XCONFIGURE

XCONFIGURE is a collection of configure wrapper scripts for various HPC applications. The purpose of the scripts is to configure the application in question to make use of Intel's software development tools (Intel Compiler, Intel MPI, Intel MKL). XCONFIGURE helps to rely on a "build recipe", which is known to expose the highest performance or to reliably complete the build process.

Contributions are very welcome!

Each application (or library) is hosted in a separate directory. To configure (and ultimately build) an application, one can rely on a single script which then downloads a specific wrapper into the current working directory (of the desired application).

```
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh qe hsw
```

On systems without access to the Internet, one can download (or clone) the entire collection upfront. To configure an application, please open the "config" folder and follow the build recipe of the desired application.

Documentation

- **ReadtheDocs**: online documentation with full text search.
- **PDF**: a single documentation file.

Related Projects

- Spack Package Manager: http://computation.llnl.gov/projects/spack-hpc-package-manager
- EasyBuild / EasyConfig (University of Gent): https://github.com/easybuilders

Please note that XCONFIGURE has a narrower scope when compared to the above package managers.

Applications

CP2K

This document focuses on building and running the Intel branch of CP2K. However, it applies to CP2K in general (unless emphasized). The Intel branch is hosted at GitHub and is supposed to represent the master version of CP2K in a timely fashion. CP2K's main repository is hosted at SourceForge but it is automatically mirrored at GitHub. The LIBXSMM library can be found at https://github.com/hfp/libxsmm. In terms of functionality (and performance) it is beneficial to rely on LIBINT and LIBXC, whereas ELPA eventually improves the performance. For high performance, it is strongly recommended to use LIBXSMM which has been incorporated since CP2K 3.0. LIBXSMM is intended to substitute CP2K's "libsmm" library.

There are below Intel compiler releases (one can combine components from different versions), which are known to reproduce correct results (regression tests):

- Intel Compiler 2017 (**any**), and the **initial** release of MKL 2017 ("update 0")
  - source /opt/intel/compilers_and_libraries_2017.[*whatever*]/linux/bin/compilervars.sh intel64
  - source /opt/intel/compilers_and_libraries_2017.0.098/linux/mkl/bin/mklvars.sh intel64
- Intel Compiler 2017 Update 4, and any later update of the 2017 suite
  - source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
  - source /opt/intel/compilers_and_libraries_2017.5.239/linux/bin/compilervars.sh intel64
- Intel Compiler 2018 suite is not validated (and fails at runtime)
- Intel MPI; usually any version is fine

There are no configuration wrapper scripts provided for CP2K, please follow below recipe. However, attempting to run below command yields an info-script:

```
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh cp2k
```

Of course, the above can be simplified:

```
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/config/cp2k/info.sh
chmod +x info.sh
```

Build Instructions

Build the CP2K/Intel Branch

To build CP2K/Intel from source, one may rely on Intel Compiler 16 or 17 series (the 2018 version may be supported at a later point in time). For the Intel Compiler 2017 prior to Update 4, one should source the compiler followed by sourcing a specific version of Intel MKL (to avoid an issue in Intel MKL):

```
source /opt/intel/compilers_and_libraries_2017.3.191/linux/bin/compilervars.sh intel64
source /opt/intel/compilers_and_libraries_2017.0.098/linux/mkl/bin/mklvars.sh intel64
```

Since Update 4 of the 2017-suite, the compiler and libraries can be used right away (see recommended compiler):

```
source /opt/intel/compilers_and_libraries_2017.6.256/linux/bin/compilervars.sh intel64
```

LIBXSMM is automatically built in an out-of-tree fashion when building CP2K/Intel branch. The only prerequisite is that the LIBXSMMROOT path needs to be detected (or supplied on the `make` command line). A recipe targeting "Haswell" (HSW) may look like:

```
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
cd cp2k/makefiles
make ARCH=Linux-x86-64-intelx VERSION=psmp AVX=2
```

To target "Knights Landing" (KNL), use "AVX=3 MIC=1" instead of "AVX=2". Similarly, the Intel Xeon Scalable processor "Skylake Server" (SKX) goes with "AVX=3 MIC=0".

To further adjust CP2K at build time of the application, additional key-value pairs can be passed at make's command line (like `ARCH=Linux-x86-64-intelx` or `VERSION=psmp`).

- **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.

To further improve performance and versatility, one may supply LIBINTROOT, LIBXCROOT, and ELPAROOT when relying on CP2K/Intel's ARCH files (see later sections about these libraries).

Build an Official Release

Since CP2K 3.0, the mainline version (non-Intel branch) also supports LIBXSMM. CP2K 6.1 includes `Linux-x86-64-intelx.*` (`arch` directory) as a starting point for an own ARCH-file. Please follow the official guide and consider the CP2K Forum in case of trouble. If an own ARCH file is used or prepared, the LIBXSMM library needs to be built separately. Building LIBXSMM is rather simple, to build the master revision:

```
git clone https://github.com/hfp/libxsmm.git
cd libxsmm ; make
```

To build an official release:

```
wget https://github.com/hfp/libxsmm/archive/1.9.tar.gz
tar xvf 1.9.tar.gz
cd libxsmm-1.9 ; make
```

To download and build an official CP2K release, one can still use the ARCH files that are part of the CP2K/Intel branch. In this case, LIBXSMM is also built implicitly.

```
git clone https://github.com/hfp/libxsmm.git
wget https://sourceforge.net/projects/cp2k/files/cp2k-5.1.tar.bz2
tar xvf cp2k-5.1.tar.bz2
cd cp2k-5.1/arch
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intelx.arch
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intelx.popt
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intelx.psmp
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intelx.sopt
wget https://github.com/cp2k/cp2k/raw/intel/cp2k/arch/Linux-x86-64-intelx.ssmp
cd ../makefiles
source /opt/intel/compilers_and_libraries_2017.6.256/linux/bin/compilervars.sh intel64
make ARCH=Linux-x86-64-intelx VERSION=psmp AVX=2
```

To further improve performance and versatility, one may supply LIBINTROOT, LIBXCROOT, and ELPAROOT when relying on CP2K/Intel's ARCH files (see the following sections about these libraries).

LIBINT, LIBXC, and ELPA

To configure, build, and install LIBINT (version 1.1.5 and 1.1.6 have been tested), one can proceed with https://xconfigure.readthedocs.io/libint/README/. Also note there is no straightforward way to cross-compile LIBINT 1.1.x for an instruction set extension that is not supported by the compiler host. To incorporate LIBINT into CP2K, the key `LIBINTROOT=/path/to/libint` needs to be supplied when using CP2K/Intel's ARCH files (make).

To configure, build, and install LIBXC (version 3.0.0 has been tested), and one can proceed with https://xconfigure.readthedo
To incorporate LIBXC into CP2K, the key `LIBXCROOT=/path/to/libxc` needs to be supplied when using CP2K/Intel's ARCH files (make).

To configure, build, and install the Eigenvalue SoLvers for Petaflop-Applications (ELPA), one can proceed with https://xconfigure.readthedocs.io/libint/README/. To incorporate ELPA into CP2K, the key `ELPAROOT=/path/to/elpa` needs to be supplied when using CP2K/Intel's ARCH files (make). The Intel-branch defaults to ELPA-2017.05 (earlier versions can rely on the ELPA key-value pair e.g., `ELPA=201611`).

```
make ARCH=Linux-x86-64-intelx VERSION=psmp ELPAROOT=/path/to/elpa/default-arch
```

At runtime, a build of the Intel-branch supports an environment variable CP2K_ELPA:

- **CP2K_ELPA=-1**: requests ELPA to be enabled; the actual kernel type depends on the ELPA configuration.
- **CP2K_ELPA=0**: ELPA is not enabled by default (only on request via input file); same as non-Intel branch.
- **CP2K_ELPA**=<not-defined>: requests ELPA-kernel according to CPUID (default with CP2K/Intel-branch).

Memory Allocation

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 such a strategy, memory allocation wrappers can be used to replace the default memory allocation at build-time or at runtime of an application.

To use the malloc-proxy of the Intel Threading Building Blocks (Intel TBB), rely on the `TBBMALLOC=1` key-value pair at build-time of CP2K. Usually, Intel TBB is already available when sourcing the Intel development tools (one can check the TBBROOT environment variable). To use TCMALLOC as an alternative, set `TCMALLOCROOT` at build-time of CP2K by pointing to TCMALLOC's installation path (configured per `./configure --enable-minimal --prefix=<TCMALLOCROOT>`).

Run Instructions

Running the application may go beyond a single node, however for first example the pinning scheme and thread affinization is introduced. As a rule of thumb, a high rank-count for single-node computation (perhaps according to the number of physical CPU cores) may be preferred. In contrast (communication bound), a lower rank count for multi-node computations may be desired. In general, CP2K prefers the total rank-count to be a square-number (two-dimensional communication pattern) rather than a Power-of-Two (POT) number.

Running an MPI/OpenMP-hybrid application, an MPI rank-count that 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 I_MPI_PIN_ORDER=bunch \
  -genv KMP_AFFINITY=compact,granularity=fine,1 \
  -genv OMP_NUM_THREADS=4 \
  cp2k/exe/Linux-x86-64-intelx/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 CP2K/Intel branch carries several "reconfigurations" and environment variables, which allow to adjust important runtime options. Most of these options are also accessible via the input file format (input reference e.g., https://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_HUGEPAGES**: environment variable for disabling (0) huge page based memory allocation, which is enabled by default (if TBBROOT was present at build-time of the application).
- **CP2K_RMA**: enables (1) an experimental Remote Memory Access (RMA) based multiplication algorithm (requires MPI3).
- **CP2K_SORT**: enables (1) an indirect sorting of each multiplication stack according to the C-index (experimental).

Sanity Check

There is nothing that can replace the full regression test suite. However, to quickly check whether a build is sane or not, one can run for instance `tests/QS/benchmark/H2O-64.inp` and check if the SCF iteration prints like the following:

```
 Step     Update method      Time    Convergence         Total energy    Change

    1 OT DIIS     0.15E+00    0.5     0.01337191     -1059.6804814927 -1.06E+03
    2 OT DIIS     0.15E+00    0.3     0.00866338     -1073.3635678409 -1.37E+01
    3 OT DIIS     0.15E+00    0.3     0.00615351     -1082.2282197787 -8.86E+00
    4 OT DIIS     0.15E+00    0.3     0.00431587     -1088.6720379505 -6.44E+00
    5 OT DIIS     0.15E+00    0.3     0.00329037     -1092.3459788564 -3.67E+00
    6 OT DIIS     0.15E+00    0.3     0.00250764     -1095.1407783214 -2.79E+00
    7 OT DIIS     0.15E+00    0.3     0.00187043     -1097.2047924571 -2.06E+00
    8 OT DIIS     0.15E+00    0.3     0.00144439     -1098.4309205383 -1.23E+00
    9 OT DIIS     0.15E+00    0.3     0.00112474     -1099.2105625375 -7.80E-01
   10 OT DIIS     0.15E+00    0.3     0.00101434     -1099.5709299131 -3.60E-01
   [...]
```

The column called "Convergence" must monotonically converge towards zero.

Performance

An info-script (`info.sh`) is available attempting to present a table (summary of all results), which is generated from log files (use `tee`, or rely on the output of the job scheduler). There are only certain file extensions supported (`.txt`, `.log`). If no file matches, then all files (independent of the file extension) are attempted to be parsed (which will go wrong eventually). For legacy reasons (run command is not part of the log, etc.), certain schemes for the filename are eventually parsed and translated as well.

```
./run-cp2k.sh | tee cp2k-h2o64-2x32x2.txt
ls -1 *.txt
cp2k-h2o64-2x32x2.txt
cp2k-h2o64-4x16x2.txt

./info.sh [-best] /path/to/logs-or-cwd
H2O-64             Nodes R/N T/R Cases/d Seconds
cp2k-h2o64-2x32x2 2       32   4     807 107.237
cp2k-h2o64-4x16x2 4       16   8     872  99.962
```

Please note that the number of cases per day (Cases/d) are currently calculated with integer arithmetic and eventually lower than just rounding down (based on 86400 seconds per day). The number of seconds taken are end-to-end (wall time), i.e. total time to solution including any (sequential) phase (initialization, etc.). Performance is higher if the workload requires more iterations (some publications present a metric based on iteration time).

ELPA

Build Instructions

ELPA 2017.11.001

Download and unpack ELPA, and make the configure wrapper scripts available in ELPA's root folder. It is recommended to package the state (Tarball or similar), which is achieved after downloading the wrapper scripts.

NOTE: this version is not suitable for Quantum Espresso (QE) since it removed some bits from the ELPA1 legacy interface (get_elpa_row_col_comms, etc.). At the moment, ELPA 2017.05.003 is the latest supported version for QE!

```
wget http://elpa.mpcdf.mpg.de/html/Releases/2017.11.001/elpa-2017.11.001.tar.gz
tar xvf elpa-2017.11.001.tar.gz
cd elpa-2017.11.001
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh elpa
```

Please make the Intel Compiler and Intel MKL available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2017.5.239/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make clean
./configure-elpa-skx-omp.sh
make -j ; make install

make clean
./configure-elpa-skx.sh
make -j ; make install
```

After building and installing the desired configuration(s), one may have a look at the installation:

```
[user@system elpa-2017.11.001]$ ls ../elpa
 default-skx
 default-skx-omp
```

For different targets (instruction set extensions) or for different versions of the Intel Compiler, the configure scripts support an additional argument ("default" is the default tagname):

```
./configure-elpa-hsw-omp.sh tagname
```

As shown above, an arbitrary "tagname" can be given (without editing the script). This might be used to build multiple variants of the ELPA library.

ELPA 2017.05.003

Download and unpack ELPA, and make the configure wrapper scripts available in ELPA's root folder. It is recommended to package the state (Tarball or similar), which is achieved after downloading the wrapper scripts.

```
wget http://elpa.mpcdf.mpg.de/html/Releases/2017.05.003/elpa-2017.05.003.tar.gz
tar xvf elpa-2017.05.003.tar.gz
cd elpa-2017.05.003
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh elpa
```

Please make the Intel Compiler and Intel MKL available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon E5v4 processor (formerly codenamed "Broadwell"):

```
make clean
./configure-elpa-hsw-omp.sh
make -j ; make install

make clean
./configure-elpa-hsw.sh
make -j ; make install
```

ELPA 2016.11.001

Download and unpack ELPA, and make the configure wrapper scripts available in ELPA's root folder. It is recommended to package the state (Tarball or similar), which is achieved after downloading the wrapper scripts. It appears that ELPA's `make clean` (or similar Makefile target) is cleaning up the entire directory including all "non-ELPA content" (the directory remains unclean such that subsequent builds may fail).

```
wget http://elpa.mpcdf.mpg.de/html/Releases/2016.11.001.pre/elpa-2016.11.001.pre.tar.gz
tar xvf elpa-2016.11.001.pre.tar.gz
cd elpa-2016.11.001.pre
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh elpa
```

Please make the Intel Compiler and Intel MKL available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon E5v4 processor (formerly codenamed "Broadwell"):

```
./configure-elpa-hsw-omp.sh
make -j ; make install
```

ELPA Development

To rely on experimental functionality, one may git-clone the master branch of the ELPA repository instead of downloading a regular version.

```
git clone --branch ELPA_KNL https://gitlab.mpcdf.mpg.de/elpa/elpa.git
```

References

https://software.intel.com/en-us/articles/quantum-espresso-for-the-intel-xeon-phi-processor

LIBINT

Version 1.x

For CP2K, LIBINT 1.x is required. Download and unpack LIBINT, and make the configure wrapper scripts available in LIBINT's root folder. Please note that the "automake" package is a prerequisite.

```
wget --no-check-certificate https://github.com/evaleev/libint/archive/release-1-1-6.tar.gz
tar xvf release-1-1-6.tar.gz
cd libint-release-1-1-6
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh libint
```

Please make the Intel Compiler available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon E5v4 processor (formerly codenamed "Broadwell"):

```
make distclean
./configure-libint-hsw.sh
make -j; make install
```

The version 1.x line of LIBINT does not support to cross-compile for an architecture (a future version of the wrapper scripts may patch this ability into LIBINT 1.x). Therefore, one can rely on the Intel Software Development Emulator (Intel SDE) to compile LIBINT for targets, which cannot execute on the compile-host.

```
/software/intel/sde/sde -knl -- make
```

To speed-up compilation, "make" might be carried out in phases: after "printing the code" (c-files), the make execution continues with building the object-file where no SDE needed. The latter phase can be sped up by interrupting "make", and executing it without SDE. The root cause of the entire problem is that the driver printing the c-code is (needlessly) compiled using the architecture-flags that are not supported on the host.

Further, for different targets (instruction set extensions) or different versions of the Intel Compiler, the configure scripts support an additional argument ("default" is the default tagname):

```
./configure-libint-hsw.sh tagname
```

As shown above, an arbitrary "tagname" can be given (without editing the script). This might be used to build multiple variants of the LIBINT library.

LIBXC

To configure, build, and install LIBXC 2.x, 3.x, and 4.x, one may proceed as shown below. Please note that CP2K 5.1 (and earlier) is only compatible with LIBXC 3.0 (or earlier, see also How to compile the CP2K code). Post-5.1, only the latest major release of LIBXC (by the time of the CP2K-release) will be supported (e.g., LIBXC 4.x).

```
wget --content-disposition http://www.tddft.org/programs/octopus/down.php?file=libxc/4.0.4/libxc-4.0.4
tar xvf libxc-4.0.3.tar.gz
cd libxc-4.0.3
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh libxc
```

Please make the Intel Compiler available on the command line. This depends on the environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2017.6.256/linux/bin/compilervars.sh intel64
```

For example, to configure and make for an Intel Xeon Scalable processor ("SKX"):

```
make distclean
./configure-libxc-skx.sh
make -j; make install
```

LIBXSMM

LIBXSMM is a library targeting Intel Architecture (x86) for small, dense or sparse matrix multiplications, and small convolutions. The build instructions can be found at https://github.com/hfp/libxsmm (PDF).

QE

Build Instructions

Download and unpack Quantum Espresso, and make the configure wrapper scripts available in QE's root folder. Please note that the configure wrapper scripts support QE 6.x (prior support for 5.x is dropped). Before building QE, one needs to complete the recipe for ELPA.

NOTE: the ELPA configuration must correspond to the desired QE configuration e.g., `configure-elpa-skx-omp.sh` and `configure-qe-skx-omp.sh` ("omp"). The version ELPA 2017.11.001 (and later) removed some bits from the ELPA1 legacy interface needed by QE (get_elpa_row_col_comms, etc.), hence ELPA 2017.05.003 is the latest supported version!

```
http://www.qe-forge.org/gf/download/frsrelease/247/1132/qe-6.2.1.tar.gz
tar xvf qe-6.2.1.tar.gz
cd qe-6.2.1
wget --no-check-certificate https://github.com/hfp/xconfigure/raw/master/configure-get.sh
chmod +x configure-get.sh
./configure-get.sh qe
```

Please make the Intel Compiler available on the command line, which may vary with the computing environment. For instance, many HPC centers rely on `module load`.

```
source /opt/intel/compilers_and_libraries_2017.4.196/linux/bin/compilervars.sh intel64
```

For example, configure for an Intel Xeon E5v4 processor (formerly codenamed "Broadwell"), and build the desired application(s) e.g., "pw", "cp", or "all".

```
./configure-qe-hsw-omp.sh
make pw -j
```

Building "all" (or `make` without target argument) requires to repeat `make all` until no compilation error occurs. This is because of some incorrect build dependencies (build order issue which might have been introduced by the configure wrapper scripts). In case of starting over, one can run `make distclean`, reconfigure the application, and build it again. For different targets (instruction set extensions) or different versions of the Intel Compiler, the configure scripts support an additional argument ("default" is the default tagname):

```
./configure-qe-hsw-omp.sh tagname
```

As shown above, an arbitrary "tagname" can be given (without editing the script). This might be used to build multiple variants of QE. Please note: this tagname also selects the corresponding ELPA library (or should match the tagname used to build ELPA). Make sure to save your current QE build before building an additional variant!

Run Instructions

To run Quantum Espresso in an optimal fashion depends on the workload and on the "parallelization levels", which can be exploited by the workload in question. These parallelization levels apply to execution phases (or major algorithms) rather than staying in a hierarchical relationship (levels). It is recommended to read some of the primary references explaining these parallelization levels (a number of them can be found in the Internet including some presentation slides). Time to solution may *vary by factors* depending on whether these levels are orchestrated or not. To specify these levels, one uses command line arguments along with the QE executable(s):

- `-npool`: try to maximize the number of pools. The number depends on the workload e.g., if the number of k-points can be distributed among independent pools. Indeed, trial and error is a rather quick to check if workload fails to pass the initialization phase. One may use prime numbers: *2*, *3*, *5*, etc. (default is *1*). For example, when *npool=2* worked it might be worth trying *npool=4*. On the other hand, increasing the number pools duplicates the memory consumption accordingly (larger numbers are increasingly unlikely to work).
- `-ndiag`: this number determines the number of ranks per pool used for dense linear algebra operations (DGEMM and ZGEMM). For example, if *64* ranks are used in total per node and *npool=2*, then put *ndiag=32* (QE selects the next square number which is less-equal than the given number e.g., *ndiag=25* in the previous example).
- `-ntg`: specifies the number of tasks groups per pool being used for e.g., FFTs. One can start with `NTG=$((NUMNODES*NRANKS/(NPOOL*2)))`. If `NTG` becomes zero, `NTG=${NRANKS}` should be used (number of ranks per node). Please note the given formula is only a rule of thumb, and the number of task groups also depends on the number of ranks as the workload is scaled out.

To run QE, below command line can be a starting point ("numbers" are presented as Shell variables to better understand the inner mechanics). Important for hybrid builds (MPI and OpenMP together) are the given environment variables. The `KMP_AFFINITY` assumes Hyperthreading (SMT) is enabled (granularity=fine), and the "scatter" policy allows to easily run less than the maximum number of Hyperthreads per core. As a rule of thumb, OpenMP adds only little overhead (often not worth a pure MPI application) but allows to scale further out when compared to pure MPI builds.

```
mpirun -bootstrap ssh -genvall \
  -np $((NRANKS_PER_NODE*NUMNODES)) -perhost ${NRANKS} \
  -genv I_MPI_PIN_DOMAIN=auto -genv I_MPI_PIN_ORDER=bunch \
  -genv KMP_AFFINITY=compact,granularity=fine,1 \
  -genv OMP_NUM_THREADS=${NTHREADS_PER_RANK} \
  /path/to/pw.x \<command-line-arguments\>
```

Performance

An info-script (`info.sh`) is available attempting to present a table (summary of all results), which is generated from log files (use `tee`, or rely on the output of the job scheduler). There are only certain file extensions supported (`.txt`, `.log`). If no file matches, then all files (independent of the file extension) are attempted to be parsed (which will go wrong eventually). For legacy reasons (run command is not part of the log, etc.), certain schemes for the filename are eventually parsed and translated as well.

```
./run-qe.sh | tee qe-asrf112-4x16x1.txt
ls -1 *.txt
qe-asrf112-2x32x1.txt
qe-asrf112-4x16x1.txt

./info.sh [-best] /path/to/logs-or-cwd
AUSURF112          Nodes R/N T/R Cases/d Seconds NPOOL NDIAG NTG
qe-asrf112-2x32x1 2       32   2     533  162.35     2    25  32
qe-asrf112-4x16x1 4       16   4     714  121.82     2    25  32
```

Please note that the number of cases per day (Cases/d) are currently calculated with integer arithmetic and eventually lower than just rounding down (based on 86400 seconds per day). The number of seconds taken are end-to-end (wall time), i.e. total time to solution including any (sequential) phase (initialization, etc.). Performance is higher if the workload requires more iterations (some publications present a metric based on iteration time).

References

https://software.intel.com/en-us/articles/quantum-espresso-for-the-intel-xeon-phi-processor
http://www.quantum-espresso.org/wp-content/uploads/Doc/user_guide/node18.html