ulm university universität
uulm

Institut of Numerical Mathematics
Dr. Andreas F. Borchert and Dr. Michael C. Lehn
Mladjan Radic

1 December 2017
Quiz 5

High Performance Computing I (WS 2016/2017)

Deadline: 8 December 2017, 2pm

Thrilled by our latest benchmarks, we want to propose our GEMM implementation to the real word (i. e. to people outside the math department). However, we learn that in the real world many people use Fortran code. We reject the idea to rewrite our code in Fortran and instead provide a Fortran interface (have a look at the skeleton on the last page).

- Start with the interfaces *ulm_sgemm* and *ulm_dgemm* for element types *float* and *double*, respectively. In the Fortran world matrices are always stored col-major. Because of that the interface only requires the increment for columns (which is called *leading dimension*).

  The interfaces and functionality should be compatible with the reference implementation from netlib.org. The functionality for dgemm is there described as

  > *DGEMM performs one of the matrix−matrix operations*
  >
  > $C := alpha*op(A)*op(B) + beta*C,$
  >
  > *where* $op(X)$ *is one of*
  >
  > $op(X) = X$ **or** $op(X) = X**T,$
  >
  > *alpha* **and** *beta are scalars*, **and** A, B **and** C *are matrices*, *with* $op(A)$
  > *an m by k matrix*, $op(B)$ *a k by n matrix* **and** C *an m by n matrix*.

  We learn from the real world that a character is used in the interface to specify whether a matrix is supposed to be transposed or not, i.e. *'n'* or *'N'* means *not-transposed* and *'t'* or *'T'* means *transposed*.

  We finally learn that function parameters in Fortran are passed by reference. For our C interface that means that all parameters are passed as pointers.

- Unfortunately the real world is complex. The functionality for zgemm (and analogously for *cgemm*) is described as

ZGEMM performs one of the matrix−matrix operations

$C := alpha*op(A)*op(B) + beta*C,$

where op(X) is one of

$op(X) = X$ **or** $op(X) = X**T$ **or** $op(X) = conjg(X**T)$,

alpha **and** *beta are scalars*, **and** *A*, *B* **and** *C are matrices*, *with op(A)*
an m by k matrix, *op(B) a k by n matrix* **and** *C an m by n matrix*.

In this interface, the characters *'c'* and *'C'* indicate that the operand is *conjugate transposed*.

We are told that a vector with complex-double values just gets passed as pointer to double values (and a vector with complex-float values as pointer to float). Bravely, we decide to use in our interface a C cast operator to convert these pointers to *std*::*complex* <**double**>*.

Shockingly we note that for *cgemm* and *zgemm* we can not cover the requirements with our current implementation. But after we calm down we start to smile again as we realise that only minor adjustments are necessary.

Implement the Fortran interface for the GEMM implementation developed in session 12. Just include the reference implementation for the micro kernel. Your submitted file "quiz05.cpp" should compile on Thales without warnings to an object file with:

```
thales$ g++ -Wall -std=c++11 -m64 -c quiz05.cpp
```

Please submit your program "quiz05.cpp" as follows:

```
thales$ submit hpc quiz05 quiz05.cpp
```

You can extend the following skeleton with your implementation:

Listing 1: Skeleton for “quiz05.cpp”

```
extern "C" {

void
ulm_sgemm_(const char *transA, const char *transB,
           const int *m, const int *n, const int *k, const float *alpha,
           const float *A, const int *ldA, const float *B, const int *ldB,
           const float *beta, float *C, const int *ldC)
{
    // call your C++ implementation
}

void
ulm_dgemm_(const char *transA, const char *transB,
           const int *m, const int *n, const int *k, const double *alpha,
           const double *A, const int *ldA, const double *B, const int *ldB,
           const double *beta, double *C, const int *ldC)
{
    // call your C++ implementation
}

void
ulm_cgemm_(const char *transA, const char *transB,
           const int *m, const int *n, const int *k, const float *alpha,
           const float *A, const int *ldA, const float *B, const int *ldB,
           const float *beta, float *C, const int *ldC)
{
    // call your C++ implementation
}

void
ulm_zgemm_(const char *transA, const char *transB,
           const int *m, const int *n, const int *k, const double *alpha,
           const double *A, const int *ldA, const double *B, const int *ldB,
           const double *beta, double *C, const int *ldC)
{
    // call your C++ implementation
}

} // extern "C"
```