LIBXSMM

Library for small matrix-matrix multiplications targeting Intel Architecture (x86). The initial version of the library was targeting the Intel Xeon Phi coprocessor (an instance of the Intel Many Integrated Core Architecture "MIC") particularly by using KNC intrinsic functions (called KNCni or IMCI). Today, the library reaches the Many Integrated Core Architecture as well as other hardware which is capable of executing Intel Advanced Vector Extensions 512 (Intel AVX-512). Please also have a look at the collection of upcoming enhancements.

The library provides a sophisticated dispatch mechanism (see More Details) which is also targeting other instruction sets (beside of the Intrinsic code path). The library can be also compiled to "MIC native code" which is able to run self-hosted as well as in an offloaded code region (via a FORTRAN directive or via C/C++ preprocessor pragma). The prerequisite for offloading the code is to compile it to position-independent (PIC) code even when building a static library.

Performance: the presented code is by no means "optimal" or "best-performing" - it just uses Intrinsics. In fact, a well-optimizing compiler may arrange better code compared to what is laid out via the library's Python scripts. The latter can be exploited by just relying on the "inlinable code" and by not generating specialized functions.

Interface

The interface of the library is *generated* according to the Build Instructions (therefore the header file 'include/libxsmm.h' is **not** stored in the code repository). The generated interface also defines certain preprocessor symbols to store the properties the library was built for. For example, LIBXSMM_ROW_MAJOR and LIBXSMM_COL_MAJOR are used to mark down the storage order.

To perform the matrix-matrix multiplication $cm \text{ x } n = cm \text{ x } n + am \text{ x } k * bk \text{ x } n$, one of the following interfaces can be used:

```
/** If non-zero function pointer is returned, call (*function)(M, N, K). */
libxsmm_smm_function libxsmm_smm_dispatch(int m, int n, int k);
libxsmm_dmm_function libxsmm_dmm_dispatch(int m, int n, int k);
/** Automatically dispatched matrix-matrix multiplication. */
void libxsmm_smm(int m, int n, int k, const float* a, const float* b, float* c);
void libxsmm_dmm(int m, int n, int k, const double* a, const double* b, double* c);
/** Non-dispatched matrix-matrix multiplication using inline code. */
void libxsmm_simm(int m, int n, int k, const float* a, const float* b, float* c);
void libxsmm_dimm(int m, int n, int k, const double* a, const double* b, double* c);
/** Matrix-matrix multiplication using BLAS. */
void libxsmm_sblasmm(int m, int n, int k, const float* a, const float* b, float* c);
void libxsmm_dblasmm(int m, int n, int k, const double* a, const double* b, double* c);
```

With C++ function overloading, the library allows to omit the 's' and 'd' denoting the numeric type in the above C interface. Further, a type 'libxsmm_mm_dispatch<*type*>' can be used to instantiate a functor rather than making a distinction for the numeric type in 'libxsmm_?mm_dispatch'.

Build Instructions

To compile the library run:

```
make
```

The interface is produced inside of the 'include' directory. The library archives are produced inside of the 'lib' directory with the 'mic' subdirectory containing the native library and the 'intel64' folder storing the hybrid archive containing host and MIC code.

To remove intermediate files use:

```
make clean
```

or to remove all generated files including the interface and library archive files:

```
make realclean
```

The usual `make install` is simply a shortcut for `make; make clean`.

The library can be configured to accept row-major (default) or column-major order matrices. Change the variable ROW_MAJOR inside of the Makefile (0 for column-major, and row-major order otherwise), or build the library in the following way to configure the column-major format:

```
make ROW_MAJOR=0
```

To specialize LIBXSMM for certain matrix sizes (M, N, and K values), one can adjust the variables inside of the Makefile or for example build in the following way:

```
make INDICES_M="2␣4" INDICES_N="1" INDICES_K="$(echo␣$(seq␣2␣5))"
```

The above example generates the following (M,N,K) values:

```
(2,1,2), (2,1,3), (2,1,4), (2,1,5),
(4,1,2), (4,1,3), (4,1,4), (4,1,5)
```

More Details

The function ‘libxsmm_?mm_dispatch’ helps to amortize the cost of the dispatch when multiple calls with the same M, N, and K are needed. In contrast, the automatic code dispatch uses three levels:

1. Specialized routine,
2. Inlined code, and
3. BLAS library call.

All three levels are accessible directly (see Interface) in order to allow a customized code dispatch. The level 2 and 3 may be supplied by the Intel Math Kernel Library (Intel MKL) 11.2 DIRECT CALL feature. Beside of the generic interface, one can call a specific kernel e.g., ‘libxsmm_dmm_4_4_4’.

Further, the preprocessor symbol LIBXSMM_MAX_MNK denotes the largest problem size (M x N x K) that belongs to level (1) and (2), and therefore determines if a matrix-matrix multiplication falls back to level (3) calling the BLAS library linked with LIBXSMM. This threshold can be configured using for example:

```
make THRESHOLD=$((24 * 24 * 24))
```

The maximum between the given threshold and the largest requested specialization (according to INDICES_M, INDICES_N, and INDICES_K) defines the value of LIBXSMM_MAX_MNK.