Mathematical Problems of Computer Science 44, 93--100, 2015.

Performance Comparisons of Eigensolutions Algorithms of a Symmetric Tridiagonal Matrix on GPU Accelerators

Hrachya V. Astsatryan and Edita E. Gichunts

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

Abstract

While finding the solutions of Hermitian matrix eigenproblem it is a key issue to find an efficient version of algorithms of symmetric tridiagonal solutions. In this paper these algorithms are compared for complex Hermitian matrices in hybrid systems. The methods were carried out on the Tesla C1060 and Tesla K40 GPU accelerators and the performances are presented as between the methods, as well as between the accelerators.

Keywords: GPU accelerator; hybrid architectures, tridiagonal matrices, MRRR, D&C, BI, algorithm.

1. Introduction

Finding the matrix eigenvalues and eigenvectors is one of the central issues in linear algebra. In particular, the solutions to the eigenproblem are in LaPACK, ScaLAPACK and PLaPACK packages, in the systems with general and distributed memory, respectively. Matrix eigensolutions can achieve a higher performance through MAGMA library in hybrid architecture on GPU accelerators.

The aim of MAGMA [1,2] library is the realization of LaPACK library sub-programs in the architecture of hybrid systems.

Due to the development of productivity of ranking algorithms, this problem is overcome in hybrid architecture.

Finding of eigenproblem solutions of Hermitian matrix, as a rule, takes place through the following three stages:

1. Through the Householder transformation the matrix is reduced to a tridiagonal form.
2. The tridiagonal matrix solutions are found through one of the following algorithms:
   - QR iteration [3,4] ,
   - Bisection for the eigenvalues and inverse iteration for the eigenvectors(BI) [5,6],
   - Divide & Conquer method (D&C) [7,8],
1. Back transformation to find the eigenvectors for the full problem from the eigenvectors of the tridiagonal problem.

In fact, the algorithm performance may depend on the matrix, platform, and on the underlying linear algebra libraries BLAS, LaPACK, ATLAS and MAGMA. MAGMA library is used to realize the projects on GPU accelerator. The reduction of matrix to tridiagonal form in all cases is carried out through the Householder transformation, as well as through magma_chetrd function of MAGMA library. It should also be noted that in all releases of MAGMA library for complex Hermitian matrices the eigenproblem solving function with QR iteration is missing. Therefore, only the remaining (D & C), (MRRR), (BI) three cases will be considered.

In this work the solutions of Hermitian problem are presented by means of three methods on GPU accelerators. Moreover, the performance comparisons for three algorithms are presented as between the methods, as well as between the accelerators.

Section 2 briefly presents the mentioned algorithms. Section 3 states about the resources required for the implementation of programs. Sections 4 and 5 give the results of the mentioned three algorithms on GPU accelerators for both standard and generalized forms of eigensolutions of complex Hermitian matrices, respectively. Section 6 covers the conclusion of the obtained results.

2. Description of Algorithms

2.1.Divide and conquer. The divide-and-conquer method can be described in terms of a binary tree where each node corresponds to a submatrix and its eigenpairs, obtained through recursively dividing the matrix in halves; see the exposition in [12].

The tree is processed bottom up, starting with submatrices of size 25 or smaller. DC uses QR to solve the small eigenproblems and then computes the eigenpairs of a parent using the already computed eigenpairs of the children.

A parent's eigenvalues can be computed as solutions of a secular equation. The eigenvector computation consists of two steps. The first one is a relatively inexpensive scaling step. The second one, which is most of the work, multiplies the eigenvectors of the current matrix by the eigenvector matrix accumulated so far. This step uses the level 3 BLAS (BLAS 3) routine GEMM (dense matrix-matrix multiply). In the worst case, DC is an $O(n^3)$ algorithm.

2.2.Multiple relatively robust representations. Since its introduction in the 1990s, much has been written about the MRRR algorithm, its theoretical foundation is discussed in several publications [13, 14, 15, 16] and practical aspects of efficient and robust implementations are discussed in [17, 18, 19, 20, 21, 22]. One could say, with an implementation of the algorithm in the widely used LAPACK library and the description of (parts of) the algorithm in textbooks such as [23], MRRR has become mainstream.

MRRR is a sophisticated variant of inverse iteration that avoids Gram–Schmidt orthogonalization and thus becomes an $O(n^2)$ algorithm. The algorithm can be described in terms of a (generally irregular) representation tree. The root node describes the entire spectrum of T, and the children define gradually refined eigenvalue approximations. The overall complexity of the algorithm depends on the clustering of the eigenvalues. If some eigenvalues of T agree to d digits on average, then the algorithm has to do work proportional to dn^2. The algorithm uses a random perturbation to ensure with high probability that eigenvalues cannot be too strongly clustered; see [24] for details. MRRR cannot make use of higher-level BLAS.

2.3.Bisection and inverse iteration. Bisection based on Sturm sequences requires O(nk) operations to compute k eigenvalues of T. If the distance between the eigenvalues is large enough (relative to ||T||), then computing the corresponding eigenvector by inverse iteration also is an O(nk) process. If, however, the eigenvalues are not well separated, Gram–Schmidt orthogonalization is employed to try to achieve numerically orthogonal eigenvectors. In this case

the complexity of the algorithm increases to $O(nk^2)$. In the worst case where almost all eigenvalues of T are "clustered," the complexity can increase to $O(n^3)$. Furthermore, from the accuracy point of view this procedure is not guaranteed to be reliable; see [11, 25]. Neither bisection nor inverse iteration make use of higher-level BLAS.

3. Software Development

The GPU equipment in hybrid systems, due to its unique architecture, is used as an accelerator to process the limited computational applications. The popularity of GPU-based hybrid systems started with the release of the NVIDIA Compute Unified Device Architecture (CUDA) and the extensions of industry-standard programming languages, such as C, C++ and Fortran, which made the GPUs easier to program, allowing the developers to exploit the computational power of modern GPU devices.

To realize the programs, the required software is presented on Tesla C1060 and Tesla K40 GPU accelerators.

The architecture of Tesla C1060 consists of 240 processor cores, using the maximum capacity of parallelization. It is endowed with a high bandwidth transmission of messages between CPU and GPU, and also has 4 GB of global memory, 512-bit GDDR3 memory interface and CUDA C programming environment.

The operation system on Tesla C1060 is Ubuntu 12.04.5 LTS, and cuda4 programming environment was used for realization of programs. MAGMA 1.3.0 package was installed. lapack-3.4.2, clapack-3.2.1 and atlas-3.8.0 packages were installed for this library compilation. gcc-4.4, gfortran-4.4 and nvcc compilers were used. During the compilation a number of references of static and dynamic libraries were made in make.incfile, such as libf77blas.a, libcblas.a, libf2c.a, libcublas.so, libcudart.so, libstdc ++.so, libpthread.so, libdl.so. MAGMA 1.3.0 package contains libmagma.a and libmagmablas.a libraries.

The architecture of Tesla K40 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 K40is Ubuntu 14.04.2 LTS.cuda7 programming environment was used for the realization of programs. MAGMA 1.6.1 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, libstdc ++. so, libpthread.so, libdl.so static and dynamic libraries. MAGMA 1.6.1 package contains libmagma.a and libmagma_sparse.a libraries.

4. Performance Results of Eigenproblem Solutions for a Standard Form

Finding of eigenproblem solutions of complex Hermitian matrices for $Az = \lambda z$ standard form is carried out with the help of the following functions of MAGMA library:

- magma_xheevdx (D&C),
- magma_xheevr (MRRR),
- magma_xheevx (BI).

Moreover, x can be c or z in complex and double complex cases, respectively.

Figures 1 and 2 show the time-dependent graphs for three methods in the case of standard form on Tesla C1060 and Tesla K40 GPU accelerators, respectively.

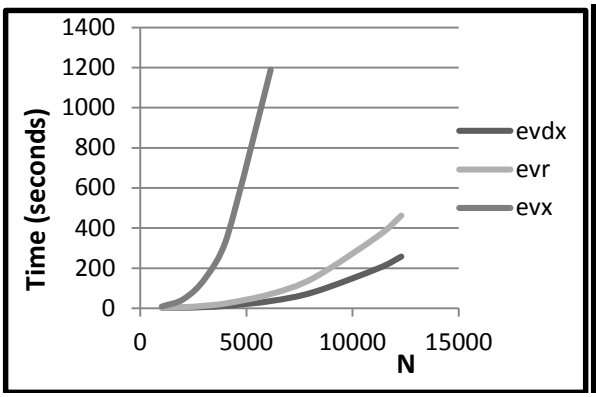

Fig. 1. Tesla C1060, standard eingensolvers.

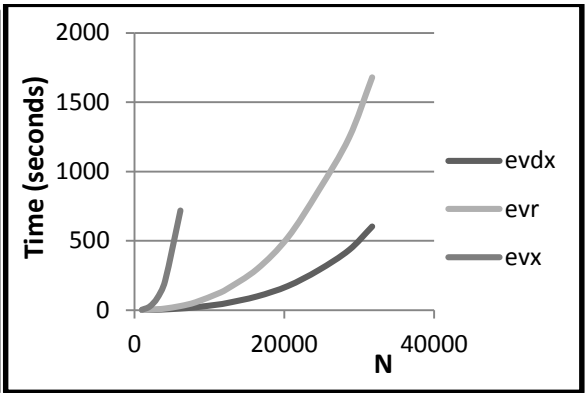

Fig. 2. Tesla K40, standard eingensolvers.

Since the global memory of Tesla C1060 is 4 GB, and the matrix A should be wholly moved to the global memory of GPU, hence, the maximum dimensionality of the input complex matrix can be 12288*12288. In the case of Tesla K40 GPU accelerator it can be 31744 * 31744 as its global memory is 12 GB.

The results show that MRRR algorithm, in a standard form, for 2-2.5 times concedes the DC algorithm. For example, on Tesla C1060 accelerator in case of 12288*12288 maximum dimensional input matrix, the DC algorithm is carried out at gpu_time = 258sec., and in the case of MRRR algorithm it is fulfilled atgpu_time = 462sec. On Tesla K40 GPU accelerator in case of 31744*31744 maximum input-dimensional complex matrix, the DC algorithm is carried out atgpu_time = 603sec., and in case of MRRR algorithm at gpu_time = 1680sec. .

Figures 3 and 4 show in a standard form case the comparisons of GPU accelerators of time-dependent DC and MRRR algorithms with equal amounts of input matrices. The maximum dimensionality of the input matrix will be equal to the possible maximum dimensionality of the matrix used on Tesla C1060.

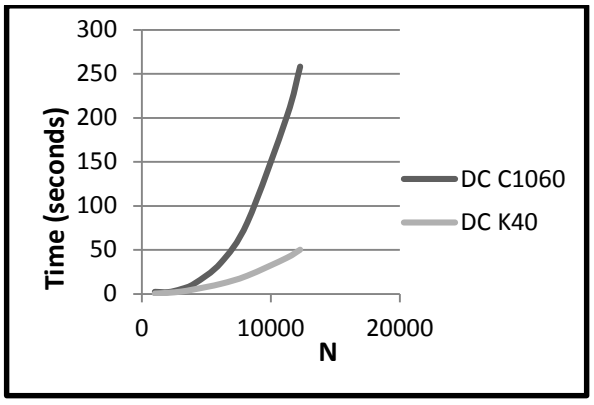

Fig. 3. DC algorithm.

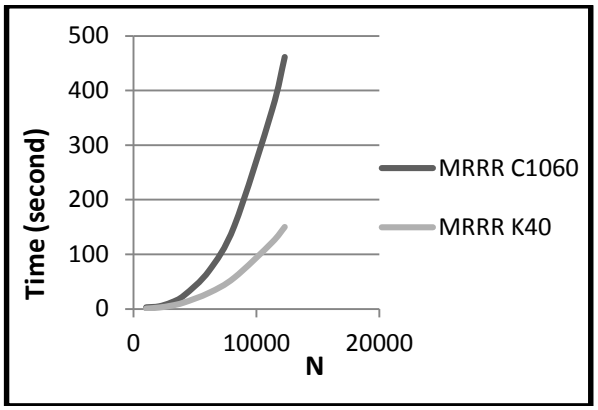

Fig. 4. MRRR algorithm.

Obviously, Tesla C1060 much concedes the Tesla K40 by its architecture, but let's present the results in the case of these two algorithms. For example, in case of the input matrix with 12288*12288 maximum dimensionality on Tesla C1060 accelerator, the DC algorithm is implemented atgpu_time = 258sec., whereas on Tesla K40 it is implemented atgpu_time = 50sec.. MRRR algorithm is implemented on Tesla C1060 accelerator atgpu_time = 462sec., whereas on Tesla K40 - at gpu_time = 150sec..

5. Performance Results of Eigenproblem Solutions for Generalized Form

Finding of eigensolutions of complex Hermitian matrices for Az= λBz generalized form is carried out with the help of the following functions of MAGMA library:

- magma_xhegvdx (D&C),
- magma_xhegvr (MRRR),
- magma_xhegvx (BI).

x can be c or z in complex and double complex cases, respectively.

Figures 5 and 6 show the time-dependent graphs for three methods in the case of generalized form on Tesla C1060 and Tesla K40 GPU accelerators, respectively.

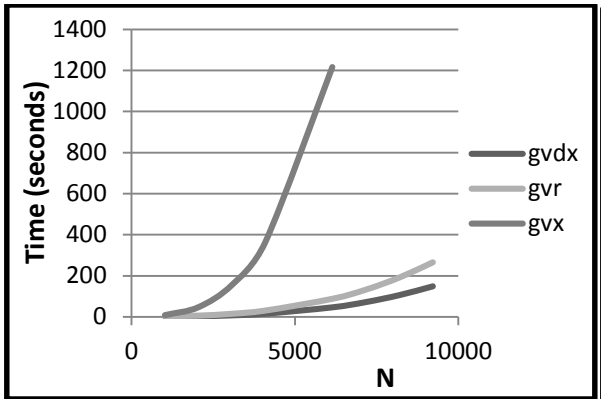

Fig. 5.Tesla C1060, generalized eingensolvers

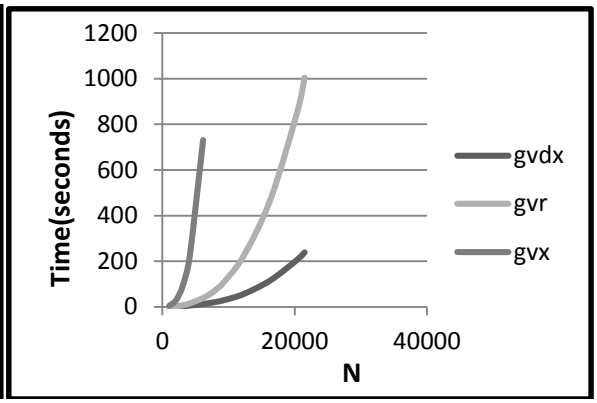

Fig. 6. Tesla K40, generalized eingensolvers

Since for a generalized form A and B matrices should be wholly moved to the global memory of GPU accelerators, therefore in case of Tesla C1060 the maximum dimensionalities of input matrices will be 9216*9216, and in case of Tesla K40 they will be 21504*21504.

The results show that for a generalized form together with the increase in dimensionalities of input matrices the MRRR algorithm concedes the DC algorithm for 1.5-4times. For example, on Tesla C1060 accelerator in case of 9216*9216 maximum dimensional input matrices, the DC algorithm is carried out at gpu_time=150sec., and in the case of MRRR algorithm it is fulfilled at gpu_time = 265sec. On Tesla K40 GPU accelerator in case of 21504*21504 maximum input-dimensional complex matrices, the DC algorithm is carried out at gpu_time = 603sec., and in case of MRRR algorithm at gpu_time = 1005sec..

Figures 7 and 8 show in a generalized form case the comparisons of GPU accelerators of time-dependent DC and MRRR algorithms with equal amounts of input matrices. The maximum dimensionality of the input matrix will be equal to the possible maximum dimensionality of the matrix used on Tesla C1060.

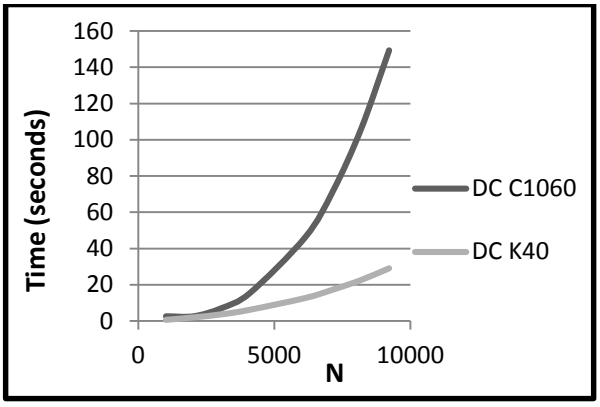

Fig. 7. DC algorithm

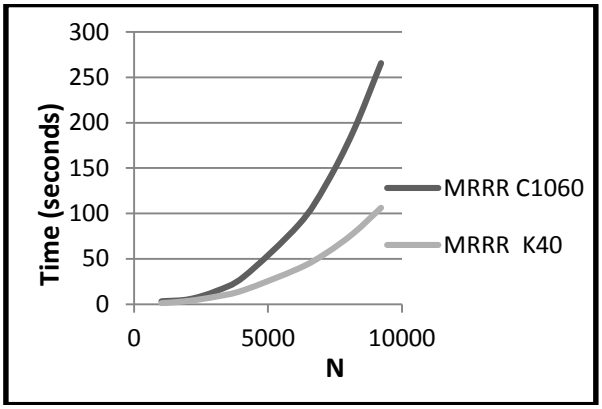

Fig. 8. MRRR algorithm

For the generalized form the results of these two algorithms will be presented. For example, in the case of input matrices with 9216*9216 maximum dimensionality on Tesla C1060 accelerator, the DC algorithm is implemented at gpu_time = 150sec., whereas on Tesla K40 it is implemented at gpu_time = 29sec. MRRR algorithm is implemented on Tesla C1060 accelerator at gpu_time = 266sec., whereas on Tesla K40 - at gpu_time = 106sec..

6. Conclusion

The performance studies of solutions of a symmetric tridiagonal matrix were carried out on both accelerators Tesla C1060 and Tesla K40. Our assessment on this issue considers the speed of the bisection and inverse iteration (BI), the divide-and-conquer method (DC), and the method of multiple relatively robust representations of (MRRR) algorithms in complex Hermitian matrices. The conclusions are as follows:

- DC and MRRR algorithms, in case of large matrices, are rather quickly than in case of BI. Test results show that the programs using BI algorithm, in both cases of standard and generalized forms, are 20 times slower than the programs using DC and MRRR algorithms.
- The results showed that the MRRR algorithm, as in both cases of standard and generalized forms, as well as on both GPU accelerators is inferior to the DC algorithm. But for the implementation of DC algorithm much more workspace is required, than for the implementation of MRRR algorithm. Therefore, the choice of the algorithms DC and MRRR depends on the user, which algorithm is more efficient for the intended issue.
- Comparing the work of two GPU accelerators, we see that on Tesla K40 in case of an input matrix with the same dimensionality, DC algorithm is implemented 5 times faster than on Tesla C1060, and the MRRR algorithm is implemented 3 times faster.

References

[1] “MAGMA Matrix Algebra on GPU and Multicore Architectures”, [Online]. Available:http://icl.cs.utk.edu/magma/, 2014.

[2] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, S. Tomov, “Numerical linear algebra on emerging architectures: The

PLASMA and MAGMA projects ", Journal of Physics: Conference Series, vol. 180, no. 1, 2009.

[3] B. N. Parlett, *The Symmetric Eigenvalue Problem. Englewood Cliffs*, NJ: Prentice-Hall, 1980.

[4] G. H.Golub, C. F. V.Loan, *Matrix Computations*, 3rd ed. Baltimore: The Johns Hopkins University Press, 1996.

[5] I. C. Ipsen, "Computing an eigenvector with inverse iteration", *SIAM review*, vol. 39, no. 2, pp. 254–291, 1997.

[6] I. S. Dhillon, "Current inverse iteration software can fail", *BIT Numerical Mathematics*, vol. 38, no. 4, pp. 685–704, 1998.

[7] J. J. M.Cuppen, "A divide and conquer method for the symmetric tridiagonaleigenproblem", *NumerischeMathematik*, vol. 36, no. 2, pp. 177–195, 1980.

[8] M. Gu and S. C. Eisenstat, "A divide-and-conquer algorithm for the symmetric tridiagonaleigenproblem", *SIAM Journal on Matrix Analysis and Applications*, vol. 16,no. 1, pp. 172–191,1995.

[9] I. S. Dhillon,B. N. Parlett, C. Vomel, "The design and Implementation of the MRRRalgorithm", *ACM Trans. Math. Soft.*,vol. 32, no. 4, pp. 533–560, 2006.

[10] I. S. Dhillon and B. N.Parlett, "Multiple representations to compute orthogonaleigenvectors of symmetric tridiagonalmatrices", *Linear Algebra and its Applications*,vol. 387, pp. 1–28, 2004.

[11] B. N. Parlett and I. S. Dhillon, "Relatively robust representations of symmetrictridiagonals", *Linear Algebra and its Applications*, vol. 309, no. 1–3, pp.121–151, 2000.

[12] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[13] I.Dhillon, and B. Parlett,"Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices",*Linear Algebra Appl*., vol. 387, pp. 1–28. 2004.

[14] I. Dhillonand B. Parlett,"Orthogonal eigenvectors and relative gaps",*SIAM J. Matrix Anal. Appl.*, vol.25,pp. 858–899, 2004.

[15] P. Willems, *On MR3-type Algorithms for the Tridiagonal Symmetric Eigenproblem and Bidiagonal SVD*. PhD thesis, University of Wuppertal, 2010.

[16] P. Willems and B. Lang, *A Framework for the MR3 Algorithm: Theory and Implementation*. Tech. Rep. 11/21, BergischeUniversit¨at Wuppertal, 2011.

[17] I. Dhillon, B. Parlettand C. V¨omel,"Glued matrices and the MRRR algorithm", *SIAM J. Sci. Comput.*, vol. 27, no. 2, pp. 496–510, 2005.

[18] I. Dhillon, B. Parlett and C. V¨omel, "The design and implementation of the MRRR algorithm",*ACM Trans. Math. Software*, vol. 32, pp. 533–560, 2006.

[19] O. Marques, B. Parlettand C. V¨omel, "Computations of eigenpairsubsets with the MRRR algorithm",*Numerical Linear Algebra with Applications*, vol. 13, no. 8,pp. 643–653, 2006.

[20] O. Marques, E. Riedy and C. V¨omel, "Benefits of IEEE-754 features in modern symmetric tridiagonaleigensolvers",*SIAM J. Sci. Comput.*, vol. 28, pp. 1613–1633, 2006.

[21] M. Petschow and P. Bientinesi, "MR3-SMP: A symmetric tridiagonaleigensolver for multi-core architectures",*Parallel Computing*, vol. 37, no. 12, pp. 795 – 805, 2011.

[22] C. V¨omel, "ScaLAPACK's MRRR Algorithm", *ACM Trans. Math. Software*, vol. 37, pp 1 , 2010.

[23] D. Watkins, *Fundamentals of Matrix Computations*. Pure and applied mathematics. John Wiley & Sons, 2010.

[24] I. S. Dhillon, B. N. Parlett, and C. VËomel, “Glued matrices and the MRRR algorithm”,*SIAM J. Sci. Comput*., vol27, pp. 496–510, 2005.
[25] I. S. Dhillon, “Current inverse iteration software can fail”, BIT Numerical Mathematics, vol. 38 pp. 685–704, 1998.

Submitted 31.08.2015, accepted 25.11.2015

Սիմետրիկ երեք անկյունագծային մատրիցի սեփական լուծումների ալգորիթմների արտադրողականությունների համեմատությունները GPU արագագործիչների վրա

Հ. Ասցատրյան և Է. Գիչունց

Ամփոփում

Հերմիտյան մատրիցի սեփական լուծումները գտնելիս շատ կարևոր խնդիր է հանդիսանում սիմետրիկ երեքանկյունագծային մատրիցի լուծումների ալգորիթմներից արդյունավետ տարբերակի որոշումը: Աշխատանքում համեմատվում են այս ալգորիթմները կոմպլեքս Հերմիտյան մատրիցների դեպքում հիբրիդային համակարգերում: Մեթոդները կիրառվել են Tesla C1060 և Tesla K40 GPU արագագործիչների վրա և ներկայացված են արտադրողականությունները ինչպես մեթոդների միջև, այնպես էլ արագագործիչների միջև:

Сравнение производительности алгоритмов вычисления собственных решений симметричных трехдиагональных матриц на графических процессорах GPU

Г. Асцатрян и Э. Гичунц

Аннотация

При нахождении решений собственных значений и векторов эрмитовых матриц большую важность представляет проблема определения эффективного варианта из алгоритмов нахождения решений симметричных трехдиагональных матриц. В данной работе эти алгоритмы сравниваются для случая комплексных эрмитовых матриц в гибридных системах. Методы были применены на графических процессорах TeslaC1060 и TeslaK40 и представлены прозводительности как для методов, так и между графическими процессорами.