Mathematical Problems of Computer Science 47, 68--75, 2017.

Performances of Factorizations of Complex Hermitian Matrices in the Architecture of GPU Accelerator

Hrachya V. Astsatryan and Edita E. Gichunts

Institute for Informatics and Automation Problems of NASRA
e-mail: hrach@sci.am, editagich@ipia.sci.am

Abstract

Linear algebra LDL^T factorizations are very important and widely used in scientific and engineering calculations. The factorizations with Bunch-Kaufmann and Aasen's algorithms, as well as the LDL^T factorization without pivoting also belong to this class of factorizations.The implementation of the mentioned three factorizations in hybrid architecture are presented in this work, and also performances are given on the NVIDIA Tesla K40c graphic processor using the *MAGMA* library for complex Hermitian matrices.

Keywords: MAGMA library, LDL^T factorization, Aasen's algorithm, Bunch-Kaufman algorithm, Hermitian matrix,Triangular matrix.

1. Introduction

Solutions of linear system of equations of Hermitian matrices have repeatedly been used in physics. To calculate the solutions of linear system of equations of $Az = b$ form, the classical method turns A matrix into the following LDL^T analysis form: $PAP^T = LDL^T$,where L is the unit lower triangular, D is the block diagonal with either 1-by-1 or 2-by-2 diagonal blocks, and P is a permutation matrix to ensure the numerical stability of the factorization.

The factorizations with Bunch-Kaufmann and Aasen's algorithms, as well as the LDL^T factorization without pivoting are very popular.

In this paper we present the mentioned factorizations for complex Hermitian matrices. Novelty of this work is the realization of LDL^T factorization by the Aasen's algorithm on GPU with the application of *MAGMA* library, as it has just been integrated into the *MAGMA 2.0.1* package. The realizations of LDL^T factorizations without pivoting and with Bunch-Kaufmann algorithm are also presented in this work, in order to have a complete understanding of performances of LDL^T factorizations of complex Hermitian matrices. Note also that this problem is presented in cases of complex single and complex double precisions.

The work contains the following sections: the first sectionis introduction, the second one describes the steps of the above mentioned algorithms in CPU / GPU hybrid architecture. The

third section contains the experiment results, where the graphs of performance and time of those algorithms are shown, as well as the analysis of comparisons of factorizations are given. The fourth section is the conclusion based on the experiment results.

2. LDL^T Factorizationin CPU/GPU Hybrid Architecture

Solutions of linear system of equations of the form $A * Z = B$ are being used in science on repeated occasions, where A is a complex Hermitian matrix, and LDL^T matrix factorization is used. Describe three cases of these factorizations in hybrid architecture that are resolved with Bunch-Kaufmann and Aasen's algorithms, as well as the case without pivoting. *LAPACK* library is used in the systems with shared memory, where as in hybrid systems its parallelized *MAGMA* library is used.

2.1Aasen's Algorithm

Aasen's algorithm [1] factorizes A into an LDL^T decomposition of the form $PAP^T = LDL^T$,where P is a permutation matrix, L is a unit lower triangular matrix, and D is a Hermitian matrix. To exploit the memory hierarchy on a modern computer,a partitioned-version of Aasen's algorithm was recently proposed [2]. This algorithm first factorizes a panel in a left-looking fashion, and then uses *BLAS-3* operations toupdate the trailing submatrix in a right-looking way.
The blocked-version of Aasen's algorithm computes an LDL^T factorization of A, where D is a banded matrix of theband width equal to the block size.
Aasen's algorithm is a *chetrf_aasen* subprogram of *MAGMA* library for complex Hermitian matrices.

CHETRF_AASEN computes the factorization of a complex Hermitian matrix A based on a communication-avoiding variant of the Aasen's algorithm. The form of the factorization is as follows:

$A = U * D * U ** H$ if uplo = MagmaUpper, or $A = L * D * L ** H$ if uplo = MagmaLower,

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and banded matrix of theband width equal to the block size.

There are three cases of LDL^T factorization computing by the Aasen's algorithm which areright-and left-looking algorithms and a blocked left-looking version of the algorithm. Factorization by the Aasen's algorithmis implemented with the left-looking algorithm in CPU / GPU hybrid architecture.

The intermediate Heisenberg matrix H is used to calculate the LDL^T factorization that is defined by $H = TL^T$. In the initial step the matrix A is completely transferred from the CPU to the GPU memory through *magma_csetmatrix_async ()* function.
The algorithm consists of the following sequence: $H(1{:}j-1, j)$ is calculated and $T(j, j)$ is updated:
Initially the Aasen's algorithm calculates the j^{-th} column of H, that is:

$$H(i,j) = T(i,i-1) * L(j,i-1)' + T(i,i) * L(j,i)' + T(i,i+1) * L(j,i+1)',$$

wherei is modified from 1 to $j-1$. Using *magma_cgemm ()* function for matrix multiplication the $(j;j)^{th}$ element of H will be the $(j;j)^{th}$ element of the equation $A=LH$. In the next step $T(j,j)$ will be the $(j;j)^{th}$ element of the equation $H=TL^T$, namely

$$T(j,j)=A(j,j)-L(j,1:j)*H(1:j,j)$$

is calculated using *magma_cher2k ()* function, which performs the Hermitian rank-2k update, and *magmablas_csymmetrize ()* function, which copies the lower triangle to upper triangle, or vice-versa, to make dA a general representation of a symmetric matrix.
If $j>1$, then

$$T(j,j)=T(j,j)-L(j,j)*T(j,j-1)*L(j,j-1)'$$

is calculated.

2.2Bunch-Kaufman Algorithm

Bunch-Kaufmann algorithmhas a very wide application in the solution of linear system of equations of Hermitian matrices through LDL^T factorization. The Bunch-Kaufman method performs the following decomposition: $PAP^T=LDL^T$,where L is an n-by-n lower triangular matrix with a unit diagonal, and D is a block diagonal matrix with either 1-by-1 or 2-by-2 diagonal blocks [3]. The pivoting strategies to compute the permutation matrix P for the LDL^T factorization include complete pivoting (Bunch-Parlett algorithm) [4], partial pivoting (Bunch-Kaufman algorithm) [5], rook pivoting (bounded Bunch-Kaufman) [6, p. 523]. In particular, the Bunch-Kaufman and rook pivoting are implemented in *LAPACK* [7]. A diagonal pivoting algorithm, the subprogram of which has the *name_hetrf()* in *LAPACK* library, is generally available by NetLib [8]. A concise matrix-notation description of this algorithm can be found in [9].

Bunch-Kaufman algorithm is *chetrf* subprogram of *MAGMA* library for complex Hermitian matrices.

CHETRF computes the factorization of a complex Hermitian matrix Ausing the Bunch-Kaufman diagonal pivoting method. The form of the factorization is

$A=U*D*U^H$ if uplo = MagmaUpper, or $A=L*D*L^H$ if uplo = MagmaLower,

where U (or L) is a product of permutation and unit upper (lower) triangular matrices, and D is Hermitian and block diagonal with1-by-1 and 2-by-2 diagonal blocks. This is the blocked version of the algorithm, calling Level 3 *BLAS*.
In the first step of hybrid CPU / GPU programming the triangular matrix moves from CPU to GPU through *magma_csetmatrix_async ()* function. If uplo = MagmaUpper, then the upper triangular matrix A moves to GPU, and if uplo = MagmaLower, then the lower triangular matrix A moves to GPU. Afterwards $U*D*U'$ or $L*D*L'$ factorization of triangular matrix A is carried out in the following way. In case ofthe upper triangular matrix each $k-kb+1:k$ column of the matrix is factorized, as well as uses a blocked code to update the columns $1:k-kb$ through the following *magma_clahef_gpu()* function. *Magma_clahef_gpu()* computes a partial factorization of a complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. K is the main loop index, decreasing from N to 1 in steps of KB, where KB is the number of columns factorized by *magma_clahef_gpu()*. In case of the lower triangular matrix each $k-kb+1:k$ column of the matrix is factorized, as well as a blocked code to update the

columns $k + kb : n$. K is the main loop index, increasing from 1 to N in steps of KB, where KB is the number of columns factorized by *magma_clahef_gpu( ).*

2.3 LDL^{T} Factorizations with no Pivoting

This method of factorization for complex Hermitian matrices is performed by *chetrf_nopiv()* function of *MAGMA* library.
CHETRF_nopiv computes the LDL^T factorization of a complex Hermitian matrix A. The factorization has the form

$$A = U^H * D * U \text{ if uplo = MagmaUpper, or } A = L * D * L^H \text{ if uplo = MagmaLower,}$$

where U is an upper triangular matrix, L is a lower triangular, and D is a diagonal matrix. This is the block version of the algorithm, calling Level 3 *BLAS*.
In hybrid CPU/GPU programming $A = U' * D * U$ factorization without pivoting will be performed in the following sequence. The matrix completely moves to GPU memory. After wards, all the $A(j, j)$ diagonal elements in the main cycle move to CPU memory. CPU is used to calculate the LDL^T factorization of the diagonal block. This diagonal block on CPU is factorized through *magma_chetrf_nopiv_cpu ()* function. Once the resulting LDL^T factors of the diagonal block are copied back to the GPU, the corresponding off-diagonal blocks of the L -factor are computed by the triangular solve on the GPU. Then the copy ofj column of U is sent to CPU. Compute the off-diagonal blocks of current block column by *magma_ctrsm()* function. After wards update the trailing submatrix with D by *magmablas_clascl_diag()* function. Finally, update each block column of the trailing submatrix, calling a matrix-matrix multiply on the GPU. $A = L * D * L'$ factorization in similar sequence will be performed for the lower triangular matrix.

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, GDDR5 memory interface, and CUDA C programming environment.

The operation system of Tesla K40 is Ubuntu 14.04.2 LTS.cuda7 programming environment was used for the realization of programs. *MAGMA 2.0.1* 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 2.0.1* package contains *libmagma.a* and *libmagma_sparse.a* libraries.

Figures 1 and 2 show the time and performance graphs of factorizations with the Aasen's algorithm, Bunch-Kaufmann algorithm and LDL^T factorization without pivoting for complex single precision.

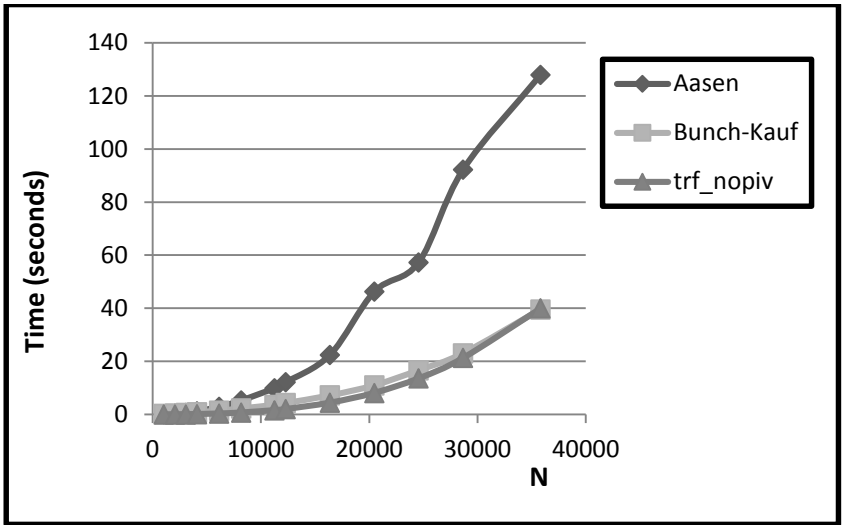

Fig. 1. Complex Single Precision

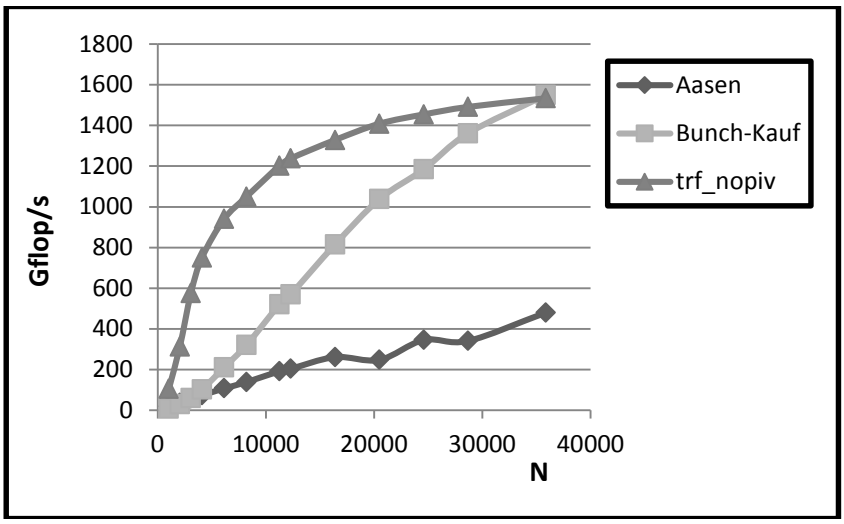

Fig. 2. Complex Single Precision

In case of complex single precisionat most $N = 35840$ -dimensional matrix can be sent to Tesla K40c graphical processor. The experiment results show that in case of up to $N = 16384$ dimensional incoming matrix the LDL^T factorization without pivoting, both in terms of time and performance, exceeds the Aasen's factorization algorithm for 5 times and the Bunch-Kaufmann factorization algorithm – for 3 times. Where as the Bunch-Kaufmann algorithm exceeds the Aasen's algorithm for 3 times. Increasing the incoming matrix up to $N = 35840$ dimension the Bunch-Kaufmann algorithm is approaching the factorization without pivoting, and it turns out that they exceed the Aasen's factorization algorithm for 3 times both in terms of time and performance.

Figures 3 and 4 show the time and performance graphs of the mentioned three factorizations for complex double-precision.

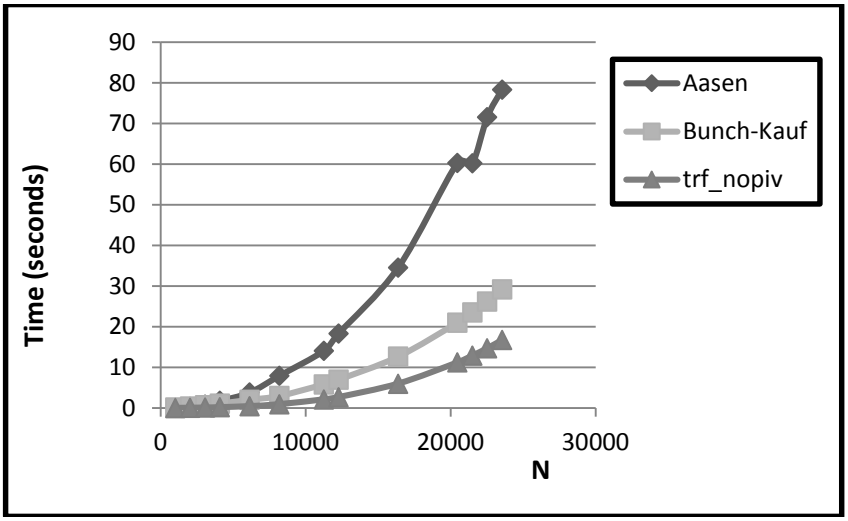

Fig. 3. Complex Double Precision

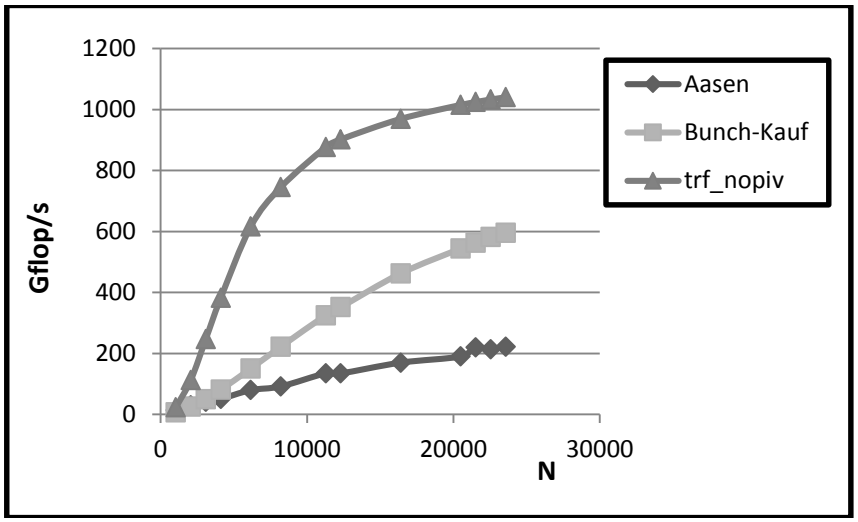

Fig. 4. Complex Double Precision

And in the case of complex double precision at most $N = 23552$ -dimensional matrix can be sent to Tesla K40c graphical processor. The experiment results show that in this casethe LDL^T factorization without pivoting in terms of time and performance exceeds the Aasen's factorization algorithm for 5 times and the Bunch-Kaufmann factorization algorithm – for 2 times, whereas the Bunch-Kaufmann algorithm, in terms of both time and performance, exceeds the Aasen's algorithm for 2.5 times.

4. Conclusion

We presented the performances of complex Hermitian matrix LDL^T factorizations in CPU/GPU hybrid architecture. We also presented the performance results of factorizations with Aasen's algorithm, Bunch-Kaufmann algorithm and LDL^T factorization without pivoting using *MAGMA* library. Based on the obtained results we came to the conclusion that the LDL^T factorization without pivotingis in a leading position by its high performance. In case of complex single

precision it equalsthe Bunch-Kaufmann algorithm along with the increase of the incoming matrix, and in the case of complex double precision it exceeds for 2 times. It exceeds the Aasen's factorization algorithm for 5 times in complex single and double precisions cases. The latter concedes the Bunch-Kaufmann algorithm in both cases for 2.5-3 times.

References

[1] J. Aasen, "On the reduction of a symmetric matrix to tridiagonal form", *BIT 11*, pp. 233-242, 1971.
[2] M. Rozlo˘zn´ık, G. Shklarski and S. Toledo, "Partitioned triangular tridiagonalization", *ACM Trans. Math. Softw.*,vol. 37, no. 4, pp. 1–16, 2011.
[3] G. Golub and Ch. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, second edition, 1989.
[4] J. R. Bunch and B. N. Parlett,"Direct methods for solving symmetric indefinite systems of linear equations", *SIAM J. Numerical Analysis*, vol. 8,pp. 639-655, 1971.
[5] J. R. Bunch and L. Kaufman,"Some stable methods for calculating inertia andsolving symmetric linear systems", *Mathematics of Computation*, vol. 31,pp. 163-179, 1977.
[6] C. Ashcraft, R. G. Grimes and J. G. Lewis,"Accurate symmetric indefinite linear equation solvers", *SIAM J. Matrix Anal. and Appl.*, vol. 20, no. 2, pp. 513-561, 1998.
[7] E. Anderson, Z. Bai, J. J. Dongarra, A. Greenbaum, A. McKenney, J. DuCroz, S. Hammarling, J. W. Demmel, C. Bischof and D. Sorensen, "LAPACK:a portable linear algebra library for high-performance computers", *Proceedings of the1990 ACM/IEEE conference on Supercomputing*.
[8] C. Anderson et al.,*LAPACK User's Guide*, SIAM Press, Philadelphia,1992.
[9] P. E. Strazdins. A dense complex symmetric indefinite solver for theFujitsu AP3000, Technical Report TR-CS-99-01, Computer ScienceDept, Australian National University, May 1999.

Submitted 03.10.2016, accepted 27.01.2017.

Կոմպլեքս Հերմիտյան մատրիցների ֆակտորիզացիաների արտադրողականությունները GPU արագագործչի ճարտարապետությունում

Հ. Ասցատրյան և Է. Գիչունց

Ամփոփում

Գծային հանրահաշվի LDL^T ֆակտորիզացիաները շատ կարևոր են և մեծ կիրառություն ունեն գիտական և ինժեներական հաշվարկներում: Այդ ֆակտորիզացիաների դասին են պատկանում Բանչ-Կաուֆմանի և Ասենի ալգորիթմներով ֆակտորիզացիաները, ինչպես

նաև առանց պտույտի LDL^T ֆակտոիզացիան: Այս աշխատանքում ներկայացված են նշված երեք ֆակտորիզացիաների իրականացումները հիբրիդային ճարտարապետությունում, և տրված են արտադրողականություններ NVIDIA Tesla K40c գրաֆիկական պրոցեսորի վրա MAGMA գրադարանի կիրառմամբ կոմպլեքս Հերմիտյան մատրիցների համար:

Производительности факторизации комплексных Эрмитовых матриц в архитектуре ускорителя GPU

Г. Асцатрян и Э. Гичунц

Аннотация

LDL^T факторизации линейной алгебры очень важны и широко используются в научных и инженерных расчетах. К этому классу относятся факторизации с алгоритмами Банча-Кауфмана и Аасена, а также факторизация LDL^T без поворота. В этой работе представлены реализации упомянутых трех факторизаций в гибридной архитектуре, а также представлены производительности на графическом процессоре NVIDIA Tesla K40c с использованием библиотеки MAGMA для комплексных Эрмитовых матриц.