CP2K MPI/OpenMP-hybrid Execution (PSMP)

Overview

CP2K's grid-based calculation as well as DBCSR's block sparse matrix multiplication (Cannon algorithm) prefer a square-number for the total rank-count (2d communication pattern). This is not to be obfuscated with a Power-of-Two (POT) rank-count that usually leads to trivial work distribution (MPI).

It can be more efficient to leave CPU-cores unused to achieve this square-number property than using all cores with a "wrong" total rank-count (sometimes a frequency upside over an "all-core turbo" emphasizes this property further). Counter-intuitively, even an unbalanced rank-count per node i.e., different rank-counts per socket can be an advantage. Pinning MPI processes and placing threads requires extra care to be taken on a per-node basis to load a dual-socket system in a balanced fashion or to setup space between ranks for the OpenMP threads.

Because of the above-mentioned complexity, a script for planning MPI/OpenMP-hybrid execution (`plan.sh`) is available. Here is a first example for running the PSMP-binary on an SMP-enabled (Hyperthreads) dual-socket system with 24 cores per processor/socket (96 hardware threads in total). At first, a run with 48 ranks and 2 threads per core comes to the mind (`48x2`). However, for instance 16 ranks with 6 threads per rank may be better for performance (`16x6`). To easily place the ranks, Intel MPI is used:

```
mpirun -np 16 \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv OMP_PLACES=threads -genv OMP_PROC_BIND=SPREAD \
  -genv OMP_NUM_THREADS=6 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

NOTE: For hybrid codes, `I_MPI_PIN_DOMAIN=auto` is recommended as it spaces the ranks according to the number of OpenMP threads (`OMP_NUM_THREADS`). It is not necessary and not recommended to build a rather complicated `I_MPI_PIN_PROCESSOR_LIST` for hybrid codes (MPI plus OpenMP). To display and to log the pinning and thread affinization at the startup of an application, `I_MPI_DEBUG=4` can be used with no performance penalty. The recommended `I_MPI_PIN_ORDER=bunch` ensures that ranks per node are split as even as possible with respect to sockets (e.g., 36 ranks on a 2x20-core system are put in 2x18 ranks instead of 20+16 ranks).

Plan Script

To configure the plan-script, the metric of the compute nodes can be given for future invocations

so that only the node-count is required as an argument. The script's help output (`-h` or `--help`) initially shows the "system metric" of the computer the script is invoked on. For a system with 48 cores (two sockets, SMP/HT enabled), setting up the "system metric" looks like (`plan.sh <num-nodes> <ncores-per-node> <nthreads-per-core> <nsockets-per-node>`):

```
./plan.sh 1 48 2 2
```

The script is storing the arguments (except for the node-count) as default values for the next plan (file: `$HOME/.xconfigure-cp2k-plan`). This allows to supply the system-type once, and to plan with varying node-counts in a convenient fashion. Planning for 8 nodes of the above kind yields the following output (`plan.sh 8`):

```
================================================================================
384 cores: 8 node(s) with 2x24 core(s) per node and 2 thread(s) per core
================================================================================
[48x2]: 48 ranks per node with 2 thread(s) per rank (14% penalty)
[24x4]: 24 ranks per node with 4 thread(s) per rank (14% penalty)
[12x8]: 12 ranks per node with 8 thread(s) per rank (33% penalty)
--------------------------------------------------------------------------------
[32x3]: 32 ranks per node with 3 thread(s) per rank (34% penalty) -> 16x16
[18x5]: 18 ranks per node with 5 thread(s) per rank (25% penalty) -> 12x12
[8x12]: 8 ranks per node with 12 thread(s) per rank (0% penalty) -> 8x8
[2x48]: 2 ranks per node with 48 thread(s) per rank (0% penalty) -> 4x4
--------------------------------------------------------------------------------
```

The first group of the output displays POT-style (trivial) MPI/OpenMP configurations (penalty denotes potential communication overhead), however the second group (if present) shows rank/thread combinations with the total rank-count hitting a square number (penalty denotes waste of compute due to not filling each node). For the given example, 8 ranks per node with 12 threads per rank may be chosen (`8x12`) and MPI-executed:

```
mpirun -perhost 8 -host node1,node2,node3,node4,node5,node6,node7,node8 \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv OMP_PLACES=threads -genv OMP_PROC_BIND=SPREAD \
  -genv OMP_NUM_THREADS=12 -genv I_MPI_DEBUG=4 \
  exe/Linux-x86-64-intelx/cp2k.psmp workload.inp
```

The script also suggests close-by configurations (lower and higher node-counts) that hit the square-property ("Try also the following node counts"). The example (as exercised above) was to illustrate how the script works, however it can be very helpful when running jobs especially on CPUs with not many prime factors in the core-count. Remember, the latter can be also the case for virtualized environments that reserve some of the cores to run the Hypervisor i.e., reporting less cores to the Operating System (guest OS) when compared to the physical core-count.

References