Interface for Matrix Multiplication

Overview

The interface of the library is *generated* per the Build Instructions, and it is therefore **not** stored in the code repository. Instead, one may have a look at the code generation template files for C/C++ and FORTRAN.

To initialize the dispatch-table or other internal resources, an explicit initialization routine helps to avoid lazy initialization overhead when calling LIBXSMM for the first time. The library deallocates internal resources at program exit, but also provides a companion to the afore mentioned initialization (finalize).

```
/** Initialize the library; pay for setup cost at a specific point. */
void libxsmm_init(void);
/** De-initialize the library and free internal memory (optional). */
void libxsmm_finalize(void);
```

Small Matrix Multiplication (SMM)

To perform the dense matrix-matrix multiplication $C_{mxn} = alpha \cdot A_{mxk} \cdot B_{kxn} + beta \cdot C_{mxn}$, the full-blown GEMM interface can be treated with "default arguments" (which is deviating from the BLAS standard, however without compromising the binary compatibility).

```
/** Automatically dispatched dense matrix multiplication (single/double-precision, C code). */
libxsmm_?gemm(NULL/*transa*/, NULL/*transb*/, &m/*required*/, &n/*required*/, &k/*required*/,
  NULL/*alpha*/, a/*required*/, NULL/*lda*/, b/*required*/, NULL/*ldb*/,
  NULL/*beta*/, c/*required*/, NULL/*ldc*/);
/** Automatically dispatched dense matrix multiplication (C++ code). */
libxsmm_gemm(NULL/*transa*/, NULL/*transb*/, m/*required*/, n/*required*/, k/*required*/,
  NULL/*alpha*/, a/*required*/, NULL/*lda*/, b/*required*/, NULL/*ldb*/,
  NULL/*beta*/, c/*required*/, NULL/*ldc*/);
```

For the C interface (with type prefix 's' or 'd'), all arguments including m, n, and k are passed by pointer. This is needed for binary compatibility with the original GEMM/BLAS interface. The C++ interface is also supplying overloaded versions where m, n, and k can be passed by-value (making it clearer that m, n, and k are non-optional arguments).

The FORTRAN interface supports optional arguments (without affecting the binary compatibility with the original BLAS interface) by allowing to omit arguments where the C/C++ interface allows for NULL to be passed.

```
! Automatically dispatched dense matrix multiplication (single/double-precision).
CALL libxsmm_?gemm(m=m, n=n, k=k, a=a, b=b, c=c)
! Automatically dispatched dense matrix multiplication (generic interface).
CALL libxsmm_gemm(m=m, n=n, k=k, a=a, b=b, c=c)
```

For convenience, a BLAS-based dense matrix multiplication (`libxsmm_blas_gemm`) is provided for all supported languages which is simply re-exposing the underlying GEMM/BLAS implementation. The BLAS-based GEMM might be useful for validation/benchmark purposes, and more important as a fallback when building an application-specific dispatch mechanism.

```
/** Automatically dispatched dense matrix multiplication (single/double-precision). */
libxsmm_blas_?gemm(NULL/*transa*/, NULL/*transb*/, &m/*required*/, &n/*required*/, &k/*required*/,
  NULL/*alpha*/, a/*required*/, NULL/*lda*/, b/*required*/, NULL/*ldb*/,
  NULL/*beta*/, c/*required*/, NULL/*ldc*/);
```

A more recently added variant of matrix multiplication is parallelized based on the OpenMP standard. These routines will open an internal parallel region and rely on "classic" thread-based OpenMP. If these routines are called from inside of a parallel region, the parallelism will be based on tasks (OpenMP 3.0). Please note that all OpenMP-based routines are hosted by the extension library (libxsmmext), which keeps the main library agnostic with respect to a threading runtime.

```
/** OpenMP parallelized dense matrix multiplication (single/double-precision). */
libxsmm_?gemm_omp(&transa, &transb, &m, &n, &k, &alpha, a, &lda, b, &ldb, &beta, c, &ldc);
```

Manual Code Dispatch and Batched Matrix Multiplication

Successively calling a kernel (i.e., multiple times) allows for amortizing the cost of the code dispatch. Moreover, to customize the dispatch mechanism, one can rely on the following interface. Overloaded function signatures are provided and allow to omit arguments (C++ and FORTRAN), which are then derived from the configurable defaults.

```
/** If non-zero function pointer is returned, call (*function_ptr)(a, b, c). */
libxsmm_smmfunction libxsmm_smmdispatch(int m, int n, int k,
  const int* lda, const int* ldb, const int* ldc,
  const float* alpha, const float* beta,
  const int* flags, const int* prefetch);
/** If non-zero function pointer is returned, call (*function_ptr)(a, b, c). */
libxsmm_dmmfunction libxsmm_dmmdispatch(int m, int n, int k,
  const int* lda, const int* ldb, const int* ldc,
  const double* alpha, const double* beta,
  const int* flags, const int* prefetch);
```

In C++, `libxsmm_mmfunction<type>` can be used to instantiate a functor rather than making a distinction between numeric types per type-prefix (see samples/smm/specialized.cpp).

```
libxsmm_mmfunction<T> xmm(m, n, k); /* generates or dispatches the code specialization */
if (xmm) { /* JIT'ted code */
  for (int i = 0; i < n; ++i) { /* perhaps OpenMP parallelized */
    xmm(a+i*asize, b+i*bsize, c+i*csize); /* already dispatched */
  }
}
```

Similarly in FORTRAN (see samples/smm/smm.f), a generic interface (`libxsmm_mmdispatch`) can be used to dispatch a `LIBXSMM_?MMFUNCTION`, and the encapsulated PROCEDURE POINTER can be called via `libxsmm_call`. Beside of dispatching code, one can also call any statically generated kernels (e.g., `libxsmm_dmm_4_4_4`) using the prototype functions included with the FORTRAN and C/C++ interface.

```
TYPE(LIBXSMM_DMMFUNCTION) :: xmm
CALL libxsmm_dispatch(xmm, m, n, k)
IF (libxsmm_available(xmm)) THEN
  DO i = LBOUND(c, 3), UBOUND(c, 3) ! perhaps OpenMP parallelized
    CALL libxsmm_dmmcall(xmm, a(:,:,i), b(:,:,i), c(:,:,i))
  END DO
END IF
```

In case of batched SMMs, it can be beneficial to supply "next locations" such that the upcoming operands are prefetched ahead of time. The "prefetch strategy" is requested at dispatch-time of a kernel. A strategy other than `LIBXSMM_PREFETCH_NONE` turns the signature of a JIT'ted kernel into a function with six-arguments (`a,b,c, pa,pb,pc` instead of `a,b,c`). To defer the decision about the strategy to a CPUID-based mechanism, one can choose `LIBXSMM_PREFETCH_AUTO`.

```
int prefetch = LIBXSMM_PREFETCH_AUTO;
int flags = 0; /* LIBXSMM_FLAGS */
libxsmm_dmmfunction xmm = NULL;
double alpha = 1, beta = 0;
xmm = libxsmm_dmmdispatch(23/*m*/, 23/*n*/, 23/*k*/,
  NULL/*lda*/, NULL/*ldb*/, NULL/*ldc*/,
  &alpha, &beta, &flags, &prefetch);
```

Above, pointer-arguments of `libxsmm_dmmdispatch` can be NULL (or OPTIONAL in FORTRAN): for LDx this means a "tight" leading dimension, alpha, beta, and flags are given by a default value (which is selected at compile-time), and for the prefetch strategy a NULL-argument refers to "no prefetch" (which is equivalent to an explicit `LIBXSMM_PREFETCH_NONE`).

Call Wrapper

Since the library is binary compatible with existing GEMM calls (BLAS), these calls can be replaced at link-time or intercepted at runtime of an application such that LIBXSMM is used instead of the original BLAS library. There are two cases to consider:

- An application which is linked statically against BLAS requires to wrap the 'sgemm_' and the 'dgemm_' symbol (an alternative is to wrap only 'dgemm_'), and a special build of the libxsmm(ext) library is required (`make WRAP=1` to to wrap SGEMM and DGEMM, or `make WRAP=2` to wrap only DGEMM):
  `gcc [...] -Wl,--wrap=sgemm_,--wrap=dgemm_ /path/to/libxsmmext.a /path/to/libxsmm.a /path/to/your_regular_blas.a`
  Relinking the application as shown above can often be accomplished by copying, pasting, modifying the linker command, and then re-invoking the modified link step. This linker command may appear as console output of the application's "make" command (or a similar build system).
  The static link-time wrapper technique may only work with a GCC tool chain (GNU Binutils: `ld`, or `ld` via compiler-driver), and it has been tested with GNU GCC, Intel Compiler, and Clang. However, this does not

work under Microsoft Windows (even when using the GNU tool chain), and it may not work under OS X (Compiler 6.1 or earlier, later versions have not been tested).

- An application which is dynamically linked against BLAS allows for intercepting the GEMM calls at startup time (runtime) of the unmodified executable by using the LD_PRELOAD mechanism. The shared library of LIBXSMM (`make STATIC=0`) allows to intercept the GEMM calls of the application:
  ```
  LD_PRELOAD=/path/to/libxsmmext.so ./myapplication
  ```

NOTE: Using the same multiplication kernel in a consecutive fashion (batch-processing) allows to extract higher performance, when using LIBXSMM's native programming interface.