Batched BLAS Moving Forward in MAGMA

Ahmad Abdelfattah, Azzam Haidar, Stan Tomov, and Jack Dongarra

Batched BLAS Mini-symposium
SIAM-CSE 2019

February 26th, 2019

(http://icl.utk.edu/bblas/siam-cse19/)
Outline

1. Status of MAGMA batched
2. Batched BLAS Performance
3. Batched LAPACK Performance
4. Applications using MAGMA Batched
5. C++ Interface for Batched BLAS
6. Conclusion
Outline

1 Status of MAGMA batched
2 Batched BLAS Performance
3 Batched LAPACK Performance
4 Applications using MAGMA Batched
5 C++ Interface for Batched BLAS
6 Conclusion
MAGMA Batched

- A wide coverage of important routines in BLAS and LAPACK
- Batched BLAS coverage
  - All level-3 BLAS routines \{GEMM, TRSM, TRMM, HEMM/SYMM, HERK/SYRK, HER2K/SYR2K\}
  - Level-2 \{GEMV and HEMV/SYMV\}
  - Both fixed and variable size supported
- Batched LAPACK coverage
  - LU and QR factorizations (fixed size)
  - Cholesky factorizations (fixed and variable size)
MAGMA Batched Interface

- Two separate C interfaces for fixed/variable size cases

```c
void magma_dgemm_batched(
    magma_trans_t transA, magma_trans_t transB,
    magma_int_t m, magma_int_t n, magma_int_t k,
    double alpha,
    double const * const * dA_array, magma_int_t ldda,
    double const * const * dB_array, magma_int_t lddb,
    double beta,
    double **dC_array, magma_int_t lddc,
    magma_int_t batchCount, magma_queue_t queue );
```

```c
void magmablas_dgemm_vbatched(
    magma_trans_t transA, magma_trans_t transB,
    magma_int_t* m, magma_int_t* n, magma_int_t* k,
    double alpha,
    double const * const * dA_array, magma_int_t* ldda,
    double const * const * dB_array, magma_int_t* lddb,
    double beta,
    double **dC_array, magma_int_t* lddc,
    magma_int_t batchCount, magma_queue_t queue );
```

- A stride-based interface for fixed size batched GEMM is recently added
Outline

1. Status of MAGMA batched
2. Batched BLAS Performance
3. Batched LAPACK Performance
4. Applications using MAGMA Batched
5. C++ Interface for Batched BLAS
6. Conclusion
Performance of Batched GEMM

- cuBLAS is optimized for relatively large sizes
- MAGMA outperforms cuBLAS for small non-square sizes
  - Very helpful in small rank updates and recursive panel factorizations
Performance of Batched GEMM “cont.”

- Special optimizations for small square matrices

![Graph showing performance of Batch DGEMM, batch=1000, Tesla V100 GPU, CUDA-10.0]
Performance of Batched Triangular Solve (TRSM)

- MAGMA outperforms cuBLAS for the most part
- A use case from batched LU factorization (if panel width = 32)
Half Precision Batched GEMM

- Using Tensor Cores (Abdelfattah et al., IPDPS'19)
- Significant speedups on small sizes

![](graph.png)

Batch HGEMM, batch=10k, Tesla V100 GPU, CUDA-10.0

- magma
- cublas
Outline

1. Status of MAGMA batched
2. Batched BLAS Performance
3. Batched LAPACK Performance
4. Applications using MAGMA Batched
5. C++ Interface for Batched BLAS
6. Conclusion
Designing Batched LAPACK on GPUs

- Currently focusing on matrix factorization (LU/QR/Cholesky)
  - Next, batch SVD and eigenvalue problems
- Two solid foundations are required
  - A well-tuned batched BLAS
  - An optimized batched panel factorization
- A multi-level blocking approach
  - Matrix blocking
  - Panel blocking
  - Column blocking
- On-the-fly switching among blocking strategies
- Case study: batched LU factorization
Matrix blocking

- Tiny matrices fit entirely in registers/shared memory
- Optimal memory traffic
- Fused factorization (unblocked)

![Batch LU factorization, batch=50k, Tesla V100 GPU, CUDA-10.0](chart.png)
Panel blocking

- Fused panel factorization (with recursion)
- Rely on tuned BLAS for performance (batch GEMM)

Larger matrices use unfused panel factorization, but the update phase become even more dominant
Outline

1. Status of MAGMA batched
2. Batched BLAS Performance
3. Batched LAPACK Performance
4. Applications using MAGMA Batched
5. C++ Interface for Batched BLAS
6. Conclusion
Applications using MAGMA-Batched

- Climate simulation (UTK)
  - Fixed size batch LU factorization (no pivoting)
- DMRG++ (ORNL and UTK)
  - Variable size batch GEMM
- Hierarchical BiCGStab solver (UTK, Univ. Tokyo, Kyushu Univ., and Tokyo Tech.)
  - Variable size batch GEMV
  - (Yamazaki et al., IPDPS’18)
- Tensor Contractions (LLNL, ANL, UTK, and others)
  - Fixed size batch GEMM (with fusion)
  - CEED (https://github.com/CEED/libCEED)
Outline

1. Status of MAGMA batched
2. Batched BLAS Performance
3. Batched LAPACK Performance
4. Applications using MAGMA Batched
5. C++ Interface for Batched BLAS
6. Conclusion
Why Batch BLAS++?

- A backend for SLATE (https://icl.utk.edu/slate/)
- E.g., batched GEMM for rank updates
- C interfaces of batched BLAS are divergent
- Arguments lack flexibility!

<table>
<thead>
<tr>
<th>Type</th>
<th>Library name</th>
<th>Argument type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Options</td>
</tr>
<tr>
<td>Flag-based flat</td>
<td>Standard</td>
<td>BATCH_FIXED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BATCH_VARIABLE</td>
</tr>
<tr>
<td>Group</td>
<td>MKL</td>
<td>Per group</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Across groups</td>
</tr>
<tr>
<td>Flat</td>
<td>MAGMA</td>
<td>Fixed size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variable size</td>
</tr>
<tr>
<td></td>
<td>cuBLAS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hipBLAS/rocBLAS</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: A summary of existing interfaces for batch BLAS. An entry with “F” means that the corresponding category of arguments is fixed across the batch. An entry with “V” means that this category of arguments is allowed to be varied across different problems in the batch.

\(^7\)A stride-based interface is also available.
Example of User’s Inconvenience

Consider batched GEMM: \[ C_{M \times N} = \alpha A_{M \times K} \times B_{K \times N} + \beta C_{M \times N} \]

What if \( M \) and \( N \) vary, but \( K \) is fixed?

- Supported only by variable size APIs
- The value of \( K \), must be duplicated in an array

What if \( A \) (or \( B \)) is unified across a fixed size batch?

- The pointer has to be duplicated in an array

Ideally, each argument should be independently fixed or variable
Batch BLAS++ provides a very flexible API

- Brute-force overloading is a cumbersome
- We propose to use the C++ `std::vector` container
  - Each vector has two possible sizes (1 or `batch`)
- Covers all possible scenarios

```cpp
define blas::batch::gemm(
    blas::Layout layout,
    std::vector<blas::Op> const &transA,
    std::vector<blas::Op> const &transB,
    std::vector<int64_t> const &m,
    std::vector<int64_t> const &n,
    std::vector<int64_t> const &k,
    std::vector<double> const &alpha,
    std::vector<double*> const &Aarray, std::vector<int64_t> const &ldda,
    std::vector<double*> const &Barray, std::vector<int64_t> const &lddb,
    std::vector<double> const &beta,
    std::vector<double*> const &Carray, std::vector<int64_t> const &lddc,
    const size_t batch,
    std::vector<int64_t> &info );
```
Batch BLAS++ Rules of Thumb

1. All the semantics are hidden within the vector sizes
   - \( \text{size} = 1 \Rightarrow \) the argument is unified "fixed" across the batch
   - \( \text{size} = \text{batch} \Rightarrow \) the argument has a distinct entry for each operation

2. Batch BLAS++ calls the optimized vendor batch routine if its signature matches the vector sizes

3. Otherwise, OpenMP or CUDA streams are used
Error Checking in Batch BLAS++

- The size of the vector `info` decides the error checking mode
  - `info.size() = 0` means no error checking
  - `info.size() = 1` means an argument-based error checking
  - `info.size() = batch` means a problem-based error checking

- An extra error checking layer: vector size integrity
  - E.g., `A.size() = 1`, but `lda.size() = batch`

- No checking is recommended for production runs.
Status of the Batch BLAS++

- A detailed design document (SWAN #4)
  - [http://www.icl.utk.edu/publications/swan-004](http://www.icl.utk.edu/publications/swan-004)
- A complete reference implementation of level-3 BLAS
  - [https://bitbucket.org/icl/blaspp](https://bitbucket.org/icl/blaspp)
  - For CPUs and GPUs (with cuBLAS backend)
- Feedback welcome
- Integration into SLATE
- Adoption into MAGMA
Outline

1 Status of MAGMA batched
2 Batched BLAS Performance
3 Batched LAPACKPerformance
4 Applications using MAGMA Batched
5 C++ Interface for Batched BLAS
6 Conclusion
To summarize

- There is a lot of active development in MAGMA-Batched
  - Performance improvements
  - New functionalities
  - Half precision routines

- Batch BLAS++ provides a unified interface that serves all purposes
Interested in joining ICL?

ICL is hiring

OPEN POSITION
Research Position in Numerical Linear Algebra

The primary duties of this full-time position are to assist in the design, development, and maintenance of numerical software libraries for solving linear algebra problems on large distributed memory machines with multicore processors and hardware accelerators; write research papers documenting research findings; present material at conferences; and direct students and a research team in their research endeavors related to ongoing and future projects. This role will have opportunities for publication, travel, and high-profile professional networking in academia, labs, and industry.

https://www.icl.utk.edu/jobs
Thank You!