CP2K Open Source Molecular Dynamics

This document focuses on offloading CP2K's DBCSR small matrix multiplication to the Intel Xeon Phi Coprocessor, and this document is accompanying the recipe for building and running CP2K's "intel" branch as described in:

> https://github.com/hfp/libxsmm/raw/master/documentation/cp2k.pdf.

Getting and Building the Source Code

Please read the above mentioned document entirely, and use one of the recommended compiler versions:

- Intel Compiler 15.0.3.187 (Build 20150407)
- Intel Compiler 16.0.0.109 (Build 20150815)

For Intel MPI, usually any version is fine.

```
git clone --branch intel https://github.com/cp2k/cp2k.git cp2k.git
ln -s cp2k.git/cp2k cp2k
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=popt ACC=1 -j
```

It is recommended to rely on a non-SMP build ("popt", "sopt") where "popt" makes the most sense in order to partition the coprocessor according to the number of ranks on the host system. Please note that although the host may be only MPI-parallelized, the coprocessor uses OpenMP within each partition formed by a host-rank.

Running the Application

To improve scalability, the coprocessor can partitioned by the MPI-ranks launched on the host system with each rank offloading work independently. This is solely achieved by setting a number of environment variables helping to place and pin the threads on the coprocessor. In order to ease this step, one may employ scripts as found at:

> https://github.com/hfp/mpirun.

Although the script can take a list of nodes, it may not be suitable for launching on a larger set of cluster nodes due to building an excessively large command line (rather than relying on the "host file").

In below example, the argument "-p8" could be very suitable for single socket of a 16-core system with a single 3-series coprocessor (57 cores) attached to the first socket. The number of cores (total number minus one core for the uOS) may be divisible by the number of ranks per host-socket. It is usually preferable to minimize the number of remaining cores on the coprocessor since the current implementation in the CP2K/intel branch is leaving the host processor(s) unutilized (beside from offloading and transferring the work).

```
wget https://raw.githubusercontent.com/hfp/mpirun/master/mpirun.sh
wget https://raw.githubusercontent.com/hfp/mpirun/master/mpirun.py
chmod +x mpirun.*
mpirun.sh -p8 -x exe/Linux-x86-64-intel/cp2k.popt workload.inp
```

For an actual workload, one may try the following:

```
&GLOBAL
  PRINT_LEVEL MEDIUM
  PROGRAM_NAME TEST
  RUN_TYPE NONE
  &TIMINGS
     THRESHOLD 0.00001
  &END
&END GLOBAL
&TEST
  &CP_DBCSR
     K  6440
     M  6440
     N  6440
     TRANSA TRUE
     TRANSB FALSE
     N_LOOP 4
     ASPARSITY 0.0001
     BSPARSITY 0.0001
     CSPARSITY 0.0001
```

```
    bs_m 1 23
    bs_n 1 23
    bs_k 1 23
    KEEPSPARSE .FALSE.
  &END
&END TEST
```