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

Build and Run Instructions

The build and run instructions for CP2K using Intel Software Development Tools are exercised at http://libxsmm.readthedocs.io/cp2k/ (pdf).

Please note, in terms of functionality it is beneficial to rely on LIBINT and LIBXC, whereas ELPA eventually improves the performance. For high performance, it is strongly recommended to make use of LIBXSMM.

Sanity Check

There is nothing that can replace the full regression test suite to be clear. 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" has to monotonically converge towards zero.

References

http://libxsmm.readthedocs.io/cp2k/

ELPA

Build Instructions

ELPA 2017.05.002

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 https://elpa.mpcdf.mpg.de/html/Releases/2017.05.002/elpa-2017.05.002.tar.gz
tar xvf elpa-2017.05.002.tar.gz
cd elpa-2017.05.002
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
```

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

```
[user@system elpa-2017.05.002]$ ls ../elpa
 default-hsw
 default-hsw-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 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
http://libxsmm.readthedocs.io/cp2k/

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 5.0 (and 4.x) are only compatible with LIBXC 3.0 (or earlier, see also How to compile the CP2K code). The CP2K development version (after 5.x and later) as well as the Intel branch (shortly prior to 5.x and later) support LIBXC 4.x.

```
wget --content-disposition http://www.tddft.org/programs/octopus/down.php?file=libxc/4.0.1/libxc-4.0.1
tar xvf libxc-4.0.1.tar.gz
cd libxc-4.0.1
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_2017.4.196/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 6.0 and QE 6.1 (prior support for 5.x is dropped). Before building QE, one needs to complete the recipe for ELPA-2016.11.001. Please note that ELPA-2017.05.001 is **not** supported!

```
wget http://www.qe-forge.org/gf/download/frsrelease/240/1075/qe-6.1.tar.gz
tar xvf qe-6.1.tar.gz
cd qe-6.1
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 ${NRANKS} \
  -genv I_MPI_PIN_DOMAIN=auto \
  -genv KMP_AFFINITY=scatter,granularity=fine,1 \
  -genv OMP_NUM_THREADS=${NTHREADS_PER_RANK} \
  /path/to/pw.x \<command-line-arguments\>
```

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