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. Usually any version of Intel MPI if fine when not relying on particular features (see Tuning).

```
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
```

Running the Application

Running the application may go beyond a single node, however below command line runs on a single node for the matter of an example. For systems with higher core counts (and a sufficient amount of memory), it is recommended to start trying a number of processes which is half the number of cores. For example running on a dual-socket system with 16 cores per processor (64 hardware threads in cases where HT is enabled) may look like:

```
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.

Tuning

The CP2K/intel branch aims to enable a clear performance advantage by default. However, sometimes there are more options available or a non-default code path (compared to CP2K/trunk) which can be disabled.

- **LIBXSMM_ACC_RECONFIGURE=0**: this environment variable setting avoids reconfiguring CP2K. There are a number of properties reconfigured e.g., the number of entries per matrix stack is raised.
- **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 XSMM keyword is supported as well (also the default in CP2K/intel).

Further, key-value pairs (`make ARCH=Linux-x86-64-intel VERSION=psmp KEY1=VALUE1 ...`) allow adjusting CP2K at build time of the application.

- **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 unoptimized code.

There is one more experimental code path (`make ARCH=Linux-x86-64-intel VERSION=psmp ACC=1 OFFLOAD=0 -j`) able to run on a general host system. It is going through CP2K's ACCeleration layer which was originally built for attached accelerators. Usually the "host" code path is showing better performance, anyhow the actual code which is performing the work is the same in both cases.