XCONFIGURE

XCONFIGURE is a collection of configure wrapper scripts for various HPC applications. The purpose of the scripts is to configure the application in question to make use of Intel's software development tools (Intel Compiler, Intel MPI, Intel MKL). XCONFIGURE helps to rely on a "build recipe", which is known to expose the highest performance or to reliably complete the build process.

> Contributions are very welcome!

Each application (or library) is hosted in a separate directory. To configure (and ultimately build) an application, one can rely on a single script which then downloads a specific wrapper into the current working directory (of the desired application).

```
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure-get.sh
chmod +x configure-get.sh

echo "EXAMPLE: recipe for Quantum Espresso"
./configure-get.sh qe hsw
```

On systems without access to the Internet, one can download (or clone) the entire collection upfront. To configure an application, please open the config folder directly or use the documentation and then follow the build recipe of the desired application or library.

Documentation

- **ReadtheDocs**: online documentation with full text search: CP2K, ELPA, LIBINT, LIBXC, LIBXSMM, QE, and TF.
- **PDF**: a single documentation file.

Related Projects

- Spack Package Manager: [http://computation.llnl.gov/projects/spack-hpc-package-manager%5D(http://computation.llnl.gov/projects/spack-hpc-package-manager)
- EasyBuild / EasyConfig (University of Gent): [https://github.com/easybuilders%5D(https://github.com/easybuilders)

Please note that XCONFIGURE has a narrower scope when compared to the above package managers.

Applications

CP2K

This document describes building CP2K with several (optional) libraries, which may be beneficial for functionality or performance.

- **LIBXSMM** targets DBCSR, DBM/DBT, GRID, and other components
- **LIBINT** enables a wide range of workloads (almost necessary)
- **LIBXC** enables exchange-correlation functionals for DFT
- **MPI** is auto-detected (Intel MPI and OpenMPI supported)
- **MKL** or Intel Math Kernel Library (also per Linux distro's package manager):
  - Provides LAPACK/BLAS and ScaLAPACK library
  - Provides FFTw library
- ELPA for matrix diagonalization

For functionality and performance, all **bold** dependencies are almost necessary or highly recommended. For instance, LIBXSMM (see also [https://libxsmm.readthedocs.io%5D(https://libxsmm.readthedocs.io)) has been incorporated since CP2K 3.0 and enables high performance in several components of CP2K. The ELPA library eventually improves the performance over ScaLAPACK (also refer to `PREFERRED_DIAG_LIBRARY` in CP2K's Input Reference).

Note: CP2K's `install_cp2k_toolchain.sh` (under `tools/toolchain`) as well as CMake based builds are out of scope in this document (see official guide).

For CP2K, XCONFIGURE provides a collection of helper scripts, optional patches, and an ARCH-file supporting:

- GNU Compiler Collection (default if `ifx` or `ifort` are not available, explicit with `make ARCH=Linux-x86-64-intelx VERSION=p`
- Intel Compiler (default if `ifx` is available, explicit with `make ARCH=Linux-x86-64-intelx VERSION=psmp INTEL=2`)

Note: Intel Classic Compiler is used by default if `ifort` is available but `ifx` or `gfortran` are not available, and it can be explicitly selected with `make ARCH=Linux-x86-64-intelx VERSION=psmp INTEL=1`).

Step-by-step Guide

This step-by-step guide assumes the GNU Compiler Collection (GNU Fortran), Intel MPI and Intel MKL as prerequisites (adding an Intel repository to the Linux distribution's package manager, installing Intel MKL and Intel MPI or supporting an HPC fabric is out of scope in this document. is not subject of this document). Building LIBXSMM, LIBINT, and LIBXC is part of the steps.

Note: in an offline-environment, it is best to download the entire XCONFIGURE project upfront and upload to the target system. Offline limitations can be worked around and overcome with additional steps. This step-by-step guide assumes Internet connectivity.

1) First, please download `configure-get.sh` to any location and make the prerequisites available (GNU Compiler Collection, Intel MPI and Intel MKL):

```
wget https://github.com/hfp/xconfigure/raw/main/configure-get.sh
chmod +x configure-get.sh

source /opt/intel/oneapi/mpi/latest/env/vars.sh
source /opt/intel/oneapi/mkl/latest/env/vars.sh
```

For the following steps, it is necessary to place LIBINT, LIBXC, LIBXSMM, and CP2K into a common directory (`$HOME` is assumed).

2) The second step builds a LIBINT which is already preconfigured for CP2K. To fully bootstrap LIBINT is out of scope for this step.

```
cd $HOME && curl -s https://api.github.com/repos/cp2k/libint-cp2k/releases/latest \
| grep "browser_download_url" | grep "lmax-6" \
| sed "s/..*: \"\(..*[^\"]\)\".*/url \1/" \
| curl -LOK-
```

A rate limit applies to anonymous GitHub API requests of the same origin. If the download fails, it can be worth trying an authenticated request relying on a GitHub account (`-u "user:password"`).

```
cd $HOME && tar xvf libint-v2.6.0-cp2k-lmax-6.tgz
rm libint-v2.6.0-cp2k-lmax-6.tgz
cd libint-v2.6.0-cp2k-lmax-6

/path/to/configure-get.sh libint
./configure-libint-gnu.sh

make -j $(nproc)
make install
make distclean
```

There can be issues about target flags requiring a build-system able to execute compiled binaries. To avoid cross-compilation (not supported here), please rely on a build-host matching the capabilities of the target system.

3) The third step builds LIBXC.

```
cd $HOME && wget https://gitlab.com/libxc/libxc/-/archive/6.2.2/libxc-6.2.2.tar.bz2
tar xvf libxc-6.2.2.tar.bz2

rm libxc-6.2.2.tar.bz2
cd libxc-6.2.2

/path/to/configure-get.sh libxc
./configure-libxc-gnu.sh

make -j $(nproc)
make install
make distclean
```

During configuration, please disregard any messages suggesting `libtoolize --force`.

4) The fourth step makes LIBXSMM available, which is compiled as part of the last step.

```
##cd $HOME && git clone https://github.com/libxsmm/libxsmm.git && cd libxsmm && git checkout develop
cd $HOME && wget https://github.com/libxsmm/libxsmm/archive/refs/heads/develop.zip
mkdir libxsmm && cd libxsmm && unzip $HOME/libxsmm-develop.zip
##make GNU=1 -j $(nproc)
```

It can be useful to build LIBXSMM also in a separate fashion (see last/commented line above). This can be useful for building a stand-alone reproducer in DBCSR's GPU backend as well as CP2K's DBM reproducer.

5) This last step builds CP2K and LIBXSMM inside of CP2K's source directory. A serial version `VERSION=ssmp` as opposed to `VERSION=psmp` is possible, however OpenMP remains a requirement of CP2K's code base.

Downloading GitHub-generated assets from from [https://github.com/cp2k/cp2k/releases%5D(https://github.com/cp2k/cp2k/releases) like "*Source code (zip)*" or "*Source code (tar.gz)*" will miss submodules which are subsequently missed when building CP2K.

```
cd $HOME && git clone https://github.com/cp2k/cp2k.git
cd $HOME/cp2k && git pull && git submodule update --init --recursive
cd $HOME/cp2k/exts/dbcsr && git checkout develop && git pull

cd $HOME/cp2k && /path/to/configure-get.sh cp2k
```

Applying XCONFIGURE for CP2K in an offline-environment is out of scope here, but one can `cp /path/to/xconfigure/config/c` `cp /path/to/xconfigure/config/cp2k/Linux-x86-64-intelx.* /path/to/cp2k/arch`, and eventually `git apply cpassert.git.diff`.

Building CP2K proceeds with:

```
rm -rf exe lib obj
make -j $(nproc) \
  ARCH=Linux-x86-64-intelx VERSION=psmp cp2k \
  GNU=1
```

The initial output of the build looks like:

```
Discovering programs ...
================================================================================
Using the following libraries:
LIBXSMMROOT=/path/to/libxsmm
LIBINTROOT=/path/to/libint/gnu
LIBXCROOT=/path/to/libxc/gnu
================================================================================
LIBXSMM develop (Linux)
--------------------------------------------------------------------------------
```

Once the build is completed, the CP2K executable is ready (`exe/Linux-x86-64-intelx/cp2k.psmp`):

```
$ LIBXSMM_VERBOSE=1 exe/Linux-x86-64-intelx/cp2k.psmp
  [...]
  LIBXSMM_VERSION: develop
  LIBXSMM_TARGET: spr
```

The ARCH-file attempts to auto-detect optional libraries using `I_MPI_ROOT`, `MKLROOT` environment variables as well as searching certain standard locations. LIBXSMM, LIBINT, and LIBXC are expected in directories parallel to CP2K's root directory. In general, build-keys such as `LIBXSMMROOT`, `LIBINTROOT`, `LIBXCROOT`, `ELPAROOT`, and others are supported.

CP2K on GPUs

Please simply apply `USE_ACCEL=opencl` (like `USE_ACCEL=cuda` for CUDA) to XCONFIGURE's build instructions. The OpenCL support enables DBCSR's OpenCL backend as well as CP2K's GPU-enabled DBM/DBT component.

Further, DBCSR can be built stand-alone and used to exercise and test GPU accleration as well, which is not subject of XCONFIGURE. Further, within DBCSR some driver code exists to exercise GPU performance in a stand-alone fashion (does not even rely on DBCSR's build system; see DBCSR ACCelerator Interface). The OpenCL backend in DBCSR provides pretuned kernels for CP2K. Similarly, CP2K's DBM component (`/path/to/cp2k/src/dbm`) can be built and exercised in a stand-alone fashion.

The OpenCL backend has comprehensive runtime-control by the means of environment variables. This can be used to assign OpenCL devices, to aggregate sub-devices (devices are split into sub-devices by default), to extract kernel shapes used by a specific workload, and to subsequently tune specific kernels.

Running CP2K

Running CP2K may go beyond a single node, and pinning processes and threads becomes even more important. There are several schemes available. As a rule of thumb, a high rank-count for lower node-counts may yield best results unless the workload is very memory intensive. In the latter case, lowering the number of MPI-ranks per node is effective especially if a larger amount of memory is replicated rather than partitioned by the rank-count. In contrast (communication bound), a lower rank count for multi-node computations may be desired. To ease running CP2K, there are a number of supportive scripts provided by XCONFIGURE: `plan.sh` (see here), `run.sh` (see here), and `info.sh`.

As soon as several experiments are finished, it becomes handy to summarize the log-output. For this case, an info-script (`info.sh`) is available attempting to present a table (summary of all results), which is generated from log files (`.txt` and `.out` extension by default). Log files can be captured with `tee`, or the output is captured by the job scheduler.

```
./run.sh benchmarks/QS/H2O-64.inp | tee cp2k-h2o64-20240725b.txt
ls -1 *.txt
cp2k-h2o64-20240725a.txt
cp2k-h2o64-20240725b.txt

./info.sh [-best] [/path/to/logs]
H2O-64                 Nodes R/N T/R Cases/d Seconds
cp2k-h2o64-20240725a 2       32   4     807 107.237
cp2k-h2o64-20240725b 4       16   8     872  99.962
```

Note: the "*Cases/d*" metric is calculated with integer arithmetic and hence represents fully completed cases per day (based on 86400 seconds per day). The number of seconds (as shown) is end-to-end (wall time), i.e., total time to solution including any (sequential) phase (initialization, etc.). Performance is higher if the workload requires more iterations (some publications present a metric based on iteration time).

Sanity Check

There is nothing that can replace the full regression test suite. However, to quickly check whether a build is sane or not, one can run for instance `benchmarks/QS/H2O-64.inp` and check if the SCF iteration prints like the following:

```
  Step     Update method      Time    Convergence         Total energy    Change
  ------------------------------------------------------------------------------
     1 OT DIIS     0.15E+00    0.5     0.01337191     -1059.6804814927 -1.06E+03
     2 OT DIIS     0.15E+00    0.3     0.00866338     -1073.3635678409 -1.37E+01
     3 OT DIIS     0.15E+00    0.3     0.00615351     -1082.2282197787 -8.86E+00
     4 OT DIIS     0.15E+00    0.3     0.00431587     -1088.6720379505 -6.44E+00
     5 OT DIIS     0.15E+00    0.3     0.00329037     -1092.3459788564 -3.67E+00
     6 OT DIIS     0.15E+00    0.3     0.00250764     -1095.1407783214 -2.79E+00
     7 OT DIIS     0.15E+00    0.3     0.00187043     -1097.2047924571 -2.06E+00
     8 OT DIIS     0.15E+00    0.3     0.00144439     -1098.4309205383 -1.23E+00
     9 OT DIIS     0.15E+00    0.3     0.00112474     -1099.2105625375 -7.80E-01
    10 OT DIIS     0.15E+00    0.3     0.00101434     -1099.5709299131 -3.60E-01
    [...]
```

The column called "*Convergence*" must monotonically converge towards zero.

References

[https://nholmber.github.io/2017/04/cp2k-build-cray-xc40/%5D(https://nholmber.github.io/2017/04/cp2k-build-cray-xc40/)
[https://xconfigure.readthedocs.io/cp2k/plan/%5D(https://xconfigure.readthedocs.io/cp2k/plan/)
[https://www.cp2k.org/static/downloads%5D(https://www.cp2k.org/static/downloads)
[https://www.cp2k.org/howto:compile%5D(https://www.cp2k.org/howto:compile)

ELPA

Build Instructions

ELPA 2021 Download and unpack ELPA and make the configure wrapper scripts available in ELPA's root folder. Consider CP2K's download area (cache) as an alternative source for downloading ELPA.

```
echo "wget --content-disposition --no-check-certificate https://www.cp2k.org/static/downloads/elpa-202
wget --content-disposition --no-check-certificate https://elpa.mpcdf.mpg.de/software/tarball-archive/R
tar xvf elpa-2021.11.002.tar.gz
cd elpa-2021.11.002

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh elpa
```

Please make the Intel Compiler and Intel MKL available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make clean
./configure-elpa-skx-omp.sh
make -j ; make install

make clean
./configure-elpa-skx.sh
make -j ; make install
```

Even if ELPA was just unpacked (and never built before), `make clean` is recommended in advance of building ELPA ("unknown module file format"). After building and installing the desired configuration(s), one may have a look at the installation:

```
[user@system elpa-2021.11.002]$ ls ../elpa
 intel-skx
 intel-skx-omp
```

For different targets (instruction set extensions) or for different versions of the Intel Compiler, the configure scripts support an additional argument ("default" is the default tagname):

```
./configure-elpa-skx-omp.sh tagname
```

As shown above, an arbitrary "tagname" can be given (without editing the script). This might be used to build multiple variants of the ELPA library.

ELPA 2020 Download and unpack ELPA and make the configure wrapper scripts available in ELPA's root folder. Consider CP2K's download area (cache) as an alternative source for downloading ELPA.

```
echo "wget --content-disposition --no-check-certificate https://elpa.mpcdf.mpg.de/software/tarball-arc
wget --content-disposition --no-check-certificate https://www.cp2k.org/static/downloads/elpa-2020.11.0
tar xvf elpa-2020.11.001.tar.gz
cd elpa-2020.11.001

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh elpa
```

Please make the Intel Compiler and Intel MKL available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make clean
./configure-elpa-skx-omp.sh
make -j ; make install

make clean
./configure-elpa-skx.sh
make -j ; make install
```

Even if ELPA was just unpacked (and never built before), `make clean` is recommended in advance of building ELPA ("unknown module file format"). After building and installing the desired configuration(s), one may have a look at the installation:

```
[user@system elpa-2020.11.001]$ ls ../elpa
 intel-skx
 intel-skx-omp
```

ELPA 2019 Download and unpack ELPA and make the configure wrapper scripts available in ELPA's root folder.

```
wget --content-disposition --no-check-certificate https://elpa.mpcdf.mpg.de/software/tarball-archive/R
tar xvf elpa-2019.11.001.tar.gz
cd elpa-2019.11.001

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh elpa
```

Please make the Intel Compiler and Intel MKL available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make clean
./configure-elpa-skx-omp.sh
make -j ; make install

make clean
./configure-elpa-skx.sh
make -j ; make install
```

Even if ELPA was just unpacked (and never built before), `make clean` is recommended in advance of building ELPA ("unknown module file format"). After building and installing the desired configuration(s), one may have a look at the installation:

```
[user@system elpa-2019.11.001]$ ls ../elpa
 intel-skx
 intel-skx-omp
```

ELPA 2018 Please use ELPA 2017.11.001 for CP2K 6.1. For CP2K 7.1, please rely on ELPA 2019. ELPA 2018 **fails or crashes in several regression tests** in CP2K (certain rank-counts produce an incorrect decomposition), and hence ELPA 2018 should be avoided in production.

ELPA 2017 Download and unpack ELPA and make the configure wrapper scripts available in ELPA's root folder. It is recommended to package the state (Tarball or similar), which is achieved after downloading the wrapper scripts.

Note: In Quantum Espresso, the ___ELPA_2018 interface must be used for ELPA 2017.11 (`-D__ELPA_2018`). The ___ELPA_2017 preprocessor definition triggers the ELPA1 legacy interface (get_elpa_row_col_comms, etc.), which was removed in ELPA 2017.11. This is already considered when using XCONFIGURE's wrapper scripts.

```
wget --content-disposition --no-check-certificate https://elpa.mpcdf.mpg.de/software/tarball-archive/R
tar xvf elpa-2017.11.001.tar.gz
cd elpa-2017.11.001

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh elpa
```

Please make the Intel Compiler and Intel MKL available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make clean
./configure-elpa-skx-omp.sh
make -j ; make install

make clean
./configure-elpa-skx.sh
make -j ; make install
```

Even if ELPA was just unpacked (and never built before), `make clean` is recommended in advance of building ELPA ("unknown module file format").

References

[https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2l-elpa-optional-improved-performance-for-diagonalization%5D(https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2l-elpa-optional-improved-performance-for-diagonalization)
[https://elpa.mpcdf.mpg.de/software/tarball-archive/ELPA_TARBALL_ARCHIVE.html%5D(https://elpa.mpcdf.mpg.de/software/tarball-archive/ELPA_TARBALL_ARCHIVE.html)
[https://www.cp2k.org/static/downloads%5D(https://www.cp2k.org/static/downloads)

HDF5

To download, configure, build, and install HDF5, one may proceed as shown below.

```
wget --content-disposition --no-check-certificate https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-
tar xvf hdf5-1.12.1.tar.bz2
cd hdf5-1.12.1

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh hdf5
```

Please make the intended compiler available on the command line. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make distclean
./configure-hdf5-skx.sh
make -j; make install
```

References

[https://support.hdfgroup.org/ftp/HDF5/releases/%5D(https://support.hdfgroup.org/ftp/HDF5/releases/) [https://hdfgroup.org/%5D(https://hdfgroup.org/)

LIBINT

Overview

For CP2K 6.1 (and earlier), LIBINT 1.1.x is required (1.2.x, 2.x, or any later version cannot be used). For CP2K 7.x and onwards, LIBINT 2.5 (or later) is needed.

Please make the Intel Compiler available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

Note: CP2K 6.1 (and earlier) depends on LIBINT 1.1.x and a newer version of LIBINT cannot be used! CP2K 7.x (and later) rely on LIBINT 2.5 (or later) and cannot use the preconfigured library as provided on LIBINT's home page.

Version 2.5 (and later)

LIBINT generates code according to a configuration and an extent that is often specific to the application. The downloads from LIBINT's home page are not configured for CP2K and hence cannot be used. Please download (take "lmax-6" if unsure), unpack LIBINT, and make the configure wrapper scripts available in LIBINT's root folder.

To just determine the download-URL of the latest version (a suitable variant can be "lmax-6"):

```
curl -s https://api.github.com/repos/cp2k/libint-cp2k/releases/latest \
| grep "browser_download_url" | grep "lmax-6" \
| sed "s/..*: \"\(..*[^\"]\)\".*/\1/"
```

To download the lmax6-version right away, run the following command:

```
curl -s https://api.github.com/repos/cp2k/libint-cp2k/releases/latest \
| grep "browser_download_url" | grep "lmax-6" \
| sed "s/..*: \"\(..*[^\"]\)\".*/url \1/" \
| curl -LOK-
```

Note: A rate limit applies to GitHub API requests of the same origin. If the download fails, it can be worth trying an authenticated request by using a GitHub account (`-u "user:password"`).

To unpack the archive and to download the configure wrapper (lmax6-version is assumed):

```
tar xvf libint-v2.6.0-cp2k-lmax-6.tgz
cd libint-v2.6.0-cp2k-lmax-6

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh libint
```

There are spurious issues about specific target flags requiring a build-system able to execute compiled binaries. To avoid cross-compilation (not supported here), please rely on a build system that matches the target system. For example, to configure and make for an Intel Xeon Scalable processor such as "Cascadelake" or "Skylake" server ("SKX"):

```
make distclean
./configure-libint-skx.sh
make -j $(nproc); make install
```

Make sure to run `make distclean` before reconfiguring a different variant, e.g., GNU and Intel variant. Further, for different targets (instruction set extensions) or different compilers, the configure-wrapper scripts support an additional argument ("default" is the default tagname):

```
./configure-libint-hsw.sh tagname
```

As shown above, an arbitrary "tagname" can be given (without editing the script). This might be used to build multiple variants of the LIBINT library.

Version 1.x

Download and unpack LIBINT and make the configure wrapper scripts available in LIBINT's root folder. Please note that the "automake" package is a prerequisite.

```
wget --content-disposition --no-check-certificate https://github.com/evaleev/libint/archive/release-1-
tar xvf release-1-1-6.tar.gz
cd libint-release-1-1-6

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh libint
```

For example, to configure and make for an Intel Xeon E5v4 processor (formerly codenamed "Broadwell"):

```
make distclean
./configure-libint-hsw.sh
make -j $(nproc); make install
```

The version 1.x line of LIBINT does not support to cross-compile for an architecture. If cross-compilation is necessary, one can rely on the Intel Software Development Emulator (Intel SDE) to compile LIBINT for targets, which cannot execute on the compile-host.

```
/software/intel/sde/sde -knl -- make
```

To speed-up compilation, "make" might be carried out in phases: after "printing the code" (c-files), the make execution continues with building the object-file where no SDE needed. The latter phase can be sped up by interrupting "make" and executing it without SDE. The root cause of the entire problem is that the driver printing the c-code is (needlessly) compiled using the architecture-flags that are not supported on the host.

Boostrap for CP2K

LIBINT consists of a compiler specializing the library by generating source files according to the needs of the desired application:

```
wget https://github.com/evaleev/libint/archive/refs/tags/v2.9.0.tar.gz
tar xvf v2.9.0.tar.gz && rm v2.9.0.tar.gz

cd libint-2.9.0 && ./autogen.sh && ./configure \
  --enable-eri=1 --enable-eri2=1 --enable-eri3=1 --with-max-am=6 \
  --with-eri-max-am=6,5 --with-eri2-max-am=8,7 --with-eri3-max-am=8,7 --with-opt-am=3 \
  --with-libint-exportdir=libint-cp2k --disable-unrolling --enable-fma \
  --with-real-type=libint2::simd::VectorAVXDouble \
  --with-cxxgen-optflags="-march=native -mtune=native"

make -j $(nproc) export
make clean
```

Note: for example `VectorAVXDouble` is permitted by `--with-cxxgen-optflags` according to CPU features on the system bootstrapping LIBINT. This requires special care to avoid discrepancies between the compilation host and the desired target system (cross-compilation).

References

[https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2g-libint-optional-enables-methods-including-hf-exchange%5D(https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2g-libint-optional-enables-methods-including-hf-exchange)
[https://github.com/evaleev/libint/releases/tag/release-1-1-6%5D(https://github.com/evaleev/libint/releases/tag/release-1-1-6)
[https://github.com/cp2k/libint-cp2k/releases/latest%5D(https://github.com/cp2k/libint-cp2k/releases/latest)

Med File Library (libmed)

To download, configure, build, and install libmed, one may proceed as shown below.

```
wget --content-disposition --no-check-certificate https://files.salome-platform.org/Salome/other/med-4
tar xvf med-4.1.0.tar.gz
cd med-4.1.0

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh libmed
```

Please make the intended compiler available on the command line. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

Note: Please make the "hdf5-tools" command available, or pay attention to the console output after configuring libmed. In general, an HDF5 development package is necessary to pass the default configuration as implemented by XCONFIGURE. One can adjust the configure wrapper script for custom-built HDF5 by pointing to an (non-default) installation location.

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make distclean
./configure-med-skx.sh
make -j; make install
```

References

[https://salome-platform.org/downloads/%5D(https://salome-platform.org/downloads/)
[http://wiki.opentelemac.org/doku.php?id=installation_linux_med%5D(http://wiki.opentelemac.org/doku.php?id=installation_linux_med)
[http://opentelemac.org/%5D(http://opentelemac.org/)

LIBXC

To download, configure, build, and install LIBXC 2.x, 3.x (CP2K 5.1 and earlier is only compatible with LIBXC 3.0 or earlier), 4.x (CP2K 7.1 and earlier is only compatible with LIBXC 4.x), or 5.x (CP2K 8.1 and later require LIBXC 5.x), one may proceed as shown below. For CP2K, see also How to compile the CP2K code). In general, only the latest major release of LIBXC (by the time of the CP2K-release) is supported.

```
wget --content-disposition --no-check-certificate https://www.tddft.org/programs/libxc/down.php?file=6
tar xvf libxc-6.2.2.tar.gz
cd libxc-6.2.2

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh libxc
```

Please make the Intel Compiler available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make distclean
./configure-libxc-skx.sh
make -j; make install
```

Note: Please disregard messages during configuration suggesting `libtoolize --force`.

References

[https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2k-libxc-optional-wider-choice-of-xc-functionals%5D (https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2k-libxc-optional-wider-choice-of-xc-functionals)

LIBXSMM

LIBXSMM is a library targeting Intel Architecture (x86) for small, dense or sparse matrix multiplications, and small convolutions. The build instructions can be found at [https://github.com/hfp/libxsmm%5D(https://libxsmm.readthedocs.io/#build-instructions) (PDF).

METIS

To download, configure, build, and install METIS, one may proceed as shown below.

```
wget --content-disposition --no-check-certificate http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/meti
tar xvf metis-5.1.0.tar.gz
cd metis-5.1.0

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh metis
```

Please make the intended compiler available on the command line. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make distclean
./configure-metis-skx.sh
make -j; make install
```

References

[http://glaros.dtc.umn.edu/gkhome/metis/metis/download%5D(http://glaros.dtc.umn.edu/gkhome/metis/metis/download)
[http://glaros.dtc.umn.edu/gkhome/views/metis%5D(http://glaros.dtc.umn.edu/gkhome/views/metis)

Plumed

To download, configure, build, and install Plumed 2.x (CP2K requires Plumed2), one may proceed as shown below. See also How to compile CP2K with Plumed.

```
wget --content-disposition --no-check-certificate https://github.com/plumed/plumed2/archive/v2.8.0.tar
tar xvf plumed2-2.8.0.tar.gz
cd plumed2-2.8.0

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh plumed
```

Please make the intended compiler available on the command line. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

Note: Please make the "python" command available, which may point to Python2 or Python3. For example, create a `bin` directory at `$HOME` (`mkdir -p ${HOME}/bin`), and create a symbolic link to either Python2 or Python3 (e.g., `ln -s /usr/bin/python3 ${HOME}/bin/python`).

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make distclean
./configure-plumed-skx.sh
make -j; make install
```

References

[https://github.com/plumed/plumed2/releases/latest%5D(https://github.com/plumed/plumed2/releases/latest)
[https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2o-plumed-optional-enables-various-enhanced-sampling-methods%5D(https://github.com/cp2k/cp2k/blob/master/INSTALL.md#2o-plumed-optional-enables-various-enhanced-sampling-methods)
[https://www.cp2k.org/howto:install_with_plumed%5D(https://www.cp2k.org/howto:install_with_plumed)
[https://www.plumed.org/%5D(https://www.plumed.org/)

QE

Build Instructions

Download, unpack Quantum Espresso and make the configure wrapper scripts available in QE's root folder. Please note that the configure wrapper scripts support QE 6.x (prior support for 5.x is dropped). Before building QE, one needs to complete the recipe for ELPA.

Note: the ELPA configuration must correspond to the desired QE configuration, e.g., `configure-elpa-skx-omp.sh` and `configure-qe-skx-omp.sh` ("omp").

```
wget https://gitlab.com/QEF/q-e/-/archive/qe-6.6/q-e-qe-6.6.tar.bz2
tar xvf q-e-qe-6.6.tar.bz2
cd q-e-qe-6.6

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh qe
```

Please make the Intel Compiler available on the command line, which may vary with the computing environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, configure for an Intel Xeon Scalable Processor (applicable to CPUs previously codenamed "Skylake" and "Cascadelake" server), and build the desired application(s), e.g., "pw", "cp", or "all".

```
./configure-qe-skx-omp.sh
make pw -j
```

Building "all" (or `make` without target argument) requires repeating `make all` until no compilation error occurs. This is because of some incorrect build dependencies (build order issue which might have been introduced by the configure wrapper scripts). In case of starting over, one can run `make distclean`, reconfigure the application, and build it again. For different targets (instruction set extensions) or different versions of the Intel Compiler, the configure scripts support an additional argument ("default" is the default tagname):

```
./configure-qe-skx-omp.sh tagname
```

As shown above, an arbitrary "tagname" can be given (without editing the script). This might be used to build multiple variants of QE. Please note: this tagname also selects the corresponding ELPA library (or should match the tagname used to build ELPA). Make sure to save your current QE build before building an additional variant!

Run Instructions

To run Quantum Espresso in an optimal fashion depends on the workload and on the "parallelization levels", which can be exploited by the workload in question. These parallelization levels apply to execution phases (or major algorithms) rather than staying in a hierarchical relationship (levels). It is recommended to read some of the primary references explaining these parallelization levels (a number of them can be found in the Internet including some presentation slides). Time to solution may *vary by factors* depending on whether these levels are orchestrated or not. To specify these levels, one uses command line arguments along with the QE executable(s):

- `-npool`: try to maximize the number of pools. The number depends on the workload, e.g., if the number of k-points can be distributed among independent pools. Indeed, per trial-and-error it is rather quick to check if a workload fails to pass the initialization phase. One may use prime numbers: *2*, *3*, *5*, etc. (default is *1*). For example, when *npool=2* worked it might be worth trying *npool=4*. On the other hand, increasing the number pools duplicates the memory consumption accordingly (larger numbers are increasingly unlikely to work).
- `-ndiag`: this number determines the number of ranks per pool used for dense linear algebra operations (DGEMM and ZGEMM). For example, if *64* ranks are used in total per node and *npool=2*, then put *ndiag=32* (QE selects the next square number which is less equal than the given number, e.g., *ndiag=25* in the previous example).
- `-ntg`: specifies the number of tasks groups per pool being used for, e.g., FFTs. One can start with `NTG=$((NUMNODES*NRANKS/(NPOOL*2)))`. If `NTG` becomes zero, `NTG=${NRANKS}` should be used (number of ranks per node). Please note the given formula is only a rule of thumb, and the number of task groups also depends on the number of ranks as the workload is scaled out.

To run QE, below command line can be a starting point ("numbers" are presented as Shell variables to better understand the inner mechanics). Important for hybrid builds (MPI and OpenMP together) are the given environment variables. The `KMP_AFFINITY` assumes Hyperthreading (SMT) is enabled (granularity=fine), and the "scatter" policy allows to easily run less than the maximum number of Hyperthreads per core. As a rule of thumb, OpenMP adds only little overhead (often not worth a pure MPI application) but allows to scale further out when compared to pure MPI builds.

```
mpirun -bootstrap ssh -genvall \
  -np $((NRANKS_PER_NODE*NUMNODES)) -perhost ${NRANKS} \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv KMP_AFFINITY=compact,granularity=fine,1 \
  -genv OMP_NUM_THREADS=${NTHREADS_PER_RANK} \
  /path/to/pw.x \<command-line-arguments\>
```

Performance

An info-script (`info.sh`) is available attempting to present a table (summary of all results), which is generated from log files (use `tee`, or rely on the output of the job scheduler). There are only certain file extensions supported (`.txt`, `.log`). If no file matches, then all files (independent of the file extension) are attempted to be parsed (which will go wrong eventually). For legacy reasons (run command is not part of the log, etc.), certain schemes for the filename are eventually parsed and translated as well.

```
./run-qe.sh | tee qe-asrf112-4x16x1.txt
ls -1 *.txt
qe-asrf112-2x32x1.txt
qe-asrf112-4x16x1.txt

./info.sh [-best] /path/to/logs-or-cwd
AUSURF112          Nodes R/N T/R Cases/d Seconds NPOOL NDIAG NTG
qe-asrf112-2x32x1 2       32   2     533  162.35     2    25  32
qe-asrf112-4x16x1 4       16   4     714  121.82     2    25  32
```

Please note that the number of cases per day (Cases/d) are currently calculated with integer arithmetic and eventually lower than just rounding down (based on 86400 seconds per day). The number of seconds taken are end-to-end (wall time), i.e., total time to solution including any (sequential) phase (initialization, etc.). Performance is higher if the workload requires more iterations (some publications present a metric based on iteration time).

References

[https://software.intel.com/en-us/articles/quantum-espresso-for-the-intel-xeon-phi-processor%5D(https://software.intel.com/en-us/articles/quantum-espresso-for-the-intel-xeon-phi-processor)
[https://www.quantum-espresso.org/Doc/user_guide/node1.html%5D(https://www.quantum-espresso.org/Doc/user_guide/node1.html)

TensorFlow™ with LIBXSMM

There is a recipe available for TensorFlow with LIBXSMM (PDF). However, the recipe also contains information about building TensorFlow with Intel MKL and MKL-DNN (see section about Performance Tuning).

TensorFlow Serving

For experimentation, there is a recipe available for TensorFlow Serving with LIBXSMM. Please note this recipe is likely outdated and not intended for production use.

Vc: SIMD Vector Classes for C++

To download, configure, build, and install Vc, one may proceed as shown below.

```
wget --content-disposition --no-check-certificate https://github.com/VcDevel/Vc/archive/refs/tags/1.4.
tar xvf Vc-1.4.2.tar.gz
cd Vc-1.4.2

wget --content-disposition --no-check-certificate https://github.com/hfp/xconfigure/raw/main/configure
chmod +x configure-get.sh
./configure-get.sh vc
```

Please make the intended compiler available on the command line. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2020.4.304/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make distclean
./configure-vc.sh
cd build; make -j; make install
```

References

[https://github.com/VcDevel/Vc/releases%5D(https://github.com/VcDevel/Vc/releases)
[https://github.com/VcDevel/Vc%5D(https://github.com/VcDevel/Vc)

Appendix

CP2K MPI/OpenMP-hybrid Execution (PSMP)

Overview

CP2K's grid-based calculation as well as DBCSR's block sparse matrix multiplication (Cannon algorithm) prefer a square-number for the total rank-count (2d communication pattern). This is not to be obfuscated with a Power-of-Two (POT) rank-count that usually leads to trivial work distribution (MPI).

It can be more efficient to leave CPU-cores unused in order to achieve this square-number property rather than using all cores with a "non-preferred" total rank-count (sometimes a frequency upside over an "all-core turbo" emphasizes this property further). Counter-intuitively, even an unbalanced rank-count per node i.e., different rank-counts per socket can be an advantage. Pinning MPI processes and placing threads requires extra care to be taken on a per-node basis to load a dual-socket system in a balanced fashion or to setup space between ranks for the OpenMP threads.

Because of the above-mentioned complexity, a script for planning MPI/OpenMP-hybrid execution (`plan.sh`) is available. Here is a first example for running the PSMP-binary on an SMP-enabled (Hyperthreads) dual-socket system with 24 cores per processor/socket (96 hardware threads in total). At first, a run with 48 ranks and 2 threads per core comes to the mind (`48x2`). However, for instance 16 ranks with 6 threads per rank can be better for performance (`16x6`). To easily place the ranks, Intel MPI is used:

```
mpirun -np 16 \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv OMP_PLACES=threads -genv OMP_PROC_BIND=SPREAD \
  -genv OMP_NUM_THREADS=6 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

Note: For hybrid codes, `I_MPI_PIN_DOMAIN=auto` is recommended as it spaces the ranks according to the number of OpenMP threads (`OMP_NUM_THREADS`). It is not necessary and not recommended to build a rather complicated `I_MPI_PIN_PROCESSOR_LIST` for hybrid codes (MPI plus OpenMP). To display and to log the pinning and thread affinization at the startup of an application, `I_MPI_DEBUG=4` can be used with no performance penalty. The recommended `I_MPI_PIN_ORDER=bunch` ensures that ranks per node are split as even as possible with respect to sockets (e.g., 36 ranks on a 2x20-core system are put in 2x18 ranks instead of 20+16 ranks).

To achieve a similar placement with OpenMPI, ranks are mapped to "execution slots" (`--map-by slot`) along with specifying the number of processing elements (`PE`). By default, execution slots are counted in number of physical cores which yields `--map-by slot:PE=3` for the same system (mentioned above).

```
mpirun -np 16 --map-by slot:PE=3 \
  -x OMP_PLACES=threads -x OMP_PROC_BIND=SPREAD \
  -x OMP_NUM_THREADS=6 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

Note: Intel MPI's `I_MPI_PIN_ORDER=bunch` to balance the number of ranks between sockets (see above) appears hard to achieve with OpenMPI therefore an undersubscribed system may not be recommended. To display and to log the pinning and thread affinization at the startup of an application, `mpirun --report-bindings` can be used.

The end of the next section continues with our example and extends execution to multiple nodes of the above-mentioned system.

Plan Script

To configure the plan-script, the metric of the compute nodes can be given for future invocations so that only the node-count is required as an argument. The script's help output (`-h` or `--help`) initially shows the "system metric" of the computer the script is invoked on. For a system with 48 cores (two sockets, SMP/HT enabled), setting up the "system metric" looks like (plan.sh <num-nodes> <ncores-per-node> <nthreads-per-core> <nsockets-per-node>):

```
./plan.sh 1 48 2 2
```

The script is storing the arguments (except for the node-count) as default values for the next plan (file: `$HOME/.xconfigure-cp2k-plan`). This allows to supply the system-type once, and to plan with varying node-counts in a convenient fashion. Planning for 8 nodes of the above kind yields the following output (`plan.sh 8`):

```
================================================================================
384 cores: 8 node(s) with 2x24 core(s) per node and 2 thread(s) per core
================================================================================
[48x2]: 48 ranks per node with 2 thread(s) per rank (14% penalty)
[24x4]: 24 ranks per node with 4 thread(s) per rank (14% penalty)
[12x8]: 12 ranks per node with 8 thread(s) per rank (33% penalty)
--------------------------------------------------------------------------------
[32x3]: 32 ranks per node with 3 thread(s) per rank (34% penalty) -> 16x16
[18x5]: 18 ranks per node with 5 thread(s) per rank (25% penalty) -> 12x12
[8x12]: 8 ranks per node with 12 thread(s) per rank (0% penalty) -> 8x8
[2x48]: 2 ranks per node with 48 thread(s) per rank (0% penalty) -> 4x4
--------------------------------------------------------------------------------
```

The first group of the output displays POT-style (trivial) MPI/OpenMP configurations (penalty denotes potential communication overhead), however the second group (if present) shows rank/thread combinations with the total rank-count hitting a square number (penalty denotes waste of compute due to not filling each node). For the given example, 8 ranks per node with 12 threads per rank is chosen (`8x12`) and MPI-executed:

```
mpirun -perhost 8 -host node1,node2,node3,node4,node5,node6,node7,node8 \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch -genv I_MPI_DEBUG=4 \
  -genv OMP_PLACES=threads -genv OMP_PROC_BIND=SPREAD -genv OMP_NUM_THREADS=12 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

Note: For Intel MPI as well as OpenMPI, mpirun's host-list (`mpirun -host`) is setup with unique node-names, and this is the only style that is explained in this article. There is a competing style where nodes names are duplicated for the sake of enumerating available ranks (or "execution slots" in case of OpenMPI), which is not exercised in this article.

For OpenMPI, the quantity (per node) of the previously mentioned "execution slots" (measured in number of physical cores) are sometimes not known to OpenMPI (depends on cluster/scheduler setup). For instance, `mpirun` may be complaining about an attempt to use too many execution slots simply because OpenMPI believes all systems represent a single such slot (instead of 2x24 cores it only "sees" a single core per system). In such case, it is not recommended to "oversubscribe" the system because rank/thread affinity will likely be wrong (`mpirun --oversubscribe`). Instead, the list of unique nodes names (`-host`) may be augmented with the number of physical cores on each of the nodes (e.g., ":48" in our case).

```
mpirun -npernode 8 -host node1:48,node2:48,node3:48,node4:48,node5:48,node6:48,node7:48,node8:48 \
  --map-by slot:PE=6 --report-bindings \
  -x OMP_PLACES=threads -x OMP_PROC_BIND=SPREAD -x OMP_NUM_THREADS=12 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

Note: It can be still insufficient to augment the nodes with the expected number of slots (`:48`). If OpenMPI's mpirun is still complaining, it might be caused and solved by the job scheduler. For example, `qsub` (PBS) may be instructed with `-l select=8:mpiprocs=48` in the above case (`mpirun` in this job can use less than 48 ranks per node).

The plan-script also suggests close-by configurations (lower and higher node-counts) that can hit the square-property ("Try also the following node counts"). The example (as exercised above) was to illustrate how the script works, however it can be very helpful when running jobs especially on CPUs with not many prime factors in the core-count. Remember, the latter can be also the case for virtualized environments that reserve some of the cores to run the Hypervisor i.e., reporting less cores to the Operating System (guest OS) when compared to the physical core-count.

References

[https://github.com/hfp/xconfigure/raw/main/config/cp2k/plan.sh%5D(https://github.com/hfp/xconfigure/raw/main/config/cp2k/plan.sh)
[https://xconfigure.readthedocs.io/cp2k/%5D(https://xconfigure.readthedocs.io/cp2k/)
[https://software.intel.com/content/www/us/en/develop/articles/pinning-simulator-for-intel-mpi-library.html%5D(https://software.intel.com/content/www/us/en/develop/articles/pinning-simulator-for-intel-mpi-library.html)