Toward a standard Batched BLAS API

Azzam Haidar

w/ A. Abdelfatah, H. Anzt, J. Dongarra, M. Gates, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki

Innovative Computing Laboratory
University of Tennessee, Knoxville
Toward a standard Batched BLAS API

Linear Algebra on small problems are needed in many applications:

- Machine learning,
- Data mining,
- High-order FEM,
- Numerical LA,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Multi-physics problems,
- Signal processing, etc
Toward a standard Batched BLAS API

Status and goal

- Batched BLAS functionalities become a major factor in our community
  - Batched routines gradually make their steps into vendor libraries (Intel, Nvidia, etc) as well as into research software (MAGMA, Kokkos, etc)
- Today’s API differ significantly which can lead to poor portability
- Thus the community needs to make an effort to standardize the Batched BLAS API
Towards a standard Batched BLAS API

Status and goal

- Heterogeneity in the hardware (GPU, Phi, CPU) deeply complicates efforts to provide a standard interface
  - The calling interface may affect the implementation (performance) which depend on the architecture
- Our objective today, is to try to define a cross-architecture standard without a severe performance penalty
- Other API’s could be considered as auxiliary API’s or API with extra features
Toward a standard Batched BLAS API

Matrices are stored BLAS-like "the usual storage that we know"

- array of pointers: that consists of a pointer to each matrix
  - Data could belong to one memory allocation
  - Data could be anywhere, different allocations
  - Matrices could be equidistant or not from each other
  - Is suitable for CPU, GPU, Phi
  - Accommodate most of the cases

- User has to fill-up the array of pointers
Toward a standard Batched BLAS API

Matrices are stored BLAS like “the usual storage that we know”

- array of pointers: that consists of a pointer to each matrix
- strided: as one pointer to a memory and matrices are strided inside
  - Fixed stride
  - Variable stride
  - Suitable for CPU, GPU, Phi

- For variable stride, user has to fill-up the array
- Cannot accommodate data that was not been allocated within the same chunk of memory. Think about adding matrices to the batch.
Toward a standard Batched BLAS API

Matrices are stored BLAS like “the usual storage that we know”
- array of pointers: that consists of a pointer to each matrix
- strided: as one pointer to a memory and matrices are strided inside

Matrices are stored in interleaved fashion or compact
- data can be interleaved by batchcount or by chunk (SIMD, AVX, Warp)
- is only good for sizes less than 20 and only for some routines such as GEMM, TRSM, while it has performance and implementation issues for routines like LU or QR factorization
- Requires user or implementation to convert/reshuffle the memory storage since most of the storage are BLAS-like
Toward a standard Batched BLAS API

API discussion

- Same or separate API for fixed and variable size batches?
  - Have two separate API’s?
  - Have a flag that switch between fixed and variable?

- To simplify user life and avoid a combinatorial combination of parameter, we propose to **distinguish between fixed and variable size APIs**

```c
void batchedblas_dgemm_vbatched (  
    batched_trans_t transA , batched_trans_t transB ,  
    batched_int_t *m, batched_int_t *n, batched_int_t *k,  
    double alpha ,  
    double const * const *dA_array, batched_int_t *ldda ,  
    double const * const *dB_array, batched_int_t *lddb ,  
    double beta ,  
    double **dC_array, batched_int_t *lddc ,  
    batched_int_t batchCount , batched_queue_t queue );
```
Toward a standard Batched BLAS API

API discussion

- Same or separate API for fixed and variable size batches?
  - Have two separate API’s?
  - Have a flag that switch between fixed and variable?
  - To simplify user life and avoid a combinatorial combination of parameters, we propose to **distinguish between fixed and variable size APIs**

Group API

- Is not suitable for GPU
- Force the user to build groups before calling the routine then why not having different calls

```c
BBLAS_dgemm( TRANSA, TRANSB, M, N, K,
            ALPHA, A, LDA, B, LDB, BETA, C, LDC,
            GROUP_COUNT, SIZE_PER_GROUP, INFO )
```
Toward a standard Batched BLAS API

Error Handling

- Legacy Error Reporting Methods “xerbla”
  - Use of global state
  - Dependence on platform-specific features
  - Limited customization
  - LAPACK has additional output error parameter “info”
  - For batched BLAS, also a xerbla output may not indicate which matrix had the error
Toward a standard Batched BLAS API

Error Handling

- Legacy Error Reporting Methods “xerbla”
- Does batched BLAS need checking?
  - All errors reported
  - Some errors reported
  - No errors reported

Can be accomplished by the “info” array
**Toward a standard Batched BLAS API**

**Summary**

- Separate API for fixed and variable size batches
- Using standard storage “BLAS like”
- Use array of “info” for error reporting allowing for different level of reporting
- Other API’s could be considered as auxiliary API’s or API with extra features

```c
void batchedblas_dgemm_batched ( 
    batched_trans_t transA , batched_trans_t transB ,
    batched_int_t m, batched_int_t n, batched_int_t k,
    double alpha ,
    double const * const * dA_array , batched_int_t ldda ,
    double const * const * dB_array , batched_int_t lddb ,
    double beta ,
    double ** dC_array , batched_int_t lddc ,
    batched_int_t batchCount , batched_queue_t queue 
    batched_int_t *info );
```
Toward a standard Batched BLAS API

Summary

- Separate API for fixed and variable size batches
- Using standard storage “BLAS like”
- Use array of “info” for error reporting allowing for different level of reporting
- Other API’s could be considered as auxiliary API’s or API with extra features

Reference:

A proposed API for Batched Basic Linear Algebra Subprograms.

Timothy Costa Jack Dongarra Piotr Luszczek Mawussi Zounon
Extension to Batched Basic Linear Algebra Subprograms
Internal Technical Report University of Tennessee