CUBLAS LIBRARY

(part of CUDA Toolkit 4.1 — Jan 2012)

CUBLAS = CUDA BLAS (Basic Linear Algebra Subprograms)

Principles:

- as before:
  - allocate memory for vectors and matrices in the GPU memory space
  - fill them with data
  - call the desired functions
  - upload results from GPU back to the host
- CUBLAS uses column-major storage and 1-based indexing

Features of the new CUBLAS API:

- handle to the CUBLAS library context
  - allows the user to have more control over the library setup when using multiple host threads and multiple GPUs
- scalars α, β can be passed by reference on the host or device (instead of being allowed to be passed by value on host)
- scalar results can be returned by reference on the host or device (instead of being allowed to be returned by value only on host)
- each function returns an error status

CUBLAS API

```
main(){
 ---
cublasHandle_t  handle
cublasCreate(&handle)
----
cublasDestroy(&handle)
}
```

- allows the user to explicitly control the library setup when using multiple host threads and multiple GPUs.
- CUBLAS can also be used within multiple streams

CUBLAS DATATYPES

cublasHandle_t → pointer to an opaque structure holding the CUBLAS library context

cublasStatus_t → used for function status returns (all CUBLAS functions return their status)

- CUBLAS_STATUS_SUCCESS
- CUBLAS_STATUS_NOT_INITIALIZED
- CUBLAS_STATUS_ALLOC_FAILED
- CUBLAS_STATUS_EXECUTION_FAILED
- etc.

cublasOperation_t → indicates which operation needs to be performed with the dense matrix

CUBLAS_OP_N

CUBLAS_OP_T

CUBLAS_OP_C

etc.

```
cublasSetVector(int n, int elemSize, const void *x, int incx, void *devPtr,
                int incy)
```

- this function copies n elements from a vector x in host memory to vector devPtr in GPU memory space
- elements of both vectors are assumed to have the same size elemSize
- the storage spacing between consecutive elements is given by incx, incy

```
cublasGetVector(....)
```

```
cublasSetMatrix(int rows, int cols, int elemSize, const void *A, int lda,
                void *B, int ldb)
```

- copies rows x cols elements from matrix A in host memory space to matrix B in GPU memory space
- it is assumed that both matrices are stored in column-major format.

```
cublasGetMatrix(...)
```

```
cublasSetVectorAsync(...)      cublasSetMatrixAsync(...)
cublasGetVectorAsync(...)      cublasGetMatrixAsync(...)
```

LEVEL-1 functions

(scalar and vector-based operations)

cublas<type>function(...)

<type> can be 's' or 'S' → real single precision

'd' or 'D' → real double precision

'c' or 'C' → complex single precision

'z' or 'Z' → complex double precision

cublasSasum (cublasHandle_t handle, int n, const float *x, int incx, float *result)

Param.	Memory	In/out	Meaning
handle		input	handle to the CUBLAS library context.
n		input	number of elements in the vector **x**.
x	device	input	<type> vector with **n** elements.
incx		input	stride between consecutive elements of **x**.
result	host or device	output	the resulting sum, which is 0.0 if **n,incx<=0**.

cublas<type>axpy(...)

calculates $\alpha x + y$

(α: scalar, x: vector, y: vector)

stores the result into y

```
cublasStatus_t cublasSaxpy(cublasHandle_t handle, int n,
                           const float           *alpha,
                           const float           *x, int incx,
                           float                 *y, int incy)
```

Param.	Memory	In/out	Meaning
handle		input	handle to the CUBLAS library context.
alpha	host or device	input	<type> scalar used for multiplication.
n		input	number of elements in the vector **x** and **y**.
x	device	input	<type> vector with **n** elements.
incx		input	stride between consecutive elements of **x**.
y	device	in/out	<type> vector with **n** elements.
incy		input	stride between consecutive elements of **y**.

cublas<t>copy(...)

copies vector x into vector y

cublas<t>dot(...)

dot product

cublas<t>nrm2(...)

Euclidean norm

etc.

CUBLAS LEVEL-2 FUNCTIONS

cublas<t>gemv(...)

$$y = \alpha \, op(A) x + \beta y$$

$$\mathrm{op}(A) = \begin{cases} A & \text{if } \texttt{transa == CUBLAS\_OP\_N} \\ A^T & \text{if } \texttt{transa == CUBLAS\_OP\_T} \\ A^H & \text{if } \texttt{transa == CUBLAS\_OP\_C} \end{cases}$$

```
cublasStatus_t cublasSgemv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *x, int incx,
                           const float           *beta,
                           float           *y, int incy)
```

Other operations:

$$A = \alpha x y^T + A$$

$$A = \alpha (x y^T + y x^T) + A$$

etc.

- specific functions can be used if the matrix A is triangular, symmetric, etc.

- for example:

cublas<t>spmv(...)

performs . $y = \alpha A x + \beta y$

if A is an $n \times n$ symmetric matrix stored in packed format

that is, as a $n(n+1)/2$ vector $A(i,j) = AP[i + ((2n-j+1) \cdot j)/2]$

CUBLAS LEVEL-3 FUNCTIONS

(matrix-matrix operations)

cublas <t> gemm (...)

$$\underbrace{C}_{} = \alpha \overbrace{op(A)}^{m \times k} \overbrace{op(B)}^{k \times n} + \beta \overbrace{C}^{m \times n}$$

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa, cublasOperation_t transb,
                           int m, int n, int k,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *B, int ldb,
                           const float           *beta,
                           float           *C, int ldc)
```

- other functions are available when dealing with symmetric, triangular, etc matrices.