CP2K Open Source Molecular Dynamics

This document is intended to be a recipe for building and running the Intel branch of CP2K, which uses the Intel Development Tools and the Intel runtime environment. Differences compared to CP2K/trunk may be incorporated into the mainline version of CP2K at any time (and subsequently released). For example, starting with CP2K 3.0 an LIBXSMM integration is available which is (optionally) substituting CP2K's "libsmm" library.

Some additional reference can found under https://groups.google.com/d/msg/cp2k/xgkJc59NKGw/U5v5FtzTBwAJ.

Getting the Source Code

The source code is hosted at GitHub and is supposed to represent the master version of CP2K in a timely fashion. CP2K's main repository is hosted at SourceForge but automatically mirrored at GitHub. The LIBXSMM library can be found under https://github.com/hfp/libxsmm.

Build Instructions

In order to build CP2K/intel from source, one may rely on Intel Compiler 16 or 17 series (the 2018 version may be supported at a later point in time). For the Intel Compiler 2017 prior to Update 4, one should source the compiler followed by sourcing a specific version of Intel MKL (to avoid an issue in Intel MKL):

```
source /opt/intel/compilers_and_libraries_2017.3.191/linux/bin/compilervars.sh intel64
source /opt/intel/compilers_and_libraries_2017.0.098/linux/mkl/bin/mklvars.sh intel64
```

Since Update 4 of the 2017-suite, the compiler and libraries can be used right away:

```
source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
```

LIBXSMM is automatically built in an out-of-tree fashion when building CP2K/intel branch. The only prerequisite is that the LIBXSMMROOT path needs to be detected (or supplied on the `make` command line). A recipe targeting "Haswell" (HSW) may look like below.

```
git clone https://github.com/hfp/libxsmm.git
git clone --branch intel https://github.com/cp2k/cp2k.git cp2k.git
ln -s cp2k.git/cp2k cp2k
cd cp2k/makefiles
make ARCH=Linux-x86-64-xintel VERSION=psmp AVX=2
```

To target for instance "Knights Landing" (KNL), use "AVX=3 MIC=1" instead of "AVX=2". Since CP2K 3.0, the mainline version (non-Intel branch) supports LIBXSMM as well. If an own ARCH file is used or prepared, the LIBXSMM library needs to be built separately and one may follow the official guide. Building LIBXSMM is rather simple (instead of the master revision, an official release can used as well):

```
git clone https://github.com/hfp/libxsmm.git
cd libxsmm ; make
```

To download and build an official CP2K release, one can still use the ARCH files that are part of the CP2K/intel branch. In this case, LIBXSMM is also built implicitly.

```
git clone https://github.com/hfp/libxsmm.git
wget https://sourceforge.net/projects/cp2k/files/cp2k-5.1.tar.bz2
tar xvf cp2k-4.1.tar.bz2
cd cp2k-4.1/arch
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-xintel.x
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-xintel.popt
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-xintel.psmp
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-xintel.sopt
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-xintel.ssmp
cd ../makefiles
source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
make ARCH=Linux-x86-64-xintel VERSION=psmp AVX=2
```

For Intel MPI, usually any version is fine. For product suites, the compiler and the MPI library are sourced in one step. To work around known issues, one may combine components from different suites. To further improve performance and versatility, one may supply LIBINTROOT, LIBXCROOT, and ELPAROOT when relying on CP2K/intel's ARCH files (see later sections about these libraries).

To further adjust CP2K at build time of the application, additional key-value pairs can be passed at make's command line (like `ARCH=Linux-x86-64-xintel` or `VERSION=psmp`).

- **SYM**: set `SYM=1` to include debug symbols into the executable e.g., helpful with performance profiling.
- **DBG**: set `DBG=1` to include debug symbols, and to generate non-optimized code.

Run Instructions

Running the application may go beyond a single node, however for first example the pinning scheme and thread affinization is introduced. As a rule of thumb, a high rank-count for single-node computation (perhaps according to the number of physical CPU cores) may be preferred. In contrast (communication bound), a lower rank count for multi-node computations may be desired. In general, CP2K prefers the total rank-count to be a square-number (two-dimensional communication pattern) rather than a Power-of-Two (POT) number.

Running an MPI/OpenMP-hybrid application, an MPI rank-count that is half the number of cores might be a good starting point (below command could be for an HT-enabled dual-socket system with 16 cores per processor and 64 hardware threads).

```
mpirun -np 16 \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv KMP_AFFINITY=compact,granularity=fine,1 \
  -genv OMP_NUM_THREADS=4 \
  cp2k/exe/Linux-x86-64-intel/cp2k.psmp workload.inp
```

For an actual workload, one may try `cp2k/tests/QS/benchmark/H2O-32.inp`, or for example the workloads under `cp2k/tests/QS/benchmark_single_node` which are supposed to fit into a single node (in fact to fit into 16 GB of memory). For the latter set of workloads (and many others), LIBINT and LIBXC may be required.

The CP2K/intel branch carries several "reconfigurations" and environment variables, which allow to adjust important runtime options. Most of these options are also accessible via the input file format (input reference e.g., https://manual.cp2k.org/trunk/CP2K_INPUT/GLOBAL/DBCSR.html).

- **CP2K_RECONFIGURE**: environment variable for reconfiguring CP2K (default depends on whether the ACCeleration layer is enabled or not). With the ACCeleration layer enabled, CP2K is reconfigured (as if CP2K_RECONFIGURE=1 is set) e.g. an increased number of entries per matrix stack is populated, and otherwise CP2K is not reconfigured. Further, setting CP2K_RECONFIGURE=0 is disabling the code specific to the Intel branch of CP2K, and relies on the (optional) LIBXSMM integration into CP2K 3.0 (and later).
- **CP2K_STACKSIZE**: environment variable which denotes the number of matrix multiplications which is collected into a single stack. Usually the internal default performs best across a variety of workloads, however depending on the workload a different value can be better. This variable is relatively impactful since the work distribution and balance is affected.
- **CP2K_HUGEPAGES**: environment variable for disabling (0) huge page based memory allocation, which is enabled by default (if TBBROOT was present at build-time of the application).
- **CP2K_RMA**: enables (1) an experimental Remote Memory Access (RMA) based multiplication algorithm (requires MPI3).
- **CP2K_SORT**: enables (1) an indirect sorting of each multiplication stack according to the C-index (experimental).

LIBINT and LIBXC

Please refer to the XCONFIGURE project (https://github.com/hfp/xconfigure), which helps to configure common HPC software for Intel software development tools.

To configure, build, and install LIBINT (version 1.1.5 and 1.1.6 have been tested), one can proceed with https://github.com/hfp/xconfigure/tree/master/config/libint. Please note there is no straightforward way to cross-compile LIBINT 1.1.x for an instruction set extension which is not supported by the compiler host. To incorporate LIBINT into CP2K, the key `LIBINTROOT=/path/to/libint` needs to be supplied when using CP2K/intel's ARCH files (make).

To configure, build, and install LIBXC (version 3.0.0 has been tested), one can proceed with https://github.com/hfp/xconfigur
To incorporate LIBXC into CP2K, the key `LIBXCROOT=/path/to/libxc` needs to be supplied when using CP2K/intel's ARCH files (make).

ELPA

Please refer to the XCONFIGURE project (https://github.com/hfp/xconfigure), which helps to configure common HPC software for Intel software development tools. To incorporate the Eigenvalue SoLvers for Petaflop-Applications (ELPA), one can proceed with https://github.com/hfp/xconfigure/tree/master/config/elpa. To incorporate ELPA

into CP2K, the key `ELPAROOT=/path/to/elpa` needs to be supplied when using CP2K/intel's ARCH files (make). The Intel-branch defaults to ELPA-2017.05 (earlier versions can rely on the ELPA key-value pair e.g., `ELPA=201611`).

```
make ARCH=Linux-x86-64-intel VERSION=psmp ELPAROOT=/path/to/elpa/default-arch
```

At runtime, a build of the Intel-branch supports an environment variable CP2K_ELPA:

- **CP2K_ELPA=-1**: requests ELPA to be enabled; the actual kernel type depends on the ELPA configuration.
- **CP2K_ELPA=0**: ELPA is not enabled by default (only on request via input file); same as non-Intel branch.
- **CP2K_ELPA**=<not-defined>: requests ELPA-kernel according to CPUID (default with CP2K/Intel-branch).

Memory Allocation

Dynamic allocation of heap memory usually requires global book keeping eventually incurring overhead in shared-memory parallel regions of an application. For this case, specialized allocation strategies are available. To use such a strategy, memory allocation wrappers can be used to replace the default memory allocation at build-time or at runtime of an application.

To use the malloc-proxy of the Intel Threading Building Blocks (Intel TBB), rely on the `TBBMALLOC=1` key-value pair at build-time of CP2K. Usually, Intel TBB is already available when sourcing the Intel development tools (one can check the TBBROOT environment variable). To use TCMALLOC as an alternative, set `TCMALLOCROOT` at build-time of CP2K by pointing to TCMALLOC's installation path (configured per `./configure --enable-minimal --prefix=<TCMALLOCROOT>`).