

MATEDOR: MAtrix, TEnsor, and Deep-learning Optimized Routines

Ahmad Abdelfattah, Jack Dongarra, Stanimire Tomov, Ichitaro Yamazaki
University of Tennessee Knoxville, USA
{ahmad,dongarra,tomov,iyamazaki}@icl.utk.edu

Azzam Haidar
NVIDIA Corporation
azzamhaidar@nvidia.com

ABSTRACT

The MAtrix, TEnsor, and Deep-learning Optimized Routines (MATEDOR) project develops software technologies and standard APIs, along with a sustainable and portable library, for large-scale computations that can be broken down into very small matrix or tensor computations. The main target of MATEDOR is to accelerate applications from important fields that fit this profile, including deep learning, data mining, astrophysics, image and signal processing, hydrodynamics, and more.

KEYWORDS

Linear algebra, batch computation, mathematical software

ACM Reference Format:

Ahmad Abdelfattah, Jack Dongarra, Stanimire Tomov, Ichitaro Yamazaki and Azzam Haidar. 2018. MATEDOR: MAtrix, TEnsor, and Deep-learning Optimized Routines. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'19)*, Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.475/123_4

1 OVERVIEW

MATEDOR is an numerical library that provides optimized batch linear algebra routines on different hardware architectures. The goal of MATEDOR is to achieve performance portability for many applications that require very small matrix and tensor computations. Such applications include machine learning, astrophysics, image processing, and many others.

In addition to achieving high performance on different platforms, MATEDOR focuses on the design of the routine interface. Such an interface serves as a standard candidate for wide adoption by vendors and library developers.

2 INTERFACE OF THE BATCH LINEAR ALGEBRA ROUTINE

MATEDOR provides C interfaces for batched BLAS and LAPACK operations that are very similar to NVIDIA's cuBLAS library. While the latter provides interfaces for only fixed size problems, MATEDOR provides a separate C interface for variable size batches as well.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SC'19, November 2018, Dallas, Texas USA

© 2018 Copyright held by the owner/author(s).

ACM ISBN 123-4567-24-567/08/06.

https://doi.org/10.475/123_4

As an example, the interface for batched matrix-matrix multiplication (GEMM) looks like:

```
1 void dgemm_batched(  
2     trans_t transA, trans_t transB,  
3     int m, int n, int k,  
4     double alpha, double const * const * dA_array, int ldda,  
5     double const * const * dB_array, int lddb,  
6     double beta, double ** dC_array, int ldc,  
7     int batch, queue_t queue );
```

On the other hand, the variable size API assumes that each matrix has its own size and leading dimension, thus promoting sizes and leading dimensions from scalar arguments to array arguments.

```
1 void dgemm_vbatched(  
2     trans_t transA, trans_t transB,  
3     int* m, int* n, int* k,  
4     double alpha, double const * const * dA_array, int* ldda,  
5     double const * const * dB_array, int* lddb,  
6     double beta, double ** dC_array, int* ldc,  
7     int batch, queue_t queue );
```

Another direction in this regard is to consider C++ for API standardization. The notions of templates, namespaces, and overloading enable a more flexible API that can cover many divergent C APIs. As an example, the use of the C++ `std::vector` container has proven to provide maximum flexibility for the interface [1].

```
1 namespace blas{  
2 namespace batch{  
3 void gemm(  
4     vector<blas::Op> const &transA, vector<blas::Op> const &transB,  
5     vector<int64_t> const &m,  
6     vector<int64_t> const &n,  
7     vector<int64_t> const &k,  
8     vector<double> const &alpha,  
9     vector<double*> const &Aarray, vector<int64_t> const &ldda,  
10    vector<double*> const &Barray, vector<int64_t> const &lddb,  
11    vector<double> const &beta,  
12    vector<double*> const &Carray, vector<int64_t> const &lddc,  
13    const size_t batch, vector<int64_t> &info );  
14 } // namespace batch  
15 } // namespace blas
```

3 IMPACT ON DIFFERENT APPLICATIONS

So far, MATEDOR has been featured in many applications that use batch linear algebra routines, especially on GPU accelerators.

- (1) **Tensor contractions in high-order FEM**, where a sequence of multiplications on very small matrices is carried out in a fused GPU kernel [2, 4]. By providing device-level matrix multiplication APIs that are callable from a GPU kernel, MATEDOR can increase the arithmetic intensity of an application by fusing many GEMM operations together into one execution context, thus maximizing data reuse. The device-level APIs are designed to work using the same thread configuration, so that they can be used seamlessly into one kernel.

- (2) **Dense/sparse linear solvers and preconditioners**, where batch on sided factorization is of great importance [3, 5]. In general, sparse direct solvers require factorization of many small matrices in parallel. MATEDOR provides the foundation of such solvers through high performance batch LU, QR, and Cholesky factorizations. When the sizes are smaller than a warp size (i.e. 32), MATEDOR uses a single computational kernel that performs the entire factorization, while minimizing memory traffic. Bigger sizes are factorized following a sequence of batch panel factorizations followed by batch updates.
- (3) **The Density Matrix Renormalization Group (DMRG)** algorithm has been accelerated using a variable size batch GEMM routine. The application uses a sparse grid DG solver that can solve high-dimensional problems of fusion. The solver, at its core, uses a sparse kronecker product that be formalized using the variable size batch GEMM routine in MATEDOR.
- (4) **A hierarchical BiCGStab solver** has been shown to benefit from a variable size batch GEMV routine [7]. The hierarchical solver is used to accelerate ppohBEM, an open-source software package that implements the boundary element method. ppohBEM relies on another software package called HACApK, which reduces the cost of solving the linear system by hierarchically compressing the coefficient matrix using adaptive cross approximation. HACApK's linear solver have been ported onto GPU clusters. Since the linear solver core operation is a matrix-vector multiply of a hierarchically compressed matrix, MATEDOR's variable size batch GEMV routine is at the core of the GPU-accelerated version of HACApK.
- (5) **Deep neural networks** heavily rely on optimized batch GEMM routines, for which MATEDOR outperforms cuBLAS on very small sizes [6].

4 COMMUNITY ENGAGEMENT

A broad group of the research community is meeting on a regular basis in order to propose a standard interface of the batched BLAS, touch base with different vendors, and highlight the needs of several applications of interest. More details can be found on <http://icl.utk.edu/bblas/>.

5 ACKNOWLEDGEMENTS

This work is partially supported by NSF Grant No. OAC 1740250 and CSR 1514286, NVIDIA, and the Department of Energy under the Exascale Computing Project (17-SC-20-SC and LLNL subcontract under DOE contract DE-AC52-07NA27344).

REFERENCES

- [1] Ahmad Abdelfattah, Konstantin Arturov, Cris Cecka, Jack Dongarra, Chip Freitag, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Panrui Wu. 2017. *C++ API for Batch BLAS*. Technical Report 4, ICL-UT-17-12.
- [2] Ahmad Abdelfattah, Marc Baboulin, Veselin Dobrev, Jack J. Dongarra, Christopher W. Earl, Joel Falcou, Azzam Haidar, Ian Karlin, Tzanio V. Kolev, Ian Masliah, and Stanimire Tomov. 2016. High-Performance Tensor Contractions for GPUs. In *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*. 108–118. <https://doi.org/10.1016/j.procs.2016.05.302>
- [3] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. 2018. Batched one-sided factorizations of tiny matrices using GPUs: Challenges and countermeasures. *Journal of Computational Science* 26 (2018), 226 – 236. <https://doi.org/10.1016/j.jocs.2018.01.005>
- [4] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. 2018. *Tensor Contractions using Optimized Batch GEMM Routines*. Technical Report. http://icl.cs.utk.edu/projectsfiles/magma/pubs/72-gtc18_tensor_poster.pdf NVIDIA GTC 2018 poster.
- [5] Azzam Haidar, Ahmad Abdelfattah, Mawussi Zounon, Stanimire Tomov, and Jack Dongarra. 2018. A Guide for Achieving High Performance with Very Small Matrices on GPU: A Case Study of Batched LU and Cholesky Factorizations. *IEEE Transactions on Parallel and Distributed Systems* 29, 5 (May 2018), 973–984. <https://doi.org/10.1109/TPDS.2017.2783929>
- [6] Lucien Ng, Kwai Wong, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. 2017. *MagmaDNN High-Performance Data Analytics for Manycore GPUs and CPUs*. Technical Report. <http://icl.cs.utk.edu/projectsfiles/magma/pubs/71-MagmaDNN.pdf> 2017 Summer Research Experiences for Undergraduate (REU).
- [7] Ichitaro Yamazaki, Ahmad Abdelfattah, Akihiro Ida, Satoshi Ohshima, Stanimire Tomov, Rio Yokota, and Jack J. Dongarra. 2018. Performance of Hierarchical-matrix BiCGStab Solver on GPU Clusters. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*. 930–939. <https://doi.org/10.1109/IPDPS.2018.00102>