Batched BLAS Problems of Complex Hermitian Small Matrices in the Architecture of GPU Accelerator

Edita Gichunts

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

ABSTRACT

The emergence of multi-core and heterogeneous architectures requires the processing of a number of linear algebra algorithms to take advantage of accelerators, such as graphics processors. There is a difficult class of problems involving linear algebra operations for thousands of small-dimensional matrices. Batched computations continue to have a wide range of applications in scientific calculations. Their goal is to more efficiently transfer the application of algorithms to high-performance multi-core architectures. Some operations of linear algebra are presented in this work for complex Hermitian small matrices, which are grouped together under the name of Batched BLAS. In this work the performances of the complex Hermitian Batched matrix-matrix multiplication, matrix-vector multiplication and 2nd rank update operations are presented on NVIDIA Tesla K40c graphics processor with the use of MAGMA library.

Keywords

MAGMA library, BLAS operations, Batched computation, Hermitian small matrix.

1. INTRODUCTION

The origin of Basic Linear Algebra Subprograms (BLAS) standard was observed from 1973, when Hanson, Krogh, and Lawson published an article that described the advantages of adoption of basic collection procedures for linear algebra. It led to the creation of a linear algebra package BLAS [1], which became quite successful and competitive. It was adopted as a standard and first used in LINPACK [2] package. Initially BLAS consisted of 2 levels [3], when the 3rd level was suggested[4], which included the most important actions associated with matrices, LINPACK was reprocessed to LAPACK [5] in order to use the newly created 3rd level of BLAS.The MAGMA package is a linear algebra library as LAPACK but it is intended for heterogeneous Multicore+GPU architectures [6]. MAGMA aims to develop algorithms and structures of linear algebra in graphics processors and hybrid multicore systems.
CUBLAS is a CUDA library on GPU produced by NVIDIA. LAPACK is a Fortran library on CPU. MAGMA calls some subprograms of CUBLAS and LAPACK, but involves more complex procedures. Upon the emergence of the heterogeneous system with accelerators, a software shortage of especially many small matrix operations of linear algebra was observed. Many modern works use solutions for many small matrix operations. The Batched subprograms have recently been integrated into MAGMA, which solve many small problems and are called MAGMA Batched subprograms. These subprograms are based on Batched BLAS package. NVIDIA already presents the optimized Batched BLAS applications in CUBLAS [7] package.
The aim of the Batched subprograms is the parallel solution of set of problems independent of each other. If one matrix is large enough, then the Batched subprograms are not required to transfer it to GPU, but when the matrices are small (for example,the sizes of which are less than or equal to 512) the amount of work needed to perform the factorization cannot saturate the device, therefore there is a need for Batched subprograms.
This work presents the matrix-matrix multiplication, matrix-vector multiplication Batched BLAS operations of small Hermitian matrices, as well as the realizations of 2nd rank update functions on Tesla K40c graphics processor using the MAGMA library, and gives their performances with the amounts of 1000 and 2000 in the case of processing of small matrices. Note, that there exist works [8,9] on the above mentioned Batched functions, concerning the general matrices, but there are still no papers on Hermitian matrices.
The implementation of the above mentioned functions in a hybrid system is realized through the latest edition of MAGMA 2.2.0 [10] library, in which the Hermitian Batched functions have just been integrated. It should also be noted that in the structure of the functions of the recently released CUBLAS library there are no Batched operations associated with the Hermitian matrices.
The second section of the article presents the stages of processing of the Hermitian Batched matrix-matrix multiplication in the hybrid CPU-GPU system.The test results are presented in the third section, which were made in the case of processing of 1000 and 2000 amount of small matrices. The fourth section is the conclusion.

2. BATCHED IMPLEMENTATION FOR GPU

The name of a Batched BLAS routine follows, and extends as needed, the conventions of the corresponding BLAS routine. In particular, the name is composed of 5 characters, specifying the BLAS routine and described below, followed by the suffix_batched:
The first character in the name denotes the data type of the matrix, as follows:

- **s** float,
- **d** double,
- **c** single complex,
- **z** double complex.

Characters two and three in the name refer to the kind of matrix involved, as follows:

- **ge** All matrices are general rectangular,
- **he** One of the matrices is Hermitian,
- **sy** One of the matrices is symmetric,
- **tr** One of the matrices is triangular.

The fourth and fifth, and in one case sixth, characters in the name denote the operation.
For the Level 3 Batched BLAS, the operations are given as follows:

- **mm** Matrix-matrix product,
- **rk** Rank-k update of a symmetric or Hermitian matrix,
- **r2k** Rank-2k update of a symmetric or Hermitian matrix,

- **sm** Solve a system of linear equations for a matrix of right-hand sides.

The arguments that specify options are of enum type with names *side*, *transa*, *transb*, *trans*, *uplo*, and *diag*. These arguments, along with the values that they can take, are described below:

side has two possible values which are used by the routines as follows:

Side=MagmaLeft: Specifes to multiply a general matrix by symmetric, Hermitian, or triangular matrix on the left.

Side=MagmaRight: Specifes to multiply general matrix by symmetric, Hermitian, or triangular matrix on the right.

transa, *transb*, and *trans* can have three possible values each, which is used to specify the following:

trans=Magma_NoTrans: Operate with the matrix as it is;

trans=Magma_Trans: Operate with the transpose of the matrix;

trans=Magma_ConjTrans: Operate with the conjugate transpose of the matrix.

uplo is used by the Hermitian, symmetric, and triangular matrix routines to specify whether the upper or lower triangle is being referenced, as follows:

uplo=MagmaLower: Lower triangle;

uplo=MagmaUpper: Upper triangle.

The *batchCount* argument is an integer that indicates the number of matrices to be processed.

The description of the matrix consists of the array name (*arrayA*, *arrayB*, or *arrayC*) followed by an array of the leading dimension as declared in the calling function (*lda*, *ldb*, or *ldc*). The i^th values of the *arrayA*, *arrayB*, and *arrayC* are pointers to the arrays of data A_i, B_i, and C_i, respectively. Similarly, the values of *lda[i]*, *ldb[i]*, and *ldc[i]* correspond to the leading dimensions of the matrices A_i, B_i, and C_i, respectively.

Arrays of scalars are named *alpha* and *beta*, where values at position i correspond to the α and β scalars for the BLAS operation involving matrices *Ai*, *Bi*, and *Ci*.

Of the three subprograms we will only show the implementation of the batched matrix-matrix multiplication on GPU, because the realization of the remaining two functions is performed on the same principle, using the appropriate arguments.

Batched Hermitian matrix-matrix multiplication implements the following computation:

```
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 matrices.

In magmablas, the interface is:

*magmablas_chemm_batched(magma_side_t side, magma_uplo_t uplo, magma_int_t m, magma_int_t n, magmaFloatComplex alpha, magmaFloatComplex **dA_array, magma_int_t ldda, magmaFloatComplex **dB_array, magma_int_t lddb, magmaFloatComplex beta, magmaFloatComplex **dC_array, magma_int_t lddc, magma_int_t batchCount, magma_queue_t queue);*

This subprogram has one of the following forms:

If *side* = MagmaLeft, then $C := alpha * A * B + beta * C$,

If *side* = MagmaRight, then, $C := alpha * B * A + beta * C$,

where the matrices A, B, and C are complex Hermitian, and $alpha$ and $beta$ are scalars.

If *uplo* = MagmaUpper, then only the upper triangular part of the Hermitian matrix is to be referenced.

If *uplo* = MagmaLower, then only the lower triangular part of the Hermitian matrix is to be referenced.

dA_array- array of pointers, dimension(batchCount). Each is a complex array A of dimension(ldda, ka), where ka is m when *side* = MagmaLower and is n, otherwise.

ldda-specifies the first dimension of each A as declared in the calling (sub) program.

dB_array- array of pointers,dimension(batchCount). Each is a complex array B of dimension(lddb, n).

lddb- specifies the first dimension of B as declared in the calling (sub) program.

dC_array- array of pointers, dimension(batchCount). Each is a complex array C of dimension (lddc, n).

lddc-specifies the first dimension of C as declared in the calling (sub) program.

batchCount-the number of matrices to operate on.

queue- queue to execute in.

The implementation of this subprogram in the hybrid CPU – GPU system is performed in the following sequence:

1) In CPU, memory is allocated for A, B and C matrices via *magma_cmalloc_cpu()* function of Magma library. For example, for the matrix A this function will have the following form: *magma_cmalloc_cpu(&A, lda*n*batchCount).*
2) In GPU,memory is allocated for A, B and C matrices via *magma_cmalloc()* function of Magma library. For the matrix A this function will have the following form: *magma_cmalloc(&d_A, ldda*n*batchCount).*
3) In GPU, memory is allocated for arrays consisting of references directed to the matrices d_A ,d_B and d_C. For example, for d_A the function *magma_malloc()* will have the following form: *magma_malloc((void**)&dA_array, batchCount * sizeof(*dA_array)).*
4) In the memory of the CPU, the matrices A, B and C are initialized using the function *lapackf77_clarnv()* of the LAPACK library.
5) Matrices A, B and C are moved from the CPU memory to the GPU memory using the *magma_csetmatrix()* function. For the matrix A *magma_csetmatrix(n, n*batchCount, a, lda, d_a, ldda).*
6) Before calling the required function, the program includes the function *magma_sync_wtime(queue)* of time performance, which is integrated into the *magma_timer.h* library.
7) We call *magmablas_chemm_batched ()* matrix-matrix function multiplication for Hermitian small matrices, where the required values of all arguments are given beforehand, for example, the dimension of matrices to be entered, the *upper* or *lower* triangular matrix used and the most important value - *batchCount*, which indicates how many matrices are to be processed.
8) After the operation of the function, fix the implementation time and then calculate the function operation performance.
9) Using the *magma_cgetmatrix()* function, the matrix C is moved from the GPU memory to the CPU memory.

It is very important to note that any magma program begins with the initialized function *magma_init()* and ends with the *magma_finalize()* function of finalization.

It should also be noted that in the hybrid system after the completion of any program, the CPU and GPU memory should be freed. It is done using the functions *magma_free_cpu ()* and *magma_free ()*, respectively.

3. EXPERIMENTAL RESULTS

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 2.2.0 package was installed in accordance with cuda7 environment. For the compilation of MAGMA 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* static and dynamic libraries. MAGMA 2.2.0 package contains *libmagma.a, libmagmablas.a* and *libmagma_sparse.a* libraries.

During the experiments the 1000 and 2000 number of small matrices with the dimension from 32 to 512 were processed.

Figures 1 and 2 show the performance graphics of Batched multiplication of Hermitian small matrices for single complex precision and double complex precision cases, respectively.

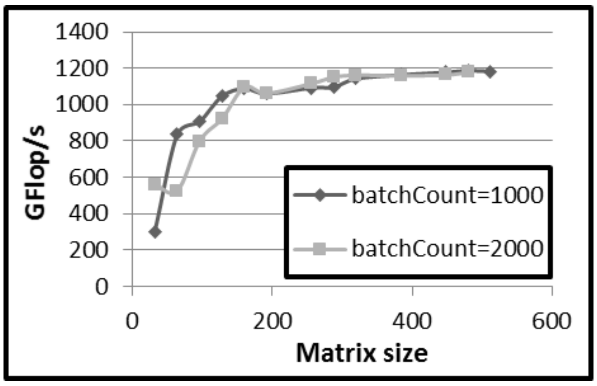

Fig. 1. Performance of Batched HEMM in single complex precision

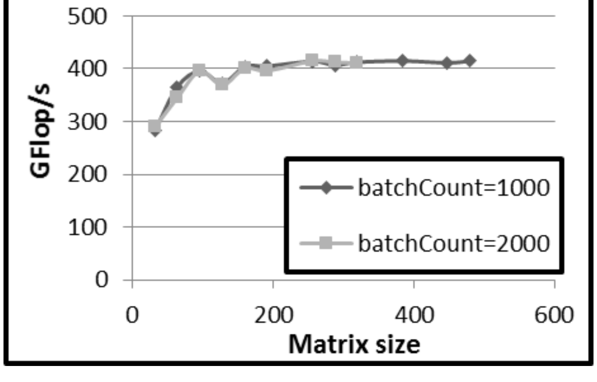

Fig. 2. Performance of Batched HEMM in double complex precision

The results show that as in the case of single complex precision, as well as in the case of double complex precision,no significant differences in performance are observed at 1000 and 2000 number of matrix processing. However, in the case of single complex precision of the mentioned incoming matrices of all cases, the performance is 3 times higher than the double complex precision. As a result of the experiments it was found that in the case of single complex precision, the program does not run at 512 dimension of 2000 number of matrices, it is also not executed in the case of double complex precision with the 512 dimension of 1000 number of matrices, but the program does work in the case of 2000 number of matrices not exceeding 400 dimension.

With respect to the execution time, only note that the processing duration in the case of single complex precision is up to 1 second, and in the case of double complex precision it is up to 2 seconds.

Figures 3 and 4 depict the performance graphs of Hermitian small matrices and vectors of Batched multiplication for the cases of single complex precision and double complex precision, respectively.

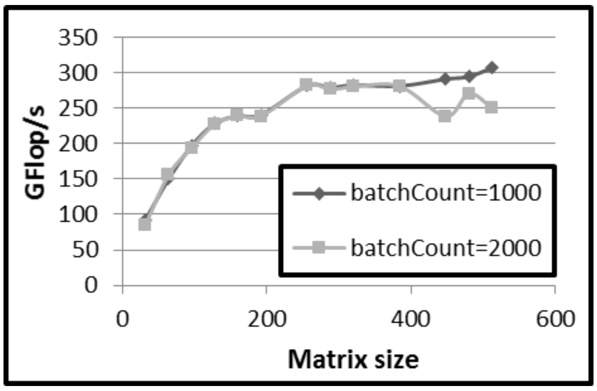

Fig. 3. Performance of Batched HEMV in single complex precision.

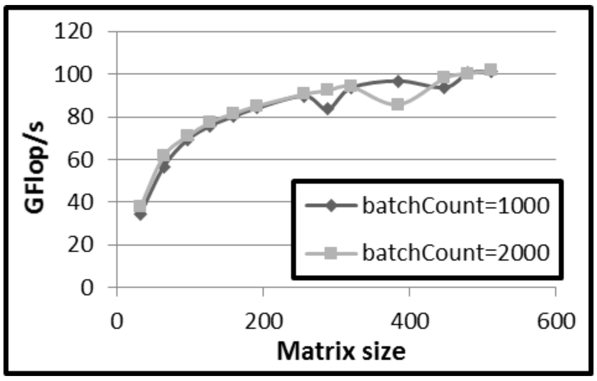

Fig. 4. Performance of Batched HEMV in double complex precision.

The results obtained in the operation of the matrix-vector multiplication show that as in the case of matrices, as well as in this case, in both single complex precision and double complex precision cases at 1000 and 2000 numbers of matrix processing no significant differences are observed. Here again, the performance of single complex precision exceeds that of double complex precision for 3 times.

Execution time duration in both single complex precision and double complex precision cases does not exceed 1 second.

Figures 5 and 6 show the performance graphics of Batched 2nd rank update operation of Hermitian small matrices for single complex precision and double complex precision cases, respectively.

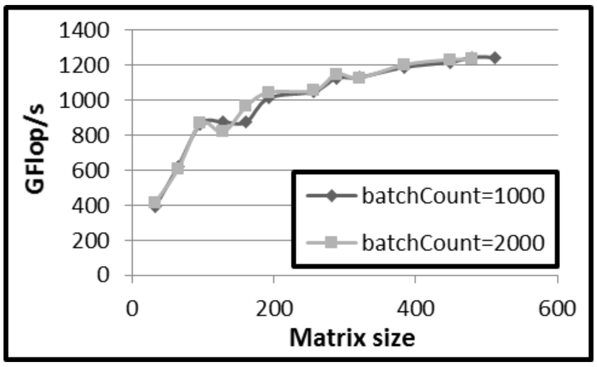

Fig. 5. Performance of Batched HER2K in single complex precision

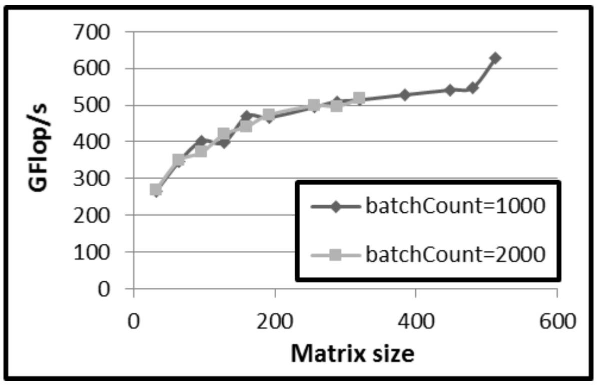

Fig. 6. Performance of Batched HER2K in double complex precision.

In the case of Batched 2nd rank update operation of Hermitian small matrices, like in the aforementioned two operations, again no significant differences in performance are observed at 1000 and 2000 numbers of matrix processing. Here the performance of the single complex precision exceeds that of double complex precision for 2 times. It should also be noted that in case of double precision the matrix dimension of 2000 number must not exceed 400.
In the case of this operation the execution time is 1 second in all cases.

6. CONCLUSION

The performances of Batched matrix-matrix multiplication, matrix-vector multiplication and 2nd rank update operations for Hermitian small matrices were presented in CPU-GPU hybrid system using Tesla K40c graphics processor. In this paper the results of the three Batched calculations are presented for Hermitian small matrices using the MAGMA 2.2.0 library, because they are missing in the structure of Batched functions of even recently released CUBLAS library. Based on the obtained results, we came to the following conclusion that in the case of matrix processing of dimensions from 32 to 512 with both 1000 and 2000 numbers, in the three mentioned Batched calculations no significant differences were observed for single complex precision and double complex precision cases. We also obtained that in the multiplications of matrices and matrix-vector of the single complex precision case the performance exceeds the double complex precision case for 3 times, and in the case of 2nd rank update operation it exceeds for 2 times. With respect to the runtime, only note that the duration of the operations is 1 and 2 seconds.

REFERENCES

[1] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprogramsfor fortran usage. ACM Trans. Math. Softw., 5(3), pp. 308-323, September 1979.
[2] J. J. Dongara, C. B. Moler, J. R. Bunch, and G. W. Stewart. LINPACK Users' Guide. SIAM, Philadelphia, PA, 1979.
[3] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extendedset of FORTRAN basic linear algebra subprograms. ACM Trans. Math. Softw., 14(1), pp. 1-17,1988.
[4] J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Du_. A set of level 3 basic linearalgebra subprograms. ACM Trans. Math. Softw., 16(1), pp. 1-17, March 1990.
[5] Edward Anderson, ZhaojunBai, Christian Bischof, Suzan L. Blackford, James W. Demmel,Jack J. Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven J. Hammarling, Alan McKenney,and Danny C. Sorensen. LAPACK Users' Guide. Society for Industrial and AppliedMathematics, Philadelphia, Third edition, 1999.
[6] Matrix algebra on GPU and multicore architectures (MAGMA), 2014. Available at http://icl.cs.utk.edu/magma/.
[7] CUBLAS 7.5, 2016. Available at http://docs.nvidia.com/cuda/cublas/.
[8] Ahmad Abdelfattah, AzzamHaidar, StanimireTomov, andJack Dongarra: Performance, Design, and Autotuning ofBatched GEMM for GPUs. High Performance Computing, pp.21-38, June 2016.
[9] Ahmad Abdelfattah, AzzamHaidar, StanimireTomov, Jack Dongarra : On the Development of Variable Size Batched Computation for Heterogeneous Parallel Architectures. Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International.
[10] Jack Dongarra, AzzamHaidar, Sven Hammarling, Jonathon Hogg, Pedro Valero-Lara, Samuel D. Relton, StanimireTomov and MawussiZounon : A Proposed API for Batched Basic LinearAlgebra Subprograms. MIMS EPrint: 2016.25, Manchester Institute for Mathematical Sciences School of Mathematics, The University of Manchester, UK, April 2016.