CSIT Conference 2019, Yerevan, Armenia, September 23-27

Strided Batched Matrices Multiplication Performance in the Architecture of GPU Accelerator

Edita Gichunts

Institute for Informatics and Automation Problems of NAS RA
e-mail: editagich@ipia.sci.am

ABSTRACT

Many high-level solutions and scientific applications require a high-performance batched linear algebra package. Batched subprograms are a part of software that simultaneously use the same action on multiple issues independent of each other. Batched matrix-matrix multiplication (GEMM) is used in many scientific tasks, and it is very important that we achieve high performance when multiplying numerous small matrices. In this work, the Strided Batched matrices multiplication implementation in the hybrid system is performed on the NVIDIA Tesla K40c graphic processor using the MAGMA 2.5.0 library.

Keywords

GPU accelerator; Strided Batched matrices multiplication; MAGMA library; hybrid architectures; C++ API; smaller matrices; performance; subprograms.

1. INTRODUCTION

Many high-priority applications require solutions that deal with smaller matrices. They contain quite a lot of calculations that consist of a large number of smaller matrices.

The emergence of heterogeneous systems with graphic processors revealed an almost complete lack of linear algebra software for batched operations. In these systems, the MAGMA package has been created to implement the problems of linear algebra.

The MAGMA program is aimed at creating linear algebra libraries of a new generation that provide the fastest and most accurate solution in a heterogeneous architecture from the modern multi-core + multi-GPU system. MAGMA's research is based on the idea that software solutions should themselves be hybridized by combining different algorithms in one structure to solve complex problems, such as heterogeneity, mass paralleling, computation speed, and CPU-GPU interaction speed. Based on this idea, the purpose for hybrid multi-core and several graphics processors is to develop algorithms and a linear algebra.

The MAGMA library is an open-source package that uses the computational power of the graphics processor for multiple BLAS and LAPACK algorithms.

MAGMA batched subprograms are intended only to perform graphic processors and solve the problems with numerous small matrices. Batched subprograms have been applied to the following targeted problems. For example, the batched LU factorization was used in underground transport modeling [1]. Batched Cholesky Factorization and Triangular Solutions are used to accelerate the smallest square solutions [2, 3]. Batched matrix-matrix multiplication (GEMM) underlies the many tensor reduction problems [4, 5]. The Block-Jacobi generation was accelerated by the batched matrix inversion [6].

The MAGMA 2.5.0 version, released in January 2019, contains the new integrated Strided Batched GEMM subprogram, which is another strategy of the Batched GEMM subprogram and can be considered as a more optimal option. Optimization reduces the software code, gains time and increases performance.

In this paper, we present the implementation of the Strided Batched matrix multiplication in the CPU-GPU hybrid system and the performance towards the Batched matrix-matrix multiplication.

2. BATCH BLAS SUBPROGRAM PACKAGE

The first release of the Batch BLAS [7] standard specifies the BLAS 3 level name, type description, and C-interface agreement. The interfaces are specifically designed to be close to BLAS and to be hardware-independent. They are given in C to be used in C / C ++ programs, but extensions and development may be called from other languages, such as Fortran. The goal is to allow the developers to express programs, compilers, execution systems, and many smaller actions in the form of one call of BLAS subprogram.

2.1. Naming Conventions

The Batched BLAS subprogram name follows the BLAS appropriate subprogram name and is expanded with _batched entry, for example, dgemm will become dgemm_batched. And in the case of the strided version of the matrix multiplication it will be dgemm_batched_strided.

Arguments that describe transpositions, conjugation parameters, dimensionality, and leading dimensions are transformed into arrays, rather than into scalars. The matrix input/output is transmitted as an array of pointers.

2.2. Error Handling

The batched BLAS subprograms in MAGMA have a bug fixing mechanism, which is the analog of the BLAS standard. Digital errors are not reported, only the errors found in the arguments are passed to the user through the XERBLA () function. Arguments are stored in the graphic processor memory in the form of arrays, so the error checks are made on the GPU, and in the case of an error detection, the processor is notified and the corresponding XERBLA () is called.

2.3. Stride APIs

For Batch BLAS, the offer of C ++ API is presented in the context of the BLAS and LAPACK [8] C ++ API, which is the basis of the SLATE [9] library. The proposed API shares the following design solutions from C ++ BLAS API:

1. Stateless interface. The proposed batch BLAS API will be stateless.
2. Templated routines.
3. C++ language standard. The C++11 standard sufficiently covers all of the features required in the proposed APIs.

4. Naming convention. BLAS and batch BLAS interfaces are reachable by including the blas.hh header and using the namespace blas.
5. Enumerated constants. The proposed APIs use the same *enum* constants defined in the blas namespace.
6. Matrix layout. For now, the proposed API shall focus only on column-major layouts. While the C++ BLAS API supports row-major layouts by calling column-major routines using.
7. Integer type. The proposed API will use the int64_t type to specify sizes and dimensions. Although batch routines usually operate on relatively small sizes, the use of int64_t unifies the object dimensions between BLAS and batch BLAS and avoids any confusion about the size of integer type.

The API uses the standard vector std :: vector container of C++, which provides a very flexible interface. In fact, almost all the BLAS subprogram arguments will be converted into the same type of std: vector argument.

Both MAGMA and cuBLAS allow the user to indirectly transmit the array of pointers through the pointer and the fixed step. If the matrices are at an equal distance from each other, it is easier than simply presenting the array of pointers. There are two solutions for Stride interface support. First of all, it is necessary to provide an auxiliary function std::vector <FloatType *> to complete, based on the pointer-step. Otherwise, instead of std::vector <FloatType *>, the proposal of C ++ API can be overloaded for the pointer-step to be accepted. However, it is unclear whether the output / input data pointers can be available. This adds another level of interface complexity.

3. STRIDED BATCHED GEMM IMPLEMENTATION IN THE HYBRID SYSTEM

Introduce the descriptions of Batched GEMM and Strided Batched GEMM subprograms in the Magmablas package.

Batched GEMM

Batched matrix-matrix multiplication performs the following calculation:

```
for ( int p = 0; p <batchCount; ++p ) {
  for ( int m = 0; m < M; ++m ) {
    for ( int n =0; n < N; ++n) {
      c_mnp = 0;
      for ( int k =0; k < K; ++k )
       c_mnp += A[ p ] [ m + k * lda ] * B[ p ] [ k + n * lda ];
      C[ p ][ m + n *ldc ] = ( *alpha ) * c_mnp +
            +( *beta ) * C[ p] [ m + n * ldc];
    }
  }
}
```

Where A[p], B[p], and C[p] are common matrices.

The gemm_batched interface in Magmablasis as follows:

magmablas_sgemm_batched(magma_trans_t transA, magma_trans_t transB, magma_int_t m, magma_int_t n, magma_int_t k, float alpha, float const * const * dA_array, magma_int_t ldda, float const* const* dB_array, magma_int_t lddb, float beta, float **dC_array, magma_int_t lddc, magma_int_t batchCount, magma_queue_t queue).

This is for real matrices with one by one precision.

TransA and transBcan accept three possible values:

Trans=Magma_NoTrans,

Trans=Magma_Trans,

Trans=Magma_ ConjTrans.

BatchCountis an integer that indicates the number of matrices being processed.

Queue–the number of consecutive execution.

In the pointer-to-pointer interface above, the user must provide a pointer to an array of pointers to matrix data, which requires the creation and calculation of these structured data that provides input for code, memory, and time. Performance can be reduced when the pointers cannot be pre-calculated and repeatedly used. Moreover, the matrix pointers should exist on the GPU and show the GPU memory. This means:

1) GPU memory allocation,

2) Moving the array of pointers to GPU,

3) GPU memory writes,

4) GPU memory exemption.

Fortunately, we had a new powerful solution in MAGMA 2.5.0 - the Strided Batched GEMM subprogram, in which the transition from matrix to matrix is performed with a firm step.

Strided Batched GEMM

The transition between the matrices in this subprogram is made with a firm step enabling to avoid the above-mentioned superfluous steps.

The Strided Batched matrix-matrix multiplication performs the following calculation:

```
for (int p = 0; p <batchCount; ++p) {
  for (int m = 0; m < M; ++m) {
    for (int n = 0; n < N; ++n) {
      c_mnp = 0;
      for (int k = 0; k < K, ++k)
       c_mnp += A[m + k*ldA + p*strideA] * B[k + n*ldB+
                + p*strideB];
      C[m + n*ldC + p*strideC] = (*alpha)*c_mnp +
            + (*beta)*C[m + n*ldC + p*strideC];
  }
 }
}
```

In Magmablas, gemm_batched_strided interface is as follows:

magmablas_sgemm_batched_strided(magma_trans_t transA, magma_trans_t transB, magma_int_t m, magma_int_t n, magma_int_t k, float alpha, float const * dA, magma_int_t ldda, magma_int_t strideA, float const * dB, magma_int_t lddb, magma_int_t strideB, float beta, float* dC, magma_int_t lddc, magma_int_t strideC, magma_int_t batchCount, magma_queue_t queue),

where a data pointer argument is passed as a combination (pointer+stride) rather than explicitly as a pointer array.

The implementation of this subprogram in CPU-GPU hybrid system is carried out by the following steps:

1. Any magma program begins with the magma_init () initialized function.

2. For matrices A, B, and C on the CPU, memory is allocated by the function magma_cmalloc_cpu (), and for the matrix A -magma_cmalloc_cpu (& A, lda * n * batchCount).

3. For matrices A, B, and C on the GPU, memory is allocated by the magma_cmalloc () function, and for the matrix A - magma_cmalloc (&d_A, ldda * n * batchCount).

4. In the CPU memory, using the lapackf77_xlarnv () function of the LAPACK library, the A, B and C matrices are initialized, and for the matrix A - lapackf77_xlarnv (&ione, ISEED, &sizeA, a).

5. The A, B and C matrices are moved from the CPU memory to the GPU memory via the magma_xsetmatrix () function. And for the matrix A - magma_xcsetmatrix (n, n * batchCount, a, lda, d_a, ldda).

6. We call the magmablas_xgemm_batched_strided () matrix-matrix product function for common matrices, indicating the required values of arguments. For example, the input common matrices and their dimensions, the firm stride step between the matrices and the most important value batchCount, which shows how many matrices we have to process.
7. After the function is completed, we record the execution time and then count the performance of the function execution.
8. As a result, the C matrix moves from the GPU memory to the CPU memory: magma_xgetmatrix (n, n * batchCount, d_c, lddc, h_c, ldc).
9. After completing any program in the hybrid system, the CPU and GPU memories will be released. It is performed using the magma_free_cpu () and the magma_free () functions, respectively.
10. Any magma program ends with the magma_finalize () finalizing function.

4. RESULTS OF EXPERIMENTS

The experiments were conducted on NVIDIA K40c GPU. The architecture of Tesla K40c consists of 2880 CUDA processor cores. It is endowed with much higher bandwidth 288 GB/s of message transfer between CPU and GPU, having 12 GB of global memory per card running at 745 MHz., GDDR5 memory interface, and CUDA C programming environment.
The operation system of Tesla K40c is Ubuntu 14.04.2 LTS. Cuda7 programming environment was used for the realization of programs.
MAGMA 2.5.0 package was installed in accordance with cuda7 environment. For the compilation of MAGMA 2.5.0 library the lapack-3.4.2, clapack-3.2.1 and atlas-3.10.0 packages were installed. Gcc-4.8, gfortran-4.8, g ++ - 4.8 and nvcc compilers were used. Such references were made in make.inc file on libf77blas.a, libcblas.a, libf2c.a, libcublas.so, libcudart.so, libm.a, libstdc++.so, libpthread.so, libdl.so, libcusparse.so, libcudadevrt.a static and dynamic libraries. MAGMA 2.5.0 package contains libmagma.a, libmagmablas.a, libmagma_sparse.a and libblas_fix.a libraries.
Experiments were carried out in 5000 matrices with dimensions from 8 * 8 to 256 * 256.
Figures 1 and 2 illustrate the performance graphs of the batched and batched_strided common matrices multiplication subprograms for real numbers with single and double precisions, respectively.

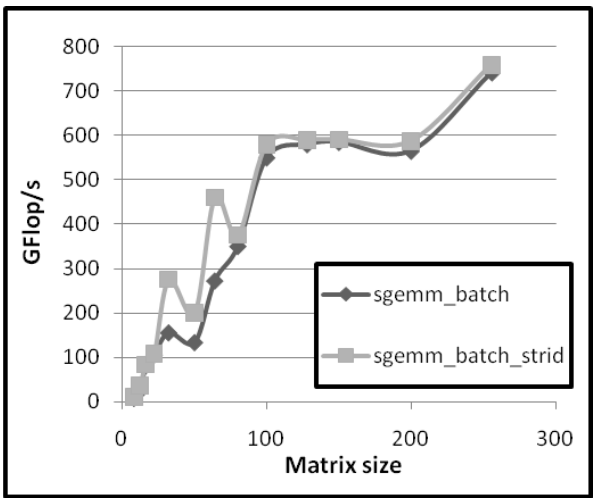

Fig.1. Real Single Precision

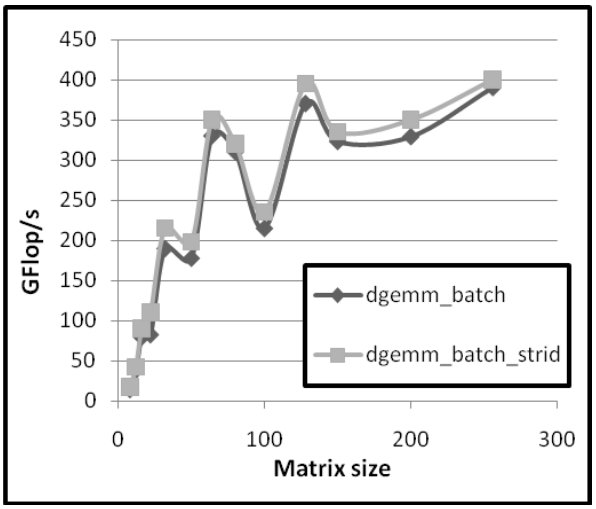

Fig.2. Real Double Precision

Experiments show that in case of single precision for real matrices, the batched_strided subprogram exceeds the batched subprogram by 20-30% in the case of matrices from 16 * 16 to 64 * 64 dimensions. In other cases, it exceeds at least by 10%. The maximum performance for this case is 760 GFlops / s.
In case of double precision for real matrices, the batched_strided subprogram exceeds the batched subprogram by 10-15 % and achieves the maximum performance at 400 GFlops / s.
Figures 3 and 4 depict the performance graphs of the batched and batched_strided common matrices multiplication subprograms for complex numbers with single and double precisions, respectively.

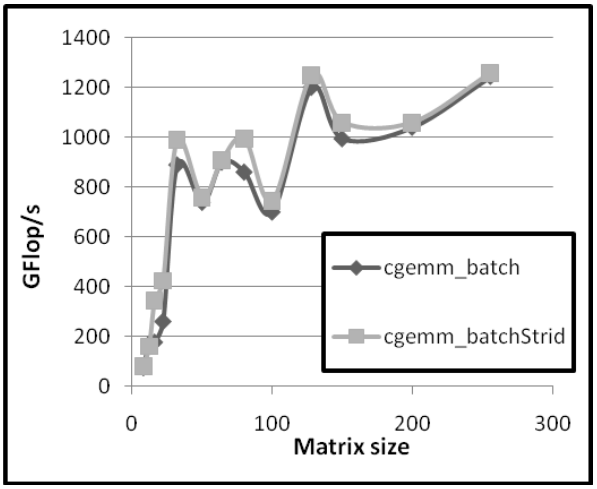

Fig.3. Complex Single Precision

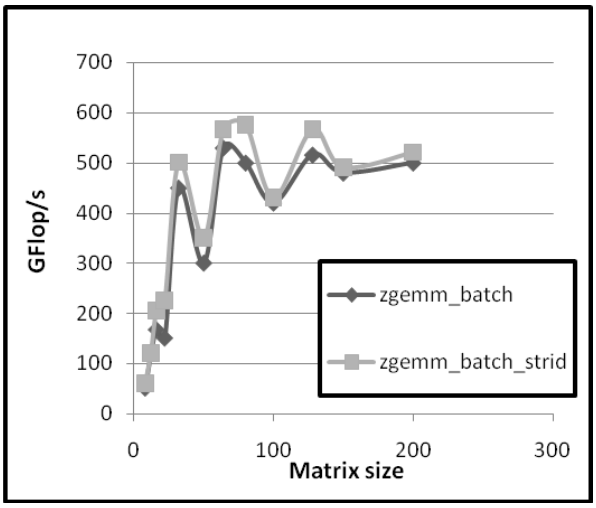

Fig.4. Complex Double Precision

In cases of single and double precisions for complex matrices, the batched_strided subprogram exceeds the batched subprogram by 10-20 %. With single precision, the maximum performance reaches 1260 GFlops / s, and with the double one - 575 GFlops / s.

Note that time is included in magmablas_xgemm_batched of two MAGMA gemm strategies, which is required for allocating, calculating and transmitting of pointer-to-pointer structural data affecting the performance.

The implementation of the magmablas_xgemm_batched_strided subprogram does not require allocation and release of CPU and GPU memories for arrays of pointers. They are especially useful for calculations on GPU, where redistribution and transfer can be relatively expensive and cause unwanted synchronization.

5. CONCLUSION

We presented the implementation of the gemm_batched_strided subprogram of 5000 multiplication matrices with dimensions from 8 * 8 to 256 * 256 on the K40c graphics processor in the CPU-GPU hybrid system using the MAGMA 2.5.0 library. Based on the results of the experiments, we came to the following conclusion:

• In case of multiplication of numerous very small matrices, the use of the gemm_batched_strided subprogram will result in higher performance than the gemm_batched subprogram.

• We have an incomparable reduction in program code, which leads to time saving, less memory-consuming of CPU and GPU.

REFERENCES

[1] Oreste Villa, Massimiliano Fatica, Nitin Gawande, and Antonino Tumeo. Power/Performance Trade-Os of Small Batched LU Based Solvers on GPUs, pages 813–825. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-40047-6. doi: 10.1007/978-3-642-40047-681.
URL: *https://doi.org/10.1007/978-3-642-40047-681*.

[2] Mark Gates, Hartwig Anzt, Jakub Kurzak, and Jack J. Dongarra. Accelerating collaborative ltering using concepts from high performance computing. In 2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015, pages 667–676, 2015. doi: 10.1109/BigData.2015.7363811.
URL: https://doi.org/10.1109/BigData.2015.7363811.

[3] Jakub Kurzak, Hartwig Anzt, Mark Gates, and Jack J. Dongarra. Implementation and Tuning of Batched Cholesky Factorization and Solve for NVIDIA GPUs. IEEE Trans. Parallel Distrib. Syst., 27(7):2036–2048, 2016. doi: 10.1109/TPDS.2015.2481890.
URL: *https: //doi.org/10.1109/TPDS.2015.2481890*.

[4] 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. High-Performance Tensor Contractions for GPUs. In International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, pages 108–118, 2016. doi: 10.1016/j.procs.2016.05.302.
URL: https://doi.org/10.1016/j.procs.2016.05.302.

[5] Yang Shi, U. N. Niranjan, Animashree Anandkumar, and Cris Cecka. Tensor Contractions with Extended BLAS Kernels on CPU and GPU. In 23rd IEEE International Conference on High Performance Computing, HiPC 2016, Hyderabad, India, December 19-22, 2016, pages 193–202, 2016. doi: 10.1109/HiPC.2016.031.
URL: https://doi.org/10.1109/HiPC.2016.031.

[6] Hartwig Anzt, Jack J. Dongarra, Goran Flegar, and Enrique S. Quintana-Orti. Batched Gauss-Jordan Elimination for Block-Jacobi Preconditioner Generation on GPUs. In Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM@PPoPP 2017, Austin, TX, USA, February 5, 2017, pages 1–10, 2017. doi: 10.1145/3026937.3026940.
URL: *http://doi.acm.org/10.1145/3026937.3026940*.

[7] Jack Dongarra, Iain Du, Mark Gates, Azzam Haidar, Sven Hammarling, Nicholas J. Higham, Jonathon Hogg, Pedro Valero-Lara, Samuel D. Relton, Stanimire Tomov, and Mawussi Zounon. A Proposed API for Batched Basic Linear Algebra Subprograms. Technical report, Manchester Institute for Mathematical Sciences, April 2016.
URL: http://eprints.maths.manchester.ac.uk/id/eprint/2464.[MIMS Preprint].

[8] Mark Gates, Piotr Luszczek, Jakub Kurzak, Jack Dongarra, Konstantin Arturov, Cris Cecka, and Chip Freitag. C++ API for BLAS and LAPACK. Technical Report 2, ICL-UT-17-03, 06-2017 2017. Revision 06-2017.

[9] Jakub Kurzak, PanruoWu, Mark Gates, Ichitaro Yamazaki, Piotr Luszczek, Gerald Ragghianti, and Jack Dongarra. Designing SLATE: Software for Linear Algebra Targeting Exascale. SLATE Working Notes 3, ICL-UT-17-06, 10-2017 2017.