THE CUBLAS LIBRARY: VERSION R15

Will Landau, Prof. Jarad Niemi

WHAT IS CULA

CULA is another set of gpu-accelerated linear algebra libraries. Essentially, it's the CUDA equivalent of LAPACK.

Its routines include:

- Utilities for initializing matrices, copying them, scaling them, etc.
- Matrix multiplication
- Factorizations: LU, QR, RQ, QL, SVD, and Cholesky
- Solving systems of linear equations (which gives you matrix inversion)
- Solving least squares problems
- Eigenproblem solvers

CULA INTERFACES

- **Standard**
  - Each CULA function micromanages GPU resources so the user doesn't have to.
  - Matrix arguments of CULA functions are pointers to CPU memory.
  - Activate by including the header, "cula_lapack.h".
- **Device**:
  - The user micromanages GPU resources as with CUBLAS.
  - Matrix arguments of CULA functions are pointers to GPU memory.
  - Activate by including the header, "cula_lapack_device.h"

There's also a "convenience header" called "cula.h", which includes both "cula_lapack.h and "cula_lapack_device.h".

Note: although convenient, the standard interface is often slow because every single function writes back and forth between the CPU and the GPU.

COMPILING WITH CULA

1. Include the header file in your source: "cula_lapack.h" for the standard interface, "cula_lapack_device.h" for the device interface, or "cula.h" for both.

2. Enter in the command line:

```
nvcc -I /usr/local/cula/include -L /usr/local/cula/lib64 -lcula_core -lcula_lapack
         -lcublas -lcudart your_source.cu -o your_binary
```

Note:

- `-I /usr/local/cula/include` tells the compiler, `nvcc`, where to find the `.h` files.
- `-L /usr/local/cula/lib64` tells `nvcc` to use the 64-bit version of the CULA library.
- `-lcula_core -lcula_lapack -lcublas -lcudart` tells `nvcc` to link the CULA library to your binary.

EXAMPLE

```
[landau@impact1 ~]$ cd gpuIntroduction/CUDA_C_sandbox/CULA/Initi
alize_and_Shutdown/
[landau@impact1 Initialize_and_Shutdown]$ ls
inst.cu
[landau@impact1 Initialize_and_Shutdown]$ nvcc -I /usr/local/cul
a/include -L /usr/local/cula/lib64 -lcula_core -lcula_lapack -lc
ublas -lcudart inst.cu -o inst
[landau@impact1 Initialize_and_Shutdown]$ ls
inst  inst.cu
[landau@impact1 Initialize_and_Shutdown]$ ./inst
[landau@impact1 Initialize_and_Shutdown]$
```

FUNCTION NAME CONVENTIONS

CULAPACK PREFIX

Interface	Culapack Prefix	Example function
Standard	`cula`	`culaSgeqrf`
Device	`culaDevice`	`culaDeviceSgeqrf`

DATA TYPE

Prefix	Data Type	CULA Standard Interface Type	CULA Device Interface Type
`S`	Single Precision Real	`culaFloat`	`culaDeviceFloat`
`C`	Single Precision Complex	`culaFloatComplex`	`culaDeviceFloatComplex`
`D`	Double Precision Real	`culaDouble`	`culaDeviceDouble`
`Z`	Double Precision Complex	`culaDoubleComplex`	`culaDeviceDoubleComplex`

MATRIX TYPE

Abbreviation	Matrix Type
bd	Bidiagonal
ge	General
gg	General matrices, generalized problem
he	Hermitian Symmetric
or	(Real) Orthogonal
sb	Symmetric Band
sy	Symmetric
tr	Triangular
un	(Complex) Unitary

COMPUTATIONAL ROUTINE

Abbreviation	Computation Routine
trf	Compute a triangular factorization
sv	Factor the matrix and solve a system of equations
qrf	Compute a QR factorization without pivoting
svd	Compute the singular value decomposition
ls	Solve over- or under-determined linear system

WHAT'S IN CULA?

FRAMEWORK FUNCTIONS

`culaInitialize`	Initializes CULA. Must be called before using any other function.
`culaShutdown`	Shuts down CULA. Must be called to deallocate CULA internal data.
`culaGetLastStatus`	Returns the last status code returned from a CULA function.
`culaGetStatusString`	Given a culaStatus number, returns a readable error string.
`culaGetStatusAsString`	Returns the culaStatus name as a string.
`culaGetErrorInfo`	Returns extended information about the last error or zero if it is unavailable
`culaGetErrorInfoString`	Associates a culaStatus and culaInfo with a readable error string.
`culaFreeBuffers`	Releases any memory buffers stored internally by CULA
`culaGetVersion`	Reports the version number of CULA.
`culaGetCudaMinimumVersion`	Reports the CUDA_VERSION that the running version of CULA was compiled against, which indicates the minimum version of CUDA that is required to use this library.
`culaGetCudaRuntimeVersion`	Reports the version of the CUDA runtime that the operating system linked against when the program was loaded.
`culaGetCudaDriverVersion`	Reports the version of the CUDA driver installed on the system.
`culaGetCublasMinimumVersion`	Reports the CUBLAS_VERSION that the running version of CULA was compiled against, which indicates the mini- mum version of CUBLAS that is required to use this library.

`culaGetCublasRuntimeVersion`	Reports the version of the CUBLAS runtime that operating system linked against when the program was loaded.
`culaGetDeviceCount`	Reports the number of GPU devices Can be called before `culaInitialize`
`culaSelectDevice`	Selects a device with which CULA will operate. Must go after `culaInitialize`
`culaGetExecutingDevice`	Returns a pointer to the id of the GPU device executing CULA.
`culaGetOptimalPitch`	Calculates a pitch that is optimal for CULA when using the device interface parameters.
`culaGetDeviceInfo`	Prints information to a buffer about a specified device.
`culaDeviceMalloc`	Allocates memory on the device.
`culaDeviceFree`	Frees memory that has been allocated with `culaDeviceMalloc`.

AUXILIARY FUNCTIONS

Matrix Type	Operation	S	C	D	Z
General	Copy from one Matrix into another	SLACPY	CLACPY	DLACPY	ZLACPY
	Convert a matrix's precision	SLAG2D	CLAG2Z	DLAG2S	ZLAG2D
	Apply a block reflector to a matrix	SLARFB	CLARFB	DLARFB	ZLARFB
	Generate an elementary reflector	SLARFG	CLARFG	DLARFG	ZLARFG
	Generate a vector of plane rotations	SLARGV	CLARGV	DLARGV	ZLARGV
	Apply a vector of plane rotations	SLARTV	CLARTV	DLARTV	ZLARTV
	Multiple a matrix by a scalar	SLASCL	CLASCL	DLASCL	ZLASCL
	Initialize a matrix	SLASET	CLASET	DLASET	ZLASET
	Apply a sequence of plane rotations	SLASR	CLASR	DLASR	ZLASR
Symmetric	Apply a vector of plane rotations	SLAR2V	CLAR2V	DLAR2V	ZLAR2V
Triangular	Triangular precision conversion	SLAT2D	CLAT2Z	DLAT2S	DLAT2Z

Conjugate (general matrix)	CgeConjugate, ZgeConjugate
Conjugates either the upper or lower triangle of a matrix, with the option to include the diagonal or not	CtrConjugate, ZtrConjugate
Check a matrix for invalid values	SgeNancheck, DgeNancheck, CgeNancheck, ZgeNancheck
out-of-place matrix transpose: $A^T \mapsto B$	SgeTranspose, DgeTranspose, CgeTranspose, ZgeTranspose
in-place matrix transpose: $A^T \mapsto A$	SgeTransposeInplace, DgeTransposeInplace, CgeTransposeInplace, ZgeTransposeInplace
out-of-place matrix transpose + conjugate all elements	SgeTransposeConjugate, DgeTransposeConjugate, CgeTransposeConjugate, ZgeTransposeConjugate
in-place matrix transpose + conjugate all elements	SgeTransposeConjugateInplace, DgeTransposeConjugateInplace, CgeTransposeConjugateInplace, ZgeTransposeConjugateInplace

MATRIX MULTIPLICATIONS

Matrix Type	Operation	S	C	D	Z
General	Matrix-matrix multiply	SGEMM	CGEMM	DGEMM	ZGEMM
	Matrix-vector multiply	SGEMV	CGEMV	DGEMV	ZGEMV
Triangular	Triangular matrix-matrix multiply	STRMM	CTRMM	DTRMM	ZTRMM
	Triangular matrix solve	STRSM	CTRSM	DTRSM	ZTRSM
Symmetric	Symmetric matrix-matrix multiply	SSYMM	CSYMM	DSYMM	ZSYMM
	Symmetric rank 2k update	SSYR2K	CSYR2K	DSYR2K	ZSYR2K
	Symmetric rank k update	SSYRK	CSYRK	DSYRK	ZSYRK
Hermitian	Hermitian matrix-matrix multiply		CHEMM		ZHEMM
	Hermitian rank 2k update		CHER2K		ZHER2K
	Hermitian rank k update		CHERK		ZHERK

FACTORIZATIONS

Matrix Type	Operation	S	C	D	Z
General	Factorize and solve	`SGESV`	`CGESV`	`DGESV`	`ZGESV`
	Factorize and solve with iterative refinement			`DSGESV`	`ZCGESV`
	LU factorization	`SGETRF`	`CGETRF`	`DGETRF`	`ZGETRF`
	Solve using LU factorization	`SGETRS`	`CGETRS`	`DGETRS`	`ZGETRS`
	Invert using LU factorization	`SGETRI`	`CGETRI`	`DGETRI`	`ZGETRI`
Positive Definite	Factorize and solve	`SPOSV`	`CPOSV`	`DPOSV`	`ZPOSV`
	Cholesky factorization	`SPOTRF`	`CPOTRF`	`DPOTRF`	`ZPOTRF`
Triangular	Invert triangular matrix	`STRTRI`	`CTRTRI`	`DTRTRI`	`ZTRTRI`
	Solve triangular system	`STRTRS`	`CTRTRS`	`DTRTRS`	`ZTRTRS`
Banded	LU factorization	`SGBTRF`	`CGBTRF`	`DGBTRF`	`ZGBTRF`
Pos Def Banded	Cholesky factorization	`SPBTRF`	`CPBTRF`	`DPBTRF`	`ZPBTRF`

ORTHOGONAL FACTORIZATIONS

Matrix Type	Operation	S	C	D	Z
General	QR factorization	SGEQRF	CGEQRF	DGEQRF	ZGEQRF
	QR factorize and solve	SGEQRS	CGEQRS	DGEQRS	ZGEQRS
	Generate Q from QR factorization	SORGQR	CUNGQR	DORGQR	ZUNGQR
	Multiply matrix by Q from QR factorization	SORMQR	CUNMQR	DORMQR	ZUNMQR
General	LQ factorization	SGELQF	CGELQF	DGELQF	ZGELQF
	Generate Q from LQ factorization	SORGLQ	CUNGLQ	DORGLQ	ZUNGLQ
	Multiply matrix by Q from LQ factorization	SORMLQ	CUNMLQ	DORMLQ	ZUNMLQ
General	RQ factorization	SGERQF	CGERQF	DGERQF	ZGERQF
	Multiply matrix by Q from RQ factorization	SORMRQ	CUNMRQ	DORMRQ	ZUNMRQ
General	QL factorization	SGEQLF	CGEQLF	DGEQLF	ZGEQLF
	Generate Q from QL factorization	SORGQL	CUNGQL	DORGQL	ZUNGQL
	Multiply matrix by Q from QL factorization	SORMQL	CUNMQL	DORMQL	ZUNMQL

SINGULAR VALUE DECOMPOSITIONS

Matrix Type	Operation	S	C	D	Z
General	General Singular Value Decomposition	SGESVD	CGESVD	DGESVD	ZGESVD
	Bidiagonal reduction	SGEBRD	CGEBRD	DGEBRD	ZGEBRD
	Generate Q from bidiagonal reduction	SORGBR	CUNGBR	DORGBR	ZUNGBR
Bidiagonal	Find singular values and vectors	SBDSQR	CBDSQR	DBDSQR	ZBDSQR

LEAST SQUARES

Matrix Type	Operation	S	C	D	Z
General	General least squares solve	SGELS	CGELS	DGELS	ZGELS
	Equality-constrained least squares	SGGLSE	CGGLSE	DGGLSE	ZGGLSE

SYMMETRIC EIGENPROBLEMS

Matrix Type	Operation	S	C	D	Z
Symmetric	Symmetric Eigenproblem solver	SSYEV		DSYEV	
	Symmetric Eigenproblem solver (expert)	SSYEVX		DSYEVX	
	Symmetric band reduction	SSYRDB		DSYRDB	
Tridiagonal	Find eigenvalues using bisection	SSTEBZ		DSTEBZ	
	Find eigenvalues using QR/QL iteration	SSTEQR	CSTEQR	DSYRDB	ZSTEQR

NON-SYMMETRIC EIGENPROBLEMS

Matrix Type	Operation	S	C	D	Z
General	General Eigenproblem solver	SGEEV	CGEEV	DGEEV	ZGEEV
	Hessenberg reduction	SGEHRD	CGEHRD	DGEHRD	ZGEHRD
	Generate Q from Hessenberg reduction	SORGHR	CUNGHR	DORGHR	ZUNGHR

EXAMPLE: systemSolve

```
/*
 * CULA Example: systemSolve
 *
 * This example shows how to use a system solve for multiple data types.  Each
 * data type has its own example case for clarity.  For each data type, the
 * following steps are done:
 *
 * 1. Allocate a matrix on the host
 * 2. Initialize CULA
 * 3. Initialize the A matrix to the Identity
 * 4. Call gesv on the matrix
 * 5. Verify the results
 * 6. Call culaShutdown
 *
 * After each CULA operation, the status of CULA is checked.  On failure, an
 * error message is printed and the program exits.
 *
```

```
*
* Note: CULA Premium and double-precision GPU hardware are required to run the
* double-precision examples
*
* Note: this example performs a system solve on an identity matrix against a
* random vector, the result of which is that same random vector.  This is not
* true in the general case and is only appropriate for this example.  For a
* general case check, the product A*X should be checked against B.  Note that
* because A is modifed by GESV, a copy of A would be needed with which to do
* the verification.
*/
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <cula_lapack.h>


void checkStatus(culaStatus status)
{
    char buf[256];

    if(!status)
        return;

    culaGetErrorInfoString(status, culaGetErrorInfo(), buf, sizeof(buf));
    printf("%s\n", buf);

    culaShutdown();
    exit(EXIT_FAILURE);
}
```

```
void culaFloatExample()
{
#ifdef NDEBUG
    int N = 8192;
#else
    int N = 1024;
#endif
    int NRHS = 1;
    int i;

    culaStatus status;

    culaFloat* A = NULL;
    culaFloat* B = NULL;
    culaFloat* X = NULL;
    culaInt* IPIV = NULL;

    culaFloat one = 1.0f;
    culaFloat thresh = 1e-6f;
    culaFloat diff;
```

```
printf("-------------------\n");
printf("       SGESV\n");
printf("-------------------\n");

printf("Allocating Matrices\n");
A = (culaFloat*)malloc(N*N*sizeof(culaFloat));
B = (culaFloat*)malloc(N*sizeof(culaFloat));
X = (culaFloat*)malloc(N*sizeof(culaFloat));
IPIV = (culaInt*)malloc(N*sizeof(culaInt));
if(!A || !B || !IPIV)
    exit(EXIT_FAILURE);

printf("Initializing CULA\n");
status = culaInitialize();
checkStatus(status);
```

```c
    // Set A to the identity matrix
    memset(A, 0, N*N*sizeof(culaFloat));
    for(i = 0; i < N; ++i)
        A[i*N+i] = one;

    // Set B to a random matrix (see note at top)
    for(i = 0; i < N; ++i)
        B[i] = (culaFloat)rand();
    memcpy(X, B, N*sizeof(culaFloat));

    memset(IPIV, 0, N*sizeof(culaInt));

    printf("Calling culaSgesv\n");
    status = culaSgesv(N, NRHS, A, N, IPIV, X, N);
    checkStatus(status);

    printf("Verifying Result\n");
    for(i = 0; i < N; ++i)
    {
        diff = X[i] - B[i];
        if(diff < 0.0f)
            diff = -diff;
        if(diff > thresh)
            printf("Result check failed:  i=%d  X[i]=%f  B[i]=%f", i, X[i], B[i]);
    }

    printf("Shutting down CULA\n\n");
    culaShutdown();

    free(A);
    free(B);
    free(IPIV);
}
```

```
int main(int argc, char** argv)
{
    culaFloatExample();
    culaFloatComplexExample();

    // Note: CULA Premium is required for double-precision
#ifdef CULA_PREMIUM
    culaDoubleExample();
    culaDoubleComplexExample();
#endif

    return EXIT_SUCCESS;
}
```

Compile and run as follows:

```
[landau@impact1 systemSolve]$ ls
Makefile  systemSolve.c
[landau@impact1 systemSolve]$ make build64
sh ../checkenvironment.sh
Warning: CULA_BIN_PATH_32 is not defined
Warning: CULA_BIN_PATH_64 is not defined
Warning: CULA_LIB_PATH_32 is not defined

------------------------------------------------------------------------
Warning: Some CULA environment variables could not be found.
         This may prevent successful building of the example projects
------------------------------------------------------------------------

gcc -m64 -o systemSolve systemSolve.c -DNDEBUG -O3 -I/usr/local/cula/include -L/usr/local/cula/lib64 -lcula_co
re -lcula_lapack -lcublas -lcudart
systemSolve.c: In function ‘culaFloatComplexExample’:
systemSolve.c:223: warning: incompatible implicit declaration of built-in function ‘sqrt’
systemSolve.c: In function ‘culaDoubleComplexExample’:
systemSolve.c:399: warning: incompatible implicit declaration of built-in function ‘sqrt’
```

```
[landau@impact1 systemSolve]$ ls
Makefile  systemSolve  systemSolve.c
[landau@impact1 systemSolve]$ ./systemSolve
-------------------
       SGESV
-------------------
Allocating Matrices
Initializing CULA
Calling culaSgesv
Verifying Result
Shutting down CULA

-------------------
       CGESV
-------------------
Allocating Matrices
Initializing CULA
Calling culaCgesv
Verifying Result
Shutting down CULA
```

```
-------------------
       DGESV
-------------------
Allocating Matrices
Initializing CULA
Calling culaDgesv
Verifying Result
Shutting down CULA

-------------------
       ZGESV
-------------------
Allocating Matrices
Initializing CULA
Calling culaZgesv
Verifying Result
Shutting down CULA

[landau@impact1 systemSolve]$
```

GPU SERIES MATERIALS

These slides, a tentative syllabus for the whole series, and code are available at:

https://github.com/wlandau/gpu.

After logging into you home directory on impact1, type:

```
git clone https://github.com/wlandau/gpu
```

into the command line to download all the materials.

REFERENCES

"CULA Programmer's Guide". CULA Tools.
http://www.hpc-sol.co.jp/products/cula/doc/CULA_dense_R15/CULAProgrammersGuide_R15.pdf

"CULA Reference Manual". CULA Tools.
http://www.hpc-sol.co.jp/products/cula/doc/CULA_dense_R15/CULAReferenceManual_R15.pdf