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.

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.

```
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
```

Build Instructions

In order to build CP2K/intel from source, make sure to rely on one of the recommended compiler versions:

- Intel Compiler 15.0.3.187 (Build 20150407)
- Intel Compiler 16 or 17 series

For Intel MPI, usually any version is fine.

```
source /opt/intel/composer_xe_2015.3.187/bin/compilervars.sh intel64
source /opt/intel/impi/5.1.0.069/intel64/bin/mpivars.sh
```

For product suites, the compiler and the MPI library can be sourced in one step.

```
source /opt/intel/compilers_and_libraries_2016.0.109/linux/bin/compilervars.sh intel64
```

To build the CP2K application, building LIBXSMM separately is not required (it will be build in an out-of-tree fashion as long as the LIBXSMMROOT path is supplied). Since CP2K 3.0, the mainline version (non-Intel branch) is also supporting the LIBXSMM however the library needs to be built separately.

```
cd cp2k/makefiles
make ARCH=Linux-x86-64-intel VERSION=psmp LIBXSMMROOT=/path/to/libxsmm -j
```

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`).

- **LIBXSMM_PREFETCH**: set `LIBXSMM_PREFETCH=1` to enable automatic software prefetches.
- **LIBXSMM_MNK, LIBXSMM_M, LIBXSMM_N, LIBXSMM_K**: see LIBXSMM documentation.
- **JIT**: set `JIT=0` to disable JIT code generation (enabled by default), and to statically specialize LIBXSMM.
- **MPI**: set `MPI=3` to experiment with more recent MPI features e.g., with remote memory access.
- **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.

The arch-files for the versions "popt", "sopt", and "ssmp" are provided for convenience and are actually based on the "x"-configuration; these files are using even more of the above key-value pairs (`OMP`, `ACC`, etc.) inside of the arch-file but include Linux-x86-64-intel.x otherwise. Please note that the entire set of arch-files (`arch/Linux-x86-64-intel.*`) can be also used with the CP2K/master code branch (trunk) by simply copying these files into the arch file folder (Linux-x86-64-intel.x deals with this case internally).

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=compact,granularity=fine,1 \
  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 following script generalizes the example from above by scattering all ranks across the entire machine, and filling each partition (rank) with threads accordingly. The variable NT denotes the number of threads per core with MAXNT denoting whether Hyperthreading is enabled (2 or 4 depending on the kind of system) or not (1). Similarly the MAXNTHREADS describes the maximum number of hardware threads available on a per node basis.

```
MAXNTHREADS=64
NRANKS=8
MAXNT=2
NT=2

SHIFT=$((MAXNTHREADS/(NRANKS*MAXNT)*MAXNT))
NTHREADS=$((SHIFT*NT/MAXNT))

mpirun -np ${NRANKS} \
  -genv I_MPI_PROCESSOR_LIST=all:shift=${SHIFT} \
  -genv KMP_AFFINITY=scatter,granularity=fine,1 \
  -genv OMP_NUM_THREADS=${NTHREADS} \
  cp2k/exe/Linux-x86-64-intel/cp2k.psmp workload.inp
```

Please note that the threads are affinitized using the "scatter" strategy. With the "compact" strategy from the previous example, the threads will be packed and an NT which is less than MAXNT would not work out with using below idea. In contrast, the MPI process placement uses a stride ("SHIFT") which makes sure to hit the correct core. It is recommended to even treat a "POPT" build similar to a "PSMP" just in case that any linked library (which uses internal OpenMP threads) will be optimally affinitized.

The CP2K/intel branch carries a number of "reconfigurations" and environment variables that 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_PREFETCH**: environment variable for enabling (default), or disabling (CP2K_PREFETCH=0), and selecting the prefetch strategy (see list at the end of the Generator Driver section). This is only in effect if CP2K_RECONFIGURE is available and enabled.
- **CP2K_DENSE**: environment variable for enabling (default), or disabling (CP2K_DENSE=0) the "densification" of matrix blocks (when the matrix stacks are build). Please note that "enabling" densification depends on other conditions such that densification might remain off. Normally, larger blocks can be picked more likely by a regular BLAS implementation (which is usually tuned towards larger matrices), however the densification itself might raise some cost.
- **CP2K_RMA**: environment variable for enabling (1), or disabling (0) the RMA-style (Remote Memory Access) MPI parallelization in DBCSR. The CP2K/intel branch uses MPI=3 implicitly when building the executable. The MPI standard version 3 is in fact necessary in order to perform Remote Memory Access (RMA) as implemented in CP2K. The effect of the environment variable is similar to the keyword from the input reference.
- **CP2K_FILTERING**: environment variable for enabling (1), or disabling (0) the filtering out matrix blocks from parallel communication when the Cannon matrix multiplication algorithm is performed. The effect of the environment variable is similar to the keyword from the input reference.
- **MM_DRIVER**: http://manual.cp2k.org/trunk/CP2K_INPUT/GLOBAL/DBCSR.html#MM_DRIVER gives a reference of the input keywords. For the CP2K/intel branch the MM_DRIVER is set to XSMM by default (if LIBXSMMROOT was present).

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=$(HOME)/libint` needs to be supplied when building the CP2K application (make).

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

To configure, build, and install LIBXC (Version 2.2.2 has been tested), one may proceed as shown below. To actually make use of LIBINT, the key `LIBXCROOT=$(HOME)/libxc` needs to be supplied when building the CP2K application (make).

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

In case 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

Intel Xeon Phi Coprocessor

For those having an Intel Xeon Phi coprocessor in reach, an experimental code path using CP2K's ACCeleration layer (which was originally built for attached accelerators) is able to offload computation from the host system. However, the implementation leaves the host processor(s) unutilized (beside from offloading and transferring the work). Please note that although the host is only MPI-parallelized, the coprocessor uses OpenMP within each partition formed by a host-rank. For more details about affinitizing the execution on the coprocessor, one may have a look at https://github.com/hfp/mpirun.

```
make ARCH=Linux-x86-64-intel VERSION=popt ACC=1 -j
mpirun.sh -p8 -x exe/Linux-x86-64-intel/cp2k.popt workload.inp
```

For more details about offloading CP2K's DBCSR matrix multiplications to an Intel Xeon Phi Coprocessor, please have a look at https://github.com/hfp/libxstream/raw/master/documentation/cp2k.pdf. Further, cross-building CP2K for the Intel Xeon Phi coprocessor in order to run in a self-hosted fashion is currently out of scope for this document. However, running through CP2K's ACCeleration layer while executing on a host system is another possibility enabled by the universal implementation. However, the code path omitting the ACCeleration layer (see Running the Application) is showing better performance (although the code which is actually performing the work is the same).

```
make ARCH=Linux-x86-64-intel VERSION=psmp ACC=1 OFFLOAD=0 -j
```

Eigenvalue SoLvers for Petaflop-Applications (ELPA)

1. Download the latest ELPA from http://elpa.rzg.mpg.de/elpa-tar-archive (2015.05.001)
2. Make use of the ELPAROOT key-value pair (`ELPA=2` is used by default to rely on ELPA2).

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>`).