```

In order to be able to use the build libraries for use with both the C++ and Python interfaces, we need to make it a bit easier to find the libraries. I do this by symlinking all the libraries with the `chemcache`, `chemist`, `integrals`, `parallelzone`, `pluginplay`, `tensorwrapper`, and `utilities` folders to the `lib/` directory. You can do this via the following script from the `lib/` directory:

```bash
#!/usr/bin/env bash
ln -s ./chemcache/libchemcache.so.1 .
ln -s ./chemist/libchemist.so.1 .
ln -s ./integrals/libintegrals.so.0 .
ln -s ./parallelzone/libparallelzone.so.0 .
ln -s ./pluginplay/libpluginplay.so.1 .
ln -s ./tensorwrapper/libtensorwrapper.so.0 .
ln -s ./utilities/libutilities.so.0 .
```

Running the script should result in the following files in the `lib/` directory:

```
Permissions Size User  Group Date Modified Git Name
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I chemcache
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I chemist
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I cmake
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I integrals
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I parallelzone
```

```
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I pkgconfig
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I pluginplay
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I sigma
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I simde
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I tensorwrapper
drwxr-xr-x     - jacob jacob 10 Jun 17:59   -I utilities
lrwxrwxrwx     - jacob jacob 12 Jun 16:41   -I libchemcache.so.1 -> ./chemcache/libchemcache.so.1
lrwxrwxrwx     - jacob jacob 12 Jun 16:41   -I libchemist.so.1 -> ./chemist/libchemist.so.1
.rwxr-xr-x  691k jacob jacob 10 Jun 17:45   -I libexchcxx.so
lrwxrwxrwx     - jacob jacob 10 Jun 17:59   -I libfort.so -> libfort.so.0.4
lrwxrwxrwx     - jacob jacob 10 Jun 17:59   -I libfort.so.0.4 -> libfort.so.0.4.2
.rwxr-xr-x  118k jacob jacob 10 Jun 17:43   -I libfort.so.0.4.2
.rw-r--r--  6.0M jacob jacob 10 Jun 17:46   -I libgauxc.a
lrwxrwxrwx     - jacob jacob 12 Jun 16:41   -I libintegrals.so.0 -> ./integrals/libintegrals.so.0
lrwxrwxrwx     - jacob jacob 12 Jun 16:41   -I libparallelzone.so.0 -> ./parallelzone/libparallelzone.s
lrwxrwxrwx     - jacob jacob 12 Jun 16:41   -I libpluginplay.so.1 -> ./pluginplay/libpluginplay.so.1
lrwxrwxrwx     - jacob jacob 10 Jun 17:59   -I libspdlog.so -> libspdlog.so.1.11
lrwxrwxrwx     - jacob jacob 10 Jun 17:59   -I libspdlog.so.1.11 -> libspdlog.so.1.11.0
.rwxr-xr-x  748k jacob jacob 10 Jun 17:44   -I libspdlog.so.1.11.0
lrwxrwxrwx     - jacob jacob 12 Jun 16:41   -I libtensorwrapper.so.0 -> ./tensorwrapper/libtensorwrappe
lrwxrwxrwx     - jacob jacob 12 Jun 16:41   -I libutilities.so.0 -> ./utilities/libutilities.so.0
lrwxrwxrwx     - jacob jacob 10 Jun 17:59   -I libxc.so -> libxc.so.12
.rwxr-xr-x   13M jacob jacob 10 Jun 17:45   -I libxc.so.12
```

Now that we have the libraries more easily accessible, we can then set the `LD_LIBRARY_PATH` variable to this directory so that our Python modules can use them.

Once all these steps are done, and the `PYTHONPATH` and `LD_LIBRARY_PATH` variables are set, this should result in a functioning Python interface.

If you plan on using C++ to interact with NWChemEx, here is an example of how I make sure that my `main.cpp` file is compiled correctly:

```
CXXFLAGS = -std=c++23 -Wall -Wextra -g \
-I/home/jacob/Applications/NWChemEx/install/include \
-I/home/jacob/Applications/NWChemEx/build/_deps/scf-src/include \
-I/home/jacob/Applications/NWChemEx/build/_deps/nux-src/include \
-I/home/jacob/Environments/base/include \
# -I/nix/store/a8r6jizjba7g1n99fla78aban17qcf9h-openmpi-5.0.6-dev/include \
# -I/nix/store/ammv4hfx001g454rn0dlgibj1imn9rkw-boost-1.87.0-dev/include

LDFLAGS = \
-L/home/jacob/Applications/NWChemEx/build/_deps/scf-build/ -lscf \
-L/home/jacob/Applications/NWChemEx/build/_deps/nux-build/ -lnux \
-L/home/jacob/Applications/NWChemEx/install/lib/utilities/ -lutilities \
-L/home/jacob/Applications/NWChemEx/install/lib/chemist/ -lchemist \
-L/home/jacob/Applications/NWChemEx/install/lib/pluginplay/ -lpluginplay \
-L/home/jacob/Applications/NWChemEx/install/lib/chemcache/ -lchemcache \
-L/home/jacob/Applications/NWChemEx/install/lib/tensorwrapper/ -ltensorwrapper \
-L/home/jacob/Applications/NWChemEx/install/lib/integrals/ -lintegrals

scf: scf_double.cpp
    @g++ $(CXXFLAGS) main.cpp -o executable_binary $(LDFLAGS)
```