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 actually 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 and libraries like:

```
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
```

The above prevents an issue in Intel MKL. Since Update 4, one can source the compiler and libraries as shown below:

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

In order to target for instance "Knights Landing" (KNL), use "AVX=3 MIC=1" instead of "AVX=2". To build the CP2K application, building LIBXSMM separately is not required when relying on CP2K/intel (it will be built in an out-of-tree fashion as long as the LIBXSMMROOT path is detected or supplied). 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-intel VERSION=psmp AVX=2
```

Since CP2K 3.0, the mainline version (non-Intel branch) is also supporting LIBXSMM. 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 rely on the ARCH files as provided by the CP2K/intel branch. In this case, LIBXSMM is also built implicitly.

```
git clone https://github.com/hfp/libxsmm.git
wget http://downloads.sourceforge.net/project/cp2k/cp2k-4.1.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-intel.x
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intel.popt
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intel.psmp
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intel.sopt
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intel.ssmp
cd ../makefiles
source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
make ARCH=Linux-x86-64-intel 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 workaround 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 (similar to `ARCH=Linux-x86-64-intel` and `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.

Running the Application

Running the application may go beyond a single node, however for first example the pinning scheme and thread affinitization is introduced. Running an MPI/OpenMP-hybrid application, a number of processes (MPI ranks) which 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 KMP_AFFINITY=scatter,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 a number of "reconfigurations" and environment variables, which allow to adjust important runtime options. Most but not all of these options are also accessible via the input file format (input reference e.g., http://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**: environment variable to enable (1) an experimental Remote Memory Access (RMA) based multiplication algorithm (requires MPI3).
- **CP2K_SORT**: environment variable to enable (1) an indirect sorting of each multiplication stack according to the C-index (experimental).

LIBINT and LIBXC Dependencies

To configure, build, and install LIBINT (Version 1.1.5 and 1.1.6 has been tested), one may proceed as shown below (please note there is no easy way to cross-built the library for an instruction set extension which is not supported by the compiler host). Finally, in order to make use of LIBINT, the key `LIBINTROOT=/path/to/libint` needs to be supplied when using CP2K/intel's ARCH files (make).

```
env \
  AR=xiar CC=icc CXX=icpc \
./configure \
  --with-cxx-optflags="-O2 -xCORE-AVX2" \
  --with-cc-optflags=" -O2 -xCORE-AVX2" \
  --with-libderiv-max-am1=4 \
  --with-libint-max-am=5 \
  --prefix=$HOME/libint/default-hsw
make
make install
make realclean
```

To configure, build, and install LIBXC (Version 3.0.0 has been tested), one may proceed as shown below. To actually make use of LIBXC, the key `LIBXCROOT=/path/to/libxc` needs to be supplied when using CP2K/intel's ARCH files (make).

```
env \
  AR=xiar F77=ifort F90=ifort FC=ifort CC=icc \
  FCFLAGS="-O2 -xCORE-AVX2" \
  CFLAGS=" -O2 -xCORE-AVX2" \
./configure \
  --prefix=$HOME/libxc/default-hsw
make
make install
make clean
```

If the library needs to be cross-compiled, one may add `--host=x86_64-unknown-linux-gnu` to the command line arguments of the configure script.

Tuning

Eigenvalue SoLvers for Petaflop-Applications (ELPA)

Please refer to the XCONFIGURE project (https://github.com/hfp/xconfigure), which helps to configure common HPC software (and ELPA in particular) for Intel software development tools. To actually make use of ELPA, the key `ELPAROOT=/path/to/elpa` needs to be supplied when using CP2K/intel's ARCH files (make). For the Intel-branch, ELPA-2017.05.001 is already supported:

```
make ARCH=Linux-x86-64-intel VERSION=psmp ELPA=201705 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 Wrapper

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