Chapter 1

Introduction

1.1 General remarks on MAGMA MIC

MAGMA is an abbreviation for Matrix Algebra for GPU and Multicore Architectures (`http://icl.cs.utk.edu/magma/`). It is a collection of linear algebra routines. MAGMA version for Many Integrated Core processors - MAGMA MIC includes (but is not limited to):

- MIC version of BLAS
- LU, QR and Cholesky factorization.
- Hessenberg reduction.
- Linear solvers based on LU, QR and Cholesky decompositions.
- Eigenvalue and singular value problem solvers.

A more detailed information on the procedures contained in MAGMA MIC can be found in the table of contents. The most comprehensive documentation can be found in `magmamic-x.y.z/src` directory. Let us notice that the source files in this directory contain a precise syntax description of MAGMA MIC functions, so we do not repeat this information in our text (the syntax is also easily available online). Instead, we present a series of examples how to use the library.
All subprograms have four versions corresponding to four data types:

- `s - float` – real single-precision
- `d - double` – real double-precision,
- `c - magmaFloatComplex` – complex single-precision,
- `z - magmaDoubleComplex` – complex double-precision.

For example `magma_<t>gemv` is a template which can represent `magma_sgemv`, `magma_dgemv`, `magma_cgemv` or `magma_zgemv`.

- We shall restrict our examples to the most popular real and complex, single and double precision versions. All examples contain computation times for easy comparison.

- Ideally, we should check for errors on every function call. Unfortunately such an approach doubles the length of our sample codes (which aim to be as short as possible). Since our set of sample code is over 400 pages long, we decided to ignore the error checking and focus on the explanations, which cannot be found in the syntax description. Let us underline that the codes in the `magmamic-x.y.z/testing` directory contain the appropriate error checking, but are essentially longer than ours.

- To obtain more compact explanations, in our examples we restrict the full generality of MAGMA MIC to the special case where the stride between consecutive elements is equal to 1. MAGMA MIC allows for more flexible approach, giving the user access to sub-matrices an sub-vectors. The corresponding generalizations can be found in syntax descriptions in `magmamic-x.y.z/src` directory.

1.2 Remarks on installation and compilation

1.2.1 MAGMA MIC

MAGMA MIC can be downloaded from the page `http://icl.cs.utk.edu/magma/software/index.html`. The `magmamic-x.y.z.tar.gz` archive contains the `magmamic-x.y.z` directory. Inside, there is the `README` file with installation instructions. The user needs to create `make.inc` and specify the path to BLAS and LAPACK packages in this file. Sample `make.inc` files can be found in `magmamic-x.y.z` directory. After making desired changes to the `make.inc` file, running

```
make
```

creates `libmicmagma.a` in `magmamic-x.y.z/lib` subdirectory as well as testing drivers in `magmamic-x.y.z/testing`.

All MAGMA MIC examples in this document were compiled in a separate subdirectory of `magmamic-x.y.z` using the following command:

```
$ icpc ex00_*_magma.cpp -I../include -L../lib -lmicmagma -mkl \
       -DHAVE_MIC -lscif -lcoi_host
```

In the examples which do not use MKL, the inclusion of this library i.e `-mkl` is not necessary. Compilation in a different directory requires changes in the paths. To execute the obtained binary, the export of MAGMA MIC path may be required, for example:

```
export MAGMA_PATH=~/magmamic-1.4.0
./a.out
```

1.2.2 MKL

Intel® Math Kernel Library can be obtained in a number of ways, for example as a part of Intel® Parallel Studio XE or Intel® System Studio XE https://software.intel.com/en-us/intel-mkl/try-buy.
Since MKL has an extensive documentation, we do not repeat the syntax descriptions (which are easily available online) and restrict ourselves to code samples.
All MKL examples for CPU in this document were compiled using the following command:

```
icpc -O3 -mkl exNN_*_mkl.cpp
```

The same examples can be compiled for Xeon Phi with the help of:

```
icpc -O3 -mkl -mmic exNN_*_mkl.cpp
```

The obtained binaries for MIC were executed in native mode using:

```
export SINK_LD_LIBRARY_PATH=
/opt/intel/lib/mic/:/opt/intel/mkl/lib/mic/

micnativeloadex ./a.out  -e "KMP_AFFINITY=compact"
```

The paths used in this example must be replaced by the appropriate ones for a given system and the `export` command does not need to be repeated before the subsequent `micnativeloadex` calls.

Remark. The binaries compiled for MIC can be executed on the device in a different way. In the next subsection (p. 7), using the executable `xlinpack_mic` as an example, we present a detailed description how to do it.

Remark. Recommendations to choose the right MKL usage model for Xeon Phi are formulated for example in software.intel.com/en-us/articles/recommendations-to-choose-the-right-mkl-usage-model-for-xeon-phi.

Remark. The algorithms from last chapters are not good candidates for native mode computations (the heterogeneous approach is more suitable) but for completeness we have performed the corresponding calculations.

1.2.3 Random matrix generation

All random matrices in this text were generated with the use of Vector Statistical Library (VSL). Definitions of the corresponding functions are provided in the appendix at the end of this document.
In the examples which use random matrices, before the `main` function we have included the line:

```
#include "randomize.cc"   // rand.matr.gen.from appendix
```

so, the file `randomize.cc` (or `randomize2.cc`) from appendix is assumed to be present in the working directory. Of course, that line can be replaced by the user's custom code.