CP2K Open Source Molecular Dynamics

This document is intended to be a recipe for building and running CP2K's "intel" branch 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).

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 --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 the recommended version of the Intel Compiler (see below command). 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
cd cp2k/makefiles
make ARCH=Linux-x86-64-intel VERSION=psmp -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`).

- **LIBXSMMROOT**: set `LIBXSMMROOT=` (no path) to disable LIBXSMM and thereby the XSMM driver.
- **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 for performance profiling.
- **DBG**: set `DBG=1` to include debug symbols, and to generate non-optimized code.

Please note that the arch-files for the versions "popt", "sopt", and "ssmp" are provided for convenience and are actually based on the "psmp"-configuration by using even more of the above key-value pairs (`OMP`, `ACC`, etc.).

Running the Application

Running the application may go beyond a single node, however for the purpose of an example the command line shown below is limited to a single node. 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`. The CP2K/intel branch aims to enable a performance advantage by default. However, there are some options allowing to re-enable default behavior (compared to CP2K/trunk).

- **LIBXSMM_ACC_RECONFIGURE=0**: this environment setting avoids reconfiguring CP2K (only enabled when the ACCeleration layer is active; see below). With the ACCeleration layer enabled, a number of properties reconfigured e.g. an increased number of entries per matrix stack.
- **MM_DRIVER**: http://manual.cp2k.org/trunk/CP2K_INPUT/GLOBAL/DBCSR.html#MM_DRIVER gives a reference of the input keywords. Beside of the listed keywords (ACC, BLAS, MATMUL, and SMM), the CP2K/intel branch is supporting the XSMM keyword (which is also the default).

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>`).