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 LIBXC"
./configure-get.sh libxc 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, and LIBXSMM.
- **PDF**: a single documentation file.

Related Projects

- Spack Package Manager: http://computation.llnl.gov/projects/spack-hpc-package-manager
- EasyBuild / EasyConfig (University of Gent): 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:
  - Provides LAPACK/BLAS and ScaLAPACK library
  - Provides FFTw library

For functionality and performance, the dependencies are almost necessary or highly recommended. For instance, LIBXSMM has been incorporated since CP2K 3.0 and enables high performance in several components of CP2K. The optional ELPA library for matrix diagonalization 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 CP2K guide).

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
- Intel Compiler (default, if ifx is available, explicit with make ARCH=Linux−x86−64−intelx VERSION=psmp INTE

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. Building LIBXSMM, LIBINT, and LIBXC are part of the steps.

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

For the following steps, it is necessary to place LIBINT, LIBXC, LIBXSMM, and CP2K into a common directory ($HOME is assumed). The resulting folder structure looks like:

- libint-v2.6.0-cp2k-lmax-6
- libint/gnu
- libxc-6.2.2
- libxc/gnu
- libxsmm
- cp2k

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
```

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 is to build 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 chec
cd $HOME && wget https://github.com/libxsmm/libxsmm/archive/refs/heads/develop.zip
unzip $HOME/libxsmm-develop.zip && ln -s libxsmm-develop libxsmm
##cd $HOME/libxsmm && 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 standalone reproducer in DBCSR's GPU backend as well as CP2K's DBM reproducer.

5) This last step builds CP2K and LIBXSMM inside 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 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
```

Offline environment: no Internet connectivity is out of scope in this guide (configure-get.sh), but one can:

```
cd $HOME/cp2k
cp /path/to/xconfigure/config/cp2k/*.sh .
cp /path/to/xconfigure/config/cp2k/Linux-x86-64-intelx.* arch
## git apply cpassert.git.diff
```

Building CP2K proceeds with:

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

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 LIBXSMMMROOT, LIBINTROOT, LIBXCROOT, ELPAROOT, and others are supported. There are several other build-keys to customize the build (e.g., SYM=1, DBG=1, OPT=2, NDEBUG=0, AVX=2, or IMPI=0).

CP2K on GPUs

Please 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 standalone and used to exercise and test GPU acceleration as well, which is not subject of XCONFIGURE. Further, within DBCSR some driver code exists to exercise GPU performance in a standalone fashion (does not even rely on DBCSR's build system; see DBCSR ACCelerator Interface). The OpenCL backend in DBCSR provides tuned kernels for CP2K. Similarly, CP2K's DBM component (/path/to/cp2k/src/dbm) can be built and exercised in a standalone 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 become 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/
https://xconfigure.readthedocs.io/cp2k/plan/
https://www.cp2k.org/static/downloads
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␣https://www.cp2k.org/static/downloads/elpa−2021.11.002.
wget —content−disposition https://elpa.mpcdf.mpg.de/software/tarball−archive/Releases/2
tar xvf elpa−2021.11.002.tar.gz
cd elpa−2021.11.002

wget —content−disposition https://github.com/hfp/xconfigure/raw/main/configure−get.sh
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␣https://elpa.mpcdf.mpg.de/software/tarball-archive/Rele
wget --content-disposition https://www.cp2k.org/static/downloads/elpa-2020.11.001.tar.gz
tar xvf elpa-2020.11.001.tar.gz
cd elpa-2020.11.001

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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 https://elpa.mpcdf.mpg.de/software/tarball-archive/Releases/2
tar xvf elpa-2019.11.001.tar.gz
cd elpa-2019.11.001

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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 https://elpa.mpcdf.mpg.de/software/tarball-archive/Releases/2
tar xvf elpa-2017.11.001.tar.gz
cd elpa-2017.11.001

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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
https://elpa.mpcdf.mpg.de/software/tarball-archive/ELPA_TARBALL_ARCHIVE.html
https://www.cp2k.org/static/downloads

HDF5

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

```
wget --content-disposition https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.12/hdf5
tar xvf hdf5-1.12.1.tar.bz2
cd hdf5-1.12.1

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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/
https://hdfgroup.org/

LIBINT

Overview

LIBINT consists of a compiler specializing the library by generating source files according to the needs of the desired application. XCONFIGURE scripts support both preconfigured LIBINT as well as starting from generic source code.

After running the desired XCONFIGURE script on the generic source code, a compressed Tarball is left behind inside of the original directory (libint-cp2k-lmax6.tgz). The exported code (as generated for CP2K's requirements), can be unarchived, compiled, and installed using the XCONFIGURE scripts again (but faster because generating the source code is omitted).

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

Version 2.x

LIBINT generates code to an extent that is often specific to the application. The downloads from LIBINT's home page are not configured for CP2K, which can be handled by XCONFIGURE. LIBINT configured for CP2K, can be downloaded as well (take "lmax-6" if unsure).

To just determine the download-URL of an already configured package:

```
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 download the latest generic source code package of LIBINT (small package but full bootstrap for CP2K applies):

```
wget https://github.com/evaleev/libint/archive/refs/tags/v2.11.0.tar.gz
```

Unpack the archive of choice and download the XCONFIGURE scripts:

```
## tar xvf libint-v2.6.0-cp2k-lmax-6.tgz && cd libint-v2.6.0-cp2k-lmax-6
tar xvf v2.11.0.tar.gz && cd libint-2.11.0

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

Note: Starting with a generic source code package of LIBINT (not downloaded from https://github.com/cp2k/libint-cp2k/ but taken from https://github.com/evaleev/), BOOST libraries are required and if setup with $BOOST_ROOT, the path shall be in original format like denoting the version, e.g., boost_1_88_0.

There can be issues with target flags requiring a build-system able to execute a binary compiled with the flags of choice. 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") using, e.g., Intel Compiler:

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

To build native code for the system running the scripts using, e.g., GNU Compiler:

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

Make sure to run make distclean before reconfiguring for a different variant, e.g., changing between GNU and Intel compiler. Further, for different compiler versions, different targets (instruction set extensions), or any other difference, the configure-wrapper scripts support an additional argument (a "tagname"):

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

As shown above, an arbitrary "tagname" can be given (without editing the script). This might be useful when building 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 https://github.com/evaleev/libint/archive/release-1-1-6.tar.g
tar xvf release-1-1-6.tar.gz
cd libint-release-1-1-6

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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
```

References

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
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 https://files.salome-platform.org/Salome/other/med-4.1.0.tar.
tar xvf med-4.1.0.tar.gz
cd med-4.1.0

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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/
http://wiki.opentelemac.org/doku.php?id=installation_linux_med
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 https://www.tddft.org/programs/libxc/down.php?file=6.2.2/libx
tar xvf libxc-6.2.2.tar.gz
cd libxc-6.2.2

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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

LIBXSMM

LIBXSMM is a library for specialized dense and sparse matrix operations, and deep learning primitives. The build instructions can be found at https://github.com/libxsmm/libxsmm (PDF).

METIS

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

```
wget --content-disposition http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/metis-5.1.0.t
tar xvf metis-5.1.0.tar.gz
cd metis-5.1.0

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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
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 https://github.com/plumed/plumed2/archive/v2.8.0.tar.gz
tar xvf plumed2-2.8.0.tar.gz
cd plumed2-2.8.0

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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
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
https://www.plumed.org/

Vc: SIMD Vector Classes for C++

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

```
wget --content-disposition https://github.com/VcDevel/Vc/archive/refs/tags/1.4.2.tar.gz
tar xvf Vc-1.4.2.tar.gz
cd Vc-1.4.2

wget --content-disposition https://github.com/hfp/xconfigure/raw/main/configure-get.sh
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
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 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 set up 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 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 us 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 set up 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,
  −−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
https://xconfigure.readthedocs.io/cp2k/
https://software.intel.com/content/www/us/en/develop/articles/pinning-simulator-for-intel-mpi-library.html