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 $\alpha, \beta$ 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

```c
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.
cublas Operation_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 (...)

cublasGetVectorAsync (...)

cublasSetMatrixAsync (...)

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)

<table>
<thead>
<tr>
<th>Param.</th>
<th>Memory</th>
<th>In/out</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td></td>
<td>input</td>
<td>handle to the CUBLAS library context.</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td>input</td>
<td>number of elements in the vector x.</td>
</tr>
<tr>
<td>x</td>
<td>device</td>
<td>input</td>
<td>&lt;type&gt; vector with n elements.</td>
</tr>
<tr>
<td>incx</td>
<td></td>
<td>input</td>
<td>stride between consecutive elements of x.</td>
</tr>
<tr>
<td>result</td>
<td>host or device</td>
<td>output</td>
<td>the resulting sum, which is 0.0 if n, incx&lt;=0.</td>
</tr>
</tbody>
</table>
Cublas<DoubleType> axpy(...)

calculates $\alpha x + y$

scalar vector 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)
```

<table>
<thead>
<tr>
<th>Param.</th>
<th>Memory</th>
<th>In/out</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>host or device</td>
<td>input</td>
<td>handle to the CUBLAS library context.</td>
</tr>
<tr>
<td>alpha</td>
<td>host or device</td>
<td>input</td>
<td>&lt;DoubleType&gt; scalar used for multiplication.</td>
</tr>
<tr>
<td>n</td>
<td>device</td>
<td>input</td>
<td>number of elements in the vector $x$ and $y$.</td>
</tr>
<tr>
<td>x</td>
<td>device</td>
<td>input</td>
<td>&lt;DoubleType&gt; vector with $n$ elements.</td>
</tr>
<tr>
<td>incx</td>
<td>device</td>
<td>input</td>
<td>stride between consecutive elements of $x$.</td>
</tr>
<tr>
<td>y</td>
<td>device</td>
<td>in/out</td>
<td>&lt;DoubleType&gt; vector with $n$ elements.</td>
</tr>
<tr>
<td>incy</td>
<td>device</td>
<td>input</td>
<td>stride between consecutive elements of $y$.</td>
</tr>
</tbody>
</table>
cublas \langle t \rangle \texttt{copy} (...)  

copies vector x into vector y

cublas \langle t \rangle \texttt{dot} (...)  

dot product

cublas \langle t \rangle \texttt{nrm2} (...)  

Euclidean norm

etc.
CUBLAS LEVEL-2 FUNCTIONS

cublas<t>gemv (...)

\[ y = \alpha \ \text{op}(A)x + \beta y \]

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

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:

  \texttt{cublas<t>spmv(...)}

  performs: \( y = \alpha A x + p y \)

  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 (...

\[ C = \alpha \circ \text{op}(A) \circ \text{op}(B) + \beta C \]

```c
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.