Mathematical Problems of Computer Science 61, 62–69, 2024.

doi: 10.51408/1963-0115

UDC 004.4

Performance of Linear Algebra Factorization in Multi-Accelerator Architectures

Edita E. Gichunts

Institute for Informatics and Automation Problems of NAS RA, Yerevan, Armenia
e-mail: editagich@iiap.sci.am

Abstract

Hardware and software are required to effectively solve problems in many domains. The idea of creating a hybrid architecture based on graphics processors arose to meet the increasing demands of modern scientific problems. Most of these problems are reduced to solving linear algebra problems. A set of efficient linear solutions has been successfully used to solve important scientific problems for many years. Factorizations play a crucial role in solving linear algebra problems.

This work presents implementations of LU, QR and Cholesky factorizations on two graphics processors using the MAGMA 2.6.0 library. Their performances are given for matrices with real and complex numbers in single and double precision.

Keywords: MAGMA, multiple GPU, Linear Algebra, Factorizations.

Article info: Received 29 February 2024; sent for review 19 March 2024; accepted 16 May 2024.

1. Introduction

Many of the most important scientific programs rely on high-performance algorithms and linear algebra technologies, highlighting their importance and widespread impact from national security to medical breakthroughs. In the current high-performance computing (HPC) environment, parallelization is crucial. With the increasing utilization of video cards worldwide, the development of GPU parallel computing is expected to greatly affect the field of high-performance computing. These possibilities have already generated a great deal of interest in scientific as well as non-scientific circles. After all, the acceleration potential of good parallelization of algorithms is not always tens of times faster. The trend in multi-computing is clearly moving towards parallel algorithms, with most new solutions and initiatives focused in this direction. The current generation of GPUs has a fairly flexible architecture, which, along with high-level programming

languages and hardware-software architectures, reveals these capabilities and makes them more accessible. In the field of high-performance computing, the hottest topic is GPU-based hybrid systems. Hybrid architecture combines the advantages of shared and distributed memory systems. In such architectures, the GPU device is used as a coprocessor or accelerator to handle multi-computation applications. Calculations on GPU are known for being developed and processed very quickly. One of the leading video chip manufacturers, Nvidia, has introduced the CUDA [1] (Compute Unified Device Architecture) platform. CUDA is both a software and hardware technology available to every developer. It is an extension of the C programming language. The only requirement is the use of different programming paradigms typical for parallel computing.

Linear algebra faces a significant challenge in terms of computational efficiency, which has led to the development of software libraries following advancements in computer architecture. In the mid-1960s, IBM released the Scientific Subroutine Package [2], a collection of FORTRAN subroutines optimized for the IBM System/360 machine. In 1974, Harwood released EISPACK [3], a package of FORTRAN routines that compute the eigenvalues and eigenvectors of a matrix. BLAS's basic linear algebra routines were the first product of a joint project with ACMSIGNUM during 1973–1977 [4], which was based on a proposal made in 1973 [5].The LINPACK library was introduced in 1979 as a collection of routines for solving linear equations and linear least squares problems on supercomputers of the 1970s and 1980s, mostly based on vector processors [6].

LINPACK used the partial rotation engine of LU analysis to solve 100-dimensional problems, allowing the user to evaluate the performance of their memory and processors. The first version of BLAS (BLAS Level 1) implemented scalar-vector and vector-vector operations. BLAS2 (BLAS Level 2) was developed in 1988 as an extension of BLAS1 to exploit the capabilities of vector processors [7, 8]. BLAS2 offers the ability to perform matrix-vector operations. LAPACK [9], released in 1992, replaces LINPACK and EISPACK and provides better performance. LAPACK specializes in solving systems of linear equations, linear least squares problems, eigenvalue problems, and singularity problems. To perform these operations, related calculations are also performed, such as matrix analysis: LU, QR (Q-matrix is unitary or Hermitian, and R is upper trapezoidal), Cholesky, etc.

For GPUs, NVIDIA offers CuBLAS [10], an implementation of BLAS in the NVIDIA CUDA and EM Photonics environments, as well as their CULA solutions [11] as implementations of LAPACK CUDA.

MAGMA [12] is an extension of LAPACK in a hybrid framework. It includes an amazing variety of subroutines for solving linear algebra problems.

MAGMA's research is based on the idea that optimal software solutions for solving complex problems in a hybrid environment should be self-hybridizing, combining the strengths of various algorithms within a single framework. Based on this idea, efforts are being made to develop algorithms for hybrid multi-core and graphics systems. Designed with LAPACK functionality, data storage and interface capabilities, the MAGMA library makes it easy for scientists to port their software components from LAPACK to MAGMA and take advantage of the new hybrid architecture.

LU, QR and Cholesky factorizations play an important role in linear algebra. LU factorization is applied to the problem of finding solutions to a system of linear equations. The first step is to perform the LU factorization of the matrix, and then solutions can be obtained. It is worth mentioning that the paper referenced as [13] presents solutions to a system of linear equations using the types of LU factorization and random butterfly transformation implemented with the MAGMA library on a single graphics processor.

QR factorization is often used to solve the linear least squares (LLS) problem. It is also the basis of the QR algorithm [14,15] for finding the eigenvalue problem.

Cholesky factorization is useful for efficient numerical solutions such as Monte Carlo simulations.
This paper presents implementations of LU, QR, and Cholesky factorizations of widely used linear algebra problems on multiple graphics processor architectures, using the MAGMA 2.6.0 library. Performances of the specified factorizations for matrices with real and complex numbers in both single and double precision are presented.

2. Stages of Implementing Factorizations with Multiple Accelerators

It was observed that when dealing with multiple accelerators, LU, QR, and Cholesky factorizations were implemented by using the MAGMA 2.6.0 library. We have provided descriptions of these factorization subroutines. It should be noted that instead of using types, the letter x was used, which in the case of real numbers is s, and it is d for single and double precision, respectively, while in the case of complex numbers, c and z are used.
*magma_xgetrf_mgpu(ngpu, M, N, d_A, ldda, ipiv, &info)*computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.
The factorization has the form

A = P * L * U,

where P is a permutation matrix, L is a lower triangular with unit diagonal elements (lower trapezoidal if *M>N*), and U is an upper triangular (upper trapezoidal if *M<N*).

*magma_xgeqrf2_mgpu(ngpu, M, N, d_A, ldda, tau, &info)*computes a QR factorization of an M-by-N matrix A.
The factorization has the form

A = Q * R.

*magma_xpotrf_mgpu(ngpu, uplo, N, d_A, ldda, &info)*computes the Cholesky factorization of a real symmetric and complex Hermitian positive definite matrix A.
The factorization has the form

A = U**H * U, if UPLO = MagmaUpper, or
A = L * L**H, if UPLO = MagmaLower,

where U is an upper triangular matrix and L is a lower triangular.
uplo= MagmaUpper: Upper triangle of A is stored,
uplo= MagmaLower: Lower triangle of A is stored.

Here are the stages of implementing Cholesky factorization when using multiple accelerators:

1. First, the MAGMA library is initialized using the magma_init() function.
2. Memory is allocated for the matrix on the CPU using the function magma_xmalloc_cpu(&h_A, lda*N). Memory is also allocated for the matrix copy on the CPU using magma_xmalloc_pinned(&h_R, lda*N).
3. To allocate memory for the matrix on GPUs, we cycle from GPU to GPU, and in each of them, the function magma_setdevice(dev) is first called, then the memory is allocated using the function magma_xmalloc(&d_A[dev], max_size), where max_size = (1 +N/(nb*ngpu))*nb * magma_roundup(N, nb) and nb= magma_get_dpotrf_nb(N).
4. The matrix is generated using the function magma_generate_matrix(opts, N, N, h_A, lda).

5. We copy the matrix using the lapackf77_xlacpy(MagmaFullStr, &N, &N, h_A, &lda, h_R, &lda) function, which will be sent to the GPU memory.
6. The function magma_xsetmatrix_1D_col_bcycle(N, N, h_R, lda, d_A, ldda, ngpu, nb) transfers the matrix to the GPU memory.
7. We fix the time using the function gpu_time = magma_wtime().
8. The function magma_xpotrf_mgpu(ngpu, uplo, N, d_A, ldda, &info) is called, which performs Cholesky factorization in parallel on GPUs.
9. Using the difference gpu_time=magma_wtime()-gpu_time, we obtain the calculation execution time.
10. After the calculations are completed, the function magma_xgetmatrix_1D_col_bcycl (N, N, d_A, ldda, h_R, lda, ngpu, nb) transfers the results from the GPUs to the CPU memory.
11. We clear the allocated memories on the CPU using the functions magma_free_cpu(h_A) and magma_free_pinned(h_R), and clear the allocated memories on the GPUs by performing a cycle transfer from GPU to GPU, first calling magma_setdevice(dev) and then magma_free(d_A[dev]) functions.
12. At the end of the program, we use magma_finalize() to terminate MAGMA.

3. Experimental Results

Tests were conducted on two NVIDIA Tesla V100-PCIE graphics processors. The cuda-10.2 platform was utilized for parallel computing. To install the MAGMA 2.6.0 library, the BLAS, LaPack, cLaPack and ATLAS libraries were installed. To install the MAGMA library, the gcc, g++, nvcc, and gfortran compilers were used. To compile MAGMA, the following static (.a) and dynamic (.so) libraries are also required: libgfortran.a, libf77blas.a, libcblas.a, libf2c.a, libm.a, libstdc++.a, libpthread.a, libdl. .a, libcublas.so, libcudart.so, libcusparse.so, libcudadevrt.a.

Let us present the results of experiments in the form of graphs.

Figures 1 and 2 display graphs of LU factorization for matrices with real and complex numbers in single and double precision, respectively.

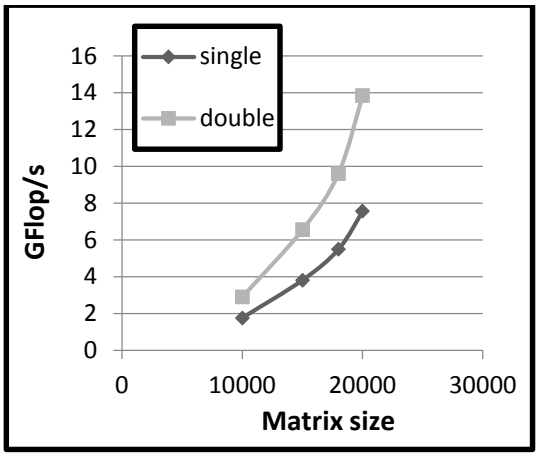

Fig.1. LU real

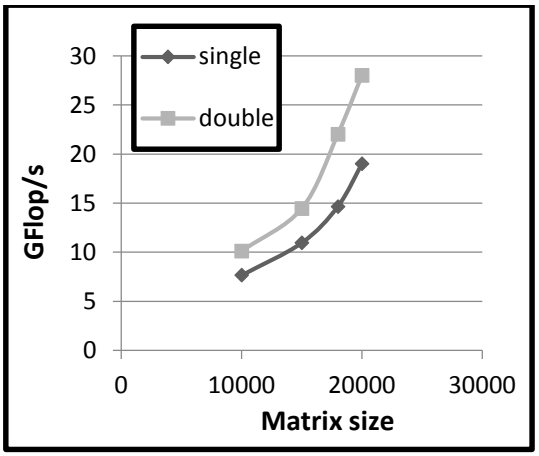

Fig. 2. LU complex

Figures 3 and 4 display graphs of QR factorization for matrices with real and complex numbers in single and double precision, respectively.

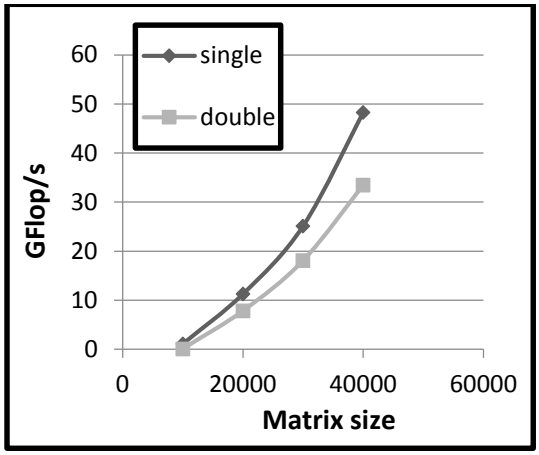

Fig. 3. QR real

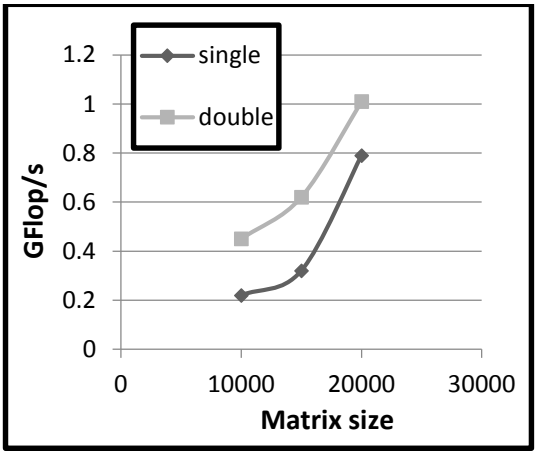

Fig. 4. QR complex

Figures 5 and 6 display Cholesky factorization graphs for matrices with real and complex numbers in single and double precision, respectively.

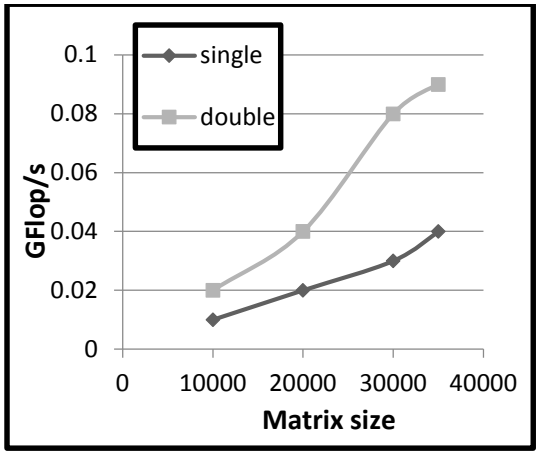

Fig. 5. Cholesky real

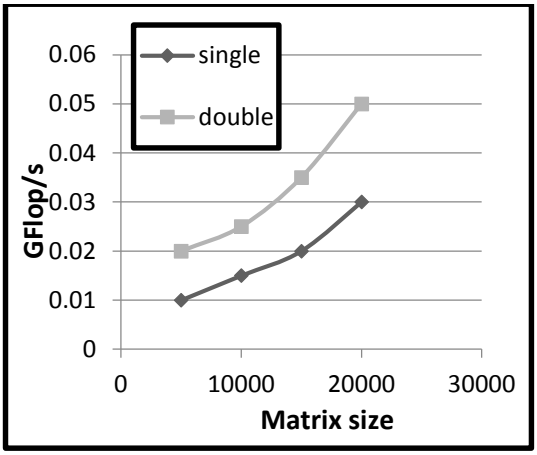

Fig. 6. Cholesky complex

4. Conclusion

We have reached the following conclusions based on the results of our experiments:

For matrices with real numbers, single precision performance in the case of LU factorization is 2 times lower than double precision. For matrices with complex numbers, single precision performance is 1.5 times lower than double precision.

In the case of QR factorization, single-precision performance for matrices with real numbers is 1.5 times higher than binary precision, and for complex numbers, single precision performance is 2 times lower than double precision.

In the case of Cholesky factorization and for matrices with real and complex numbers, single precision performance is 2 times lower than double precision performance.

References

[1] NVIDIA, “NVIDIA CUDA Parallel Computing Platform”. http://www.nvidia.com/object/cuda_home_new.html, NVIDIA, 2013.

[2] International Business Machines Corporation. System/360 Scientific Subroutine Package (360A-CM-03X) Version II, Programmer’s Manual. IBM Technical Publications Department, White Plains, NY, 1967.

[3] B. S. Garbow. EISPACK-a package of matrix eigensystem routines. Computer Physics Communications, 7(4):179–184, 1974.

[4] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. ACM Trans. Math. Softw., 5(3):308–323, September 1979.

[5] R. J. Hanson, F. T. Krogh, and C. L. Lawson. A proposal for standard linear algebra subprograms. ACM Signum Newsletter, 1973.

[6] J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. LINPACK Users’ Guide, volume 8. SIAM, 1979.

[7] J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. Algorithm 656: an extended set of basic linear algebra subprograms: model implementation and test programs. ACM Transactions on Mathematical Software (TOMS), 14(1):18–32, 1988.

[8] Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An Extended Set of FORTRAN Basic Linear Algebra Subprograms. ACM Trans. Math. Softw., 14(1):1–17, March 1988.

[9] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. LAPACK Users’ Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[10] CUDA Nvidia. Cublas library. NVIDIA Corporation, Santa Clara, California, 15, 2008.

[11] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis. CULA: hybrid GPU accelerated linear algebra routines. In SPIE Defense, Security, and Sensing, pages 770502–770502. International Society for Optics and Photonics, 2010.

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

[13] H. V. Astsatryan, E. E. Gichunts, “Performances of Methods for Solving a Linear System of Equations in the Architecture of GPU Accelerator”, Transactions of IIAP NAS RA, Mathematical Problems of Computer Science, vol. 45, pp. 44—52, 2016.

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

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

Գծային հանրահաշվի ֆակտորիզացիաների արտադրողականությունները բազմաքանակ արագացուցիչների ճարտարապետությունում

Էդիտա Ե. Գիչունց
ՀՀ ԳԱԱ Ինֆորմատիկայի և ավտոմատացման պրոբլեմների ինստիտուտ, Երևան, Հայաստան
e-mail: editagich@iiap.sci.am

Ամփոփում

Բազմաթիվ բնագավառների խնդիրների արդյունավետ լուծումների համար պահանջվում է ապարատային և ծրագրային ապահովում: Հիբրիդային ճարտարապետության ստեղծման գաղափարը, որը հիմնված է գրաֆիկական պրոցեսորների վրա, ծագել է ժամանակակից գիտական հիմնախնդիրների աճող պահանջների բավարարման պատճառով: Այդ խնդիրների մեծ մասը բերվում է գծային հանրահաշվի խնդիրների լուծումներին: Արդյունավետ գծային լուծումների հավաքածուն շատ երկար տարիներ կարողանում է լուծել կարևորագույն գիտական խնդիրներ: Գծային հանրահաշվի խնդիրների լուծումներում կարևորագույն դերակատարում ունեն ֆակտորիզացիաները:
Այս աշխատանքում ներկայացված են LU, QR և Խոլեցկու (Cholesky) ֆակտորիզացիաների իրականացումները երկու գրաֆիկական պրոցեսորների վրա՝MAGMA 2.6.0 գրադարանի կիրառմամբ: Տրված են նրանց արտադրողականությունները իրական և կոմպլեքս թվերով մատրիցների համար՝մեկական և երկուական ճշգրտություններում:

Բանալի բառեր՝ MAGMA, բազմակի GPU, գծային հանրահաշիվ, ֆակտորիզացիա:

Производительность факторизации линейной алгебры в мультиускорительных архитектурах

Эдита Е. Гичунц

Институт проблем информатики и автоматизации НАН РА, Ереван, Армения
[2] Сименс Индастри Софтвер, Ереван, Армения
e-mail: editagich@iiap.sci.am

Аннотация

Для эффективного решения проблем во многих областях требуется аппаратное и программное обеспечение. Идея создания гибридной архитектуры на базе графических процессоров возникла для удовлетворения растущих требований современных научных задач. Большинство этих задач сводятся к решению задач линейной алгебры. Набор эффективных линейных решений уже много лет успешно используется для решения

важных научных задач. Факторизации играют решающую роль в решении задач линейной алгебры.

В этой работе представлены реализации факторизации LU, QR и Холецкого на двух графических процессорах с использованием библиотеки MAGMA 2.6.0. Приведены их производительности для матриц с действительными и комплексными числами одинарной и двоичной точности.

Ключевые слова` МАГМА, GPU, линейная алгебра, факторизация.