A Proposed Modification to the Batch BLAS Interface

Ahmad Abdelfattah
with Azzam Haidar and Stan Tomov

Workshop on Batched, Reproducible, and Reduced Precision BLAS

Friday, February 24th, 2017
Outline

1 At a Glance

2 Introduction

3 Proposed Modifications

4 Examples

5 Conclusion
Outline

1. At a Glance
2. Introduction
3. Proposed Modifications
4. Examples
5. Conclusion
What this talk is all about

- Few modifications to the batch BLAS interface
- Based on practical development experience (batched LAPACK)
- The modification is a generalization over the existing interface
  - Both can exist side-by-side
- While the work originally targeted GPUs, most of it is generic, and applies to other architectures
Outline

1. At a Glance
2. Introduction
3. Proposed Modifications
4. Examples
5. Conclusion
Motivation

This is a sample MAGMA interface

```c
void magmablas_dgemm_vbatched(
    enum transA, enum transB,
    int* m, int* n, int* k,
    double alpha, double ** Aarray, int* lda,
    double ** Barray, int* ldb,
    double beta, double ** Carray, int* ldc,
    int batchCount, magma_queue_t queue );
```

What are the issues?

- Most of the time, we are dealing with **submatrices**
- Consistent manipulation of pointer and integer arrays
- In the case of GPUs, **manipulations = kernels**
  - Potential overhead
  - Bad code readability
  - Challenging to write efficient recursive functions
A non-batched example

Trailing matrix update of an LU factorization (one matrix)

```
#define dA(i,j) (dA[ (j) * lda + (i) ])
/* some code */
magmablas_dgemm( MagmaNoTrans, MagmaNoTrans,
    N-(j+nb), N-(j+nb), nb,
    NEG_ONE, dA(j+nb, j), lda,
    dA(j, j+nb), lda,
    ONE , dA(j+nb, j+nb), lda, queue );
```
Now consider a batched example (fixed size)

We must preprocess arrays before the kernel call

```c
/* some code */
magma_displace_pointers(dA1_array, dA_array, lda, j+nb, j , batchCount, queue);
magma_displace_pointers(dA2_array, dA_array, lda, j , j+nb, batchCount, queue);
magma_displace_pointers(dA3_array, dA_array, lda, j+nb, j+nb, batchCount, queue);
magmablas_dgemm_batched( MagmaNoTrans, MagmaNoTrans,
                          N-(j+nb), N-(j+nb), nb,
                          NEG_ONE, dA1_array, lda,
                          dA2_array, lda,
                          ONE , dA3_array, lda, batchCount, queue );
```
Now consider a batched example (fixed size)

We must preprocess arrays before the kernel call

```c
/* some code */
magma_displace_pointers(dA1_array, dA_array, lda, j+nb, j , batchCount, queue);
magma_displace_pointers(dA2_array, dA_array, lda, j , j+nb, batchCount, queue);
magma_displace_pointers(dA3_array, dA_array, lda, j+nb, j+nb, batchCount, queue);
magmablas_dgemm_batched( MagmaNoTrans, MagmaNoTrans,
                            N-(j+nb), N-(j+nb), nb,
                            NEG_ONE, dA1_array, lda,
                            dA2_array, lda,
                            ONE   , dA3_array, lda, batchCount, queue );
```
Now consider a batched example (fixed size)

We must preprocess arrays before the kernel call

Fixed size: we need kernels to manipulate pointer arrays

Var. size: we also need kernels to manipulate integer arrays

```c
/* some code */
magma_displace_pointers(dA1_array, dA_array, lda, j+nb, j , batchCount, queue);
magma_displace_pointers(dA2_array, dA_array, lda, j , j+nb, batchCount, queue);
magma_displace_pointers(dA3_array, dA_array, lda, j+nb, j+nb, batchCount, queue);
magmablas_dgemm_batched( MagmaNoTrans, MagmaNoTrans,
        N-(j+nb), N-(j+nb), nb,
        NEG_ONE, dA1_array, lda,
        dA2_array, lda,
        ONE , dA3_array, lda, batchCount, queue );
```
To wrap up

- We need to manipulate pointer and integer arrays (custom kernels)
- This can be done in-place, but:
  - These arrays are inputs (EVEN FOR OUTPUT MATRICES)
  - Multiple displacements of the same array in a single call
- Many allocations are required per routines
  - Problem of asynchronicity rather than overhead
- Recursion is very difficult
  - Allocation/deallocation inside a recursive function!
Outline

1. At a Glance
2. Introduction
3. Proposed Modifications
4. Examples
5. Conclusion
Design Goals

- No manipulation of pointer/integer arrays in separate kernels
- Better code readability
- Fully asynchronous routines
  - At least for BLAS
- Full support for recursive functions
What do we propose?

More Informative APIs

- We are going to **expand the interface**
- Fixed size routines:
  - $(i, j)$ offsets for every pointer array
- Variable size routines
  - $(i, j)$ offsets for every pointer array, and
  - a scalar size per integer array

Eventually, batched kernels deduce location/size instead of just reading them
Outline

1. At a Glance
2. Introduction
3. Proposed Modifications
4. Examples
5. Conclusion
Fixed size APIs

- Every pointer array is associated with two integers that represent row and column offsets

```
/* some code */
magmablas_dgemm_batched( MagmaNoTrans, MagmaNoTrans, 
N-(j+nb), N-(j+nb), nb, 
NEG_ONE, dA_array, j+nb, j , lda, 
dA_array, j , j+nb, lda, 
ONE , dA_array, j+nb, j+nb, lda, batchCount, queue );
```
Fixed size APIs

- Every pointer array is associated with **two integers that represent row and column offsets**

```c
/* some code */
magmablas_dgemm_batched( MagmaNoTrans, MagmaNoTrans,
    N-(j+nb), N-(j+nb), nb,
    NEG_ONE, dA_array, j+nb, j , lda,
    dA_array, j , j+nb, lda,
    ONE , dA_array, j+nb, j+nb, lda, batchCount, queue );
```

Each sub-kernel computes the correct pointer
Variable size APIs

- Every pointer array is associated with two integers that represent row and column offsets
- Integer arrays remain untouched
- We need to specify extra sizes of the operation according to the biggest matrix

```c
/* some code */
magmablas_dgemm_vbatched( MagmaNoTrans, MagmaNoTrans, 
                        N_array, N_array, N_array, 
                       NEG_ONE , dA_array, j+nb, j , lda_array, 
                        dA_array, j , j+nb, lda_array, 
                       ONE       , dA_array, j+nb, j+nb, lda_array, 
                       Nmax-(j+nb), Nmax-(j+nb), nb, 
                      batchCount, queue );
```

- Similarly, every sub-kernel deduces the correct sizes based on the offsets + the extra sizes
Variable size APIs "cont."

LU update \((C = C - A \times B)\), 1st iteration, \(nb=4\), \(N_{\text{max}}=10\)

\[
\begin{array}{ccc}
\text{N=3} & \text{N=6} & \text{N=10} \\
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
A \\
\hline
C
\end{array}
\end{array}
\end{array}
\end{array}
\]

Configuration of GEMM update: \(C_{m \times n} = C_{m \times n} - A_{m \times k} \times B_{k \times n}\)

<table>
<thead>
<tr>
<th>matrix</th>
<th>size</th>
<th>requested ((m, n, k))</th>
<th>affordable ((m, n, k))</th>
<th>executed ((m, n, k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>((6, 6, 4))</td>
<td>((0, 0, 0))</td>
<td>((0, 0, 0))</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>((6, 6, 4))</td>
<td>((2, 2, 6))</td>
<td>((2, 2, 4))</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>((6, 6, 4))</td>
<td>((6, 6, 10))</td>
<td>((6, 6, 4))</td>
</tr>
</tbody>
</table>
Recursive Batched Computation

**Example: batched TRSM (fixed size)**

```c
/* some code */
const int m2 = m / 2;
const int m1 = m - m2;

magmablas_dtrsm_batched( side, uplo, transA, diag,
   m1, n, alpha,
   dA_array, 0, 0, ldda,
   dB_array, 0, 0, lddb,
   batchCount, queue );

magmablas_dgemm_batched( MagmaNoTrans, MagmaNoTrans,
   m2, n, m1,
   neg_one, dA_array, m1, 0, ldda,
   dB_array, 0, 0, lddb,
   alpha , dB_array, m1, 0, lddb,
   batchCount, queue );

magmablas_dtrsm_batched( side, uplo, transA, diag,
   m2, n, c_one,
   dA_array, m1, m1, ldda,
   dB_array, m1, 0, lddb,
   batchCount, queue );
```

A. Abdelfattah
Outline

1 At a Glance
2 Introduction
3 Proposed Modifications
4 Examples
5 Conclusion
Conclusion and Future Work

To summarize:
- The current batch BLAS API makes it challenging to develop higher level batched operations
- The new API is longer, but it can look nicer
- The new API is very efficient in recursive function

Future Directions:
- Populate the new API in several MAGMA routines (while keeping the old one)
- Look for feedback/potential improvements
Thank You!