Mathematical Problems of Computer Science 65, 63–71, 2026.

doi:10.51408/1963-0146

UDC 004.4

Matrix-Vector Multiplication Performances 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

This paper presents the performance of symmetric and Hermitian matrix-vector multiplication on two Volta 100 graphics processors in single and double precision. The implementations were performed using the Magma library.

The goal of this work is to present the implementations and performance evaluations of symmetric and Hermitian matrix-vector multiplication on 2 GPUs, and to compare their performance with that of a 1-GPU implementation.

Keywords: Multiple GPUs, MAGMA, Matrix-Vector Multiplication.

Article info: Received 24 Murch 2026; sent for review 9 April 2026; receieved in revised from 30 April 2026; accepted 8 May 2026.

1. Introduction

For high-performance computing, architectural solutions are being developed that facilitate work in this environment. In recent years, general-purpose graphics processors have emerged as a prominent topic in high-performance computing. Calculations on GPUs have been and are being developed very quickly, thanks to the CUDA (Compute Unified Device Architecture) [1] platform developed by Nvidia. CUDA is a software-hardware technology, based on the C programming language, along with its compiler and libraries, which is available to all developers.

The popularity of hybrid GPU-based systems began with the introduction of the NVIDIA CUDA architecture and the extension of the standard programming languages C, C++, and Fortran, which simplified GPU programming, allowing developers to leverage the computational power of modern GPUs. The hybrid architecture combines the advantages of shared- and distributed-memory architectures.

The development of computer architecture was followed by software libraries. Computational efficiency in linear algebra is a significant challenge, making optimized program implementations necessary. Linear algebra libraries are crucial in addressing this issue.

In the mid-1960s, IBM released the Scientific Routine Package [2], a set of FORTRAN routines. In 1974, Harwood published EISPACK [3], a FORTRAN routine package designed to

compute the eigenvalues and eigenvectors of matrices. Additionally, BLAS (Basic Linear Algebra Subroutines), the first product of the ACMSIGNUM joint project, was developed during the 1973-1977 period [4]. The LINPACK library was introduced in 1979 as a collection of subroutines specifically designed for the supercomputers of the 1970s and 1980s, particularly those employing vector processors, primarily for solving linear equations and linear least-squares problems. LINPACK (HPL) [6, 7] also enables evaluation of the most powerful supercomputers, as ranked by the TOP500 [8]. The initial version of BLAS (BLAS Level 1) implemented scalar-vector and vector-vector operations. In 1988, BLAS2 (BLAS Level 2) was developed as an extension of BLAS1 to exploit vector processors' capabilities [9, 10]. BLAS2 enables matrix-vector operations.

In 1990, another extension was added to BLAS3 [11, 12]. This extension accounted for advancements in computer memory by implementing matrix-matrix operations.

LAPACK [13], released in 1992, replaced LINPACK and EISPACK, providing better performance. LAPACK focuses on solving systems of linear equations, linear least squares problems, eigenvalue problems, and uniqueness problems. To perform these operations, it also carries out related calculations such as matrix analyses (LU, QR, LDLT, Cholesky, etc.).

For GPUs, NVIDIA offers CuBLAS [14], an implementation of BLAS in NVIDIA CUDA that encompasses all three levels.

The MAGMA [15] project aims to develop a linear algebra library, similar to LAPACK, for heterogeneous/hybrid architectures, starting with modern Multicore + GPU systems. MAGMA also includes MAGMA BLAS, which complements the CUBLAS subroutines.

For linear algebra problems in multi-GPU architectures, the cuBlasXt library [16] and MAGMA library subroutines are used, designed for multi-GPU systems. It should be noted that the cuBlasXt library includes only the BLAS3 level, i.e., matrix-matrix operations. Therefore, in this work, we use the MAGMA library, which also includes several subroutines for multi-GPU systems.

This paper outlines the algorithmic steps for implementing symmetric and Hermitian matrix-vector multiplication on two graphics processors. It also provides performance evaluations for matrices and vectors with dimensions of up to 40,000. Additionally, the paper presents estimates of the performance differences when using a single graphics processor.

The obtained results expand the possibilities of applying GPU computing technologies to high-performance computing tasks and can be used in the development of new parallel algorithms and engineering computations.

The results of this work can be applied:

- In high-performance computing (HPC) systems;
- In the development of software for scientific simulation;
- In artificial intelligence and machine learning tasks;
- In engineering and physical computations;
- In big data processing;
- In supercomputing systems and computational complexes.

2. Steps to Implement a Matrix-Vector Multiplication in Multiple-GPU Architecture

Linear algebra problems are crucial in high-performance computing, with matrix-vector multiplication being one of the most widely used operations. High productivity in these computing systems is often achieved through the use of linear algebra libraries. It is important to note that the

matrix-vector multiplication is classified as a Level 2 subroutine in the BLAS library, which is not included in cuBlasXt. Consequently, in multi-accelerator architectures, the magmablas_xmv_mgpu() subroutines from the MAGMA magmablas library are utilized for performing the matrix-vector multiplication. Additionally, when referring to symmetric or Hermitian matrices, specific abbreviations are indicated instead of 'x.'
The objective of this work is to implement this multiplication across multiple GPUs. When these subroutines are called, the matrix-vector multiplication is performed in parallel across all GPUs simultaneously. The model used in this work describes how the matrix and vector are distributed across each GPU.
Since the result of the matrix-vector multiplication is a vector, for example, the first element is obtained from the multiplication of the first row of the matrix and the vector, the second element is obtained from the multiplication of the second row of the matrix and the vector, and so forth, then the parallelization model used in this problem is as follows: the matrix is divided into as many parts as there are GPUs, and each GPU receives the divided part of the matrix A along with the entire multiplication vector. In this model, each graphics processing unit (GPU) generates a vector element as many times as there are rows in the partitioned matrix.
Below are the steps for implementing a matrix-vector multiplication in a hybrid system with multiple accelerators, indicating the main subroutines involved.

Here, we outline the algorithmic steps for implementing single-precision Hermitian matrix-vector multiplication.

1. We include the following necessary header files:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <cuda.h>
#include <cuda_runtime_api.h>
#include <magma.h>
#include <magma_v2.h>
#include "magma_lapack.h"
#include "flops.h"
#include "magma_cbulge.h"
#include "magma_threadsetting.h"
#include "magma_operators.h"
#include <magma_internal.h>
#include "magma_timer.h"
```

2. The MAGMA library is initialized:

```
magma_init().
```

3. Memory is allocated on the CPU for the matrix and vectors. Additionally, memory is allocated for the final result vector transferred from the GPU to the CPU, where hwork serves as an extra workspace on the CPU allocated with the lhwork dimension:

```
magma_cmalloc_cpu(&A,  matsize );
magma_cmalloc_pinned(&X, vecsize );
magma_cmalloc_cpu(&Y,  vecsize );
magma_cmalloc_cpu(&Ymagma, vecsize );
magma_cmalloc_pinned(&hwork, lhwork ),
```

where hwork serves as an extra workspace on the CPU allocated with the lhwork dimension.

4. On the GPU, memory is allocated for the transferred matrix, vector, and result vector:

```
magma_setdevice(opts.device );
```

magma_cmalloc(&dA, matsize);
magma_cmalloc(&dX, vecsize);
magma_cmalloc(&dY, vecsize).

On GPUs, local memory is allocated to the partitioned sections of the matrix, moving cyclically from one GPU to another. In each instance, the magma_setdevice(dev) function is called first, followed by the memory allocation functions:

magma_cmalloc(&d_lA[dev], ldda*n_local[dev]);

magma_cmalloc(&dwork[dev], ldwork), where dwork is an additional workspace on the GPU with dimension ldwork.

Note that n_local = ((n / nb) / ngpu + 1) * nb.

5. Since the result of the matrix-vector multiplication is a vector, the first element of which, for example, is obtained from the product of the first row of the matrix and the vector, the parallelization algorithm is as follows: the matrix is divided into as many parts as there are GPUs, and each GPU is sent the divided part of the matrix A. The transfer of the matrix A is performed using the following function, which sends the matrix A from the CPU memory to the GPU dA, which is distributed cyclically across multiple GPUs into 1D row blocks.

magma_csetmatrix_1D_col_bcyclic(Noffset, Noffset, A, lda, d_lA, ldda, opts.ngpu, nb), where

Noffset = N + offset;
offset = min(n,nb).

6. We fix the initial time using the gpu_time = magma_sync_wtime(0) function.

7. The subroutine

magmablas_chemv_mgpu(opts.uplo, N, alpha, d_lA, ldda, offset, X + offset, incx,

beta, Ymagma + offset, incx, hwork, lhwork, dwork, ldwork, opts.ngpu, nb, queues) is called.

Note that in the program, we have previously introduced all the required parameters to be included in the subroutine.

This subroutine calculates the operation y = alpha*A*x + beta*y. In the case of alpha=1 and beta=0 values, we have the matrix-vector multiplication y = A*x.

8. Using the gpu_time = magma_sync_wtime(0) - gpu_time difference, we get the calculation execution time.

9. After completing the calculations, the results obtained from the GPUs are transferred to the CPU memory using the following function:

magma_cgetvector(Noffset, dY, incx, Ymagma, incx):

10. At the end of the program, the allocated memory on the CPU is cleared:

magma_free_cpu(A);
magma_free_cpu(Y);
magma_free_pinned(X);
magma_free_cpu(Ymagma);
magma_free_pinned(hwork).

11. We clear the allocated memory on GPUs by moving from one GPU to another in a loop, first calling magma_setdevice(dev) function, then magma_free(d_lA[dev]) and magma_free(dwork[dev]) functions.

We also clear the memory allocated for the moved vectors:

magma_free(dA);
magma_free(dX);
magma_free(dY).

12. We finalize MAGMA processing using the magma_finalize() function.

3. Experimental Results

The research was conducted on two NVIDIA Tesla V100-PCIE graphics processors using the Magma 2.6.0 library. To install the MAGMA 2.6.0 library, the BLAS, LaPack, cLaPack, ATLAS libraries, as well as the following static (.a), dynamic (.so) libraries were loaded: libgfortran.a, libf77blas.a, libcblas.a, libf2c.a, libm.a, libstdc++.a, libpthread.a, libdl.a, libcublas.so, libcudart.so, libcusparse.so, libcudadevrt.a. The gcc, g++, nvcc, and gfortran compilers were used to compile the MAGMA library.

It should also be noted that during the research, when applied to 2 GPUs, both symmetric and Hermitian matrix-vector multiplication were accessed with matrices and vectors of dimensions up to n=40000. And when applied to 1 GPU, in the case of symmetric matrices, n=30000 dimensions were accessed for single precision, and n=20000 dimensions were accessed for double precision. In the case of Hermitian matrices, n=20000 dimensions were accessed for both single and double precision.

Let us present the results from the experiments in the form of graphs and tables. The tables display the productivity values for the specified input matrix dimension.

Figures 1 and 2 show the performance graphs of the symmetric and Hermitian matrix-vector multiplication on two GPUs in single and double precision, respectively.

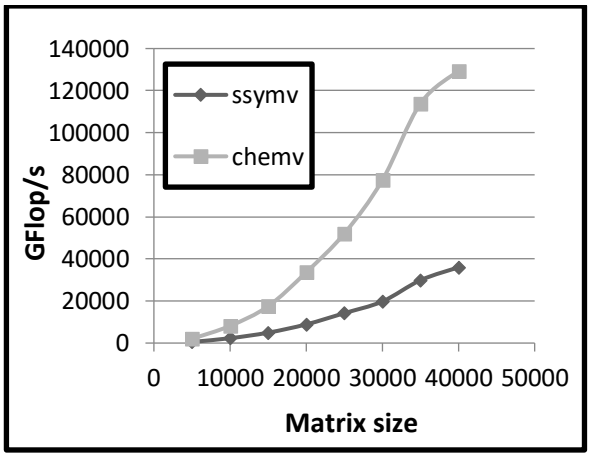

Fig. 1. Single presicion

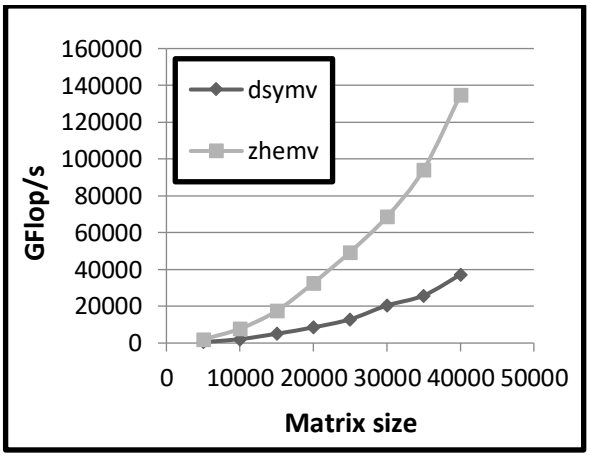

Fig. 2. Double presicion

x	ssymv	chemv
5000	531,03	1979,04
10000	2304,79	8165,32
15000	4941,26	17478,01
20000	8877,28	33641,05
25000	14208,92	52041,63
30000	19764,39	77437,18
35000	29873,08	113866,87
40000	35984,20	129371,34

x	dsymv	zhemv
5000	556,38	1979,04
10000	2107,90	7768,39
15000	5060,49	17478,01
20000	8516,78	32658,83
25000	12914,01	49464,10
30000	20405,40	68637,50
35000	25819,95	94279,68
40000	37180,36	134897,25

Figures 3 and 4 show graphs of the performance difference for symmetric matrix-vector multiplication when using one and two graphics processors in single and double precision, respectively.

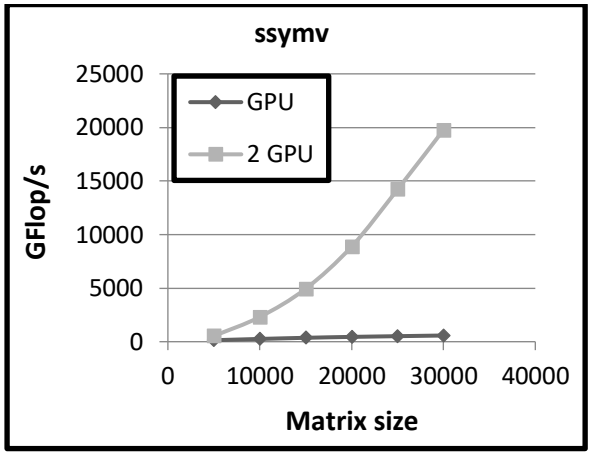

Fig. 3. Symmetrical Single precision

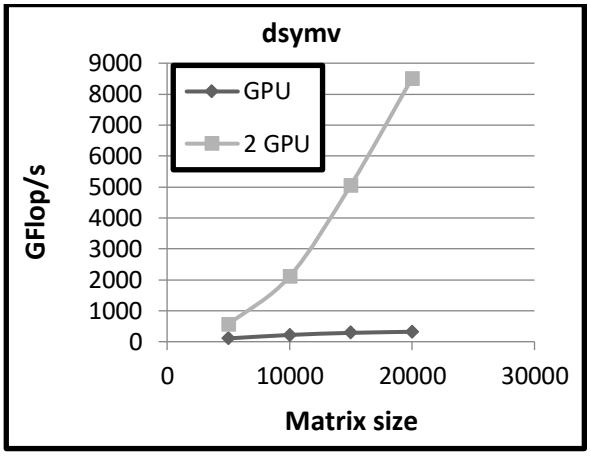

Fig. 4. Symmetrical Double precision

	ssymv	
x	GPU(GFlop/s)	2 GPU
5000	136,29	531,03
10000	256,40	2304,79
15000	362,92	4941,26
20000	448,97	8877,28
25000	514,63	14208,92
30000	577,70	19764,39

	dsymv	
x	GPU(GFlop/s)	2 GPU
5000	105,94	556,83
10000	217,40	2107,90
15000	285,73	5060,49
20000	319,52	8516,78

Figures 5 and 6 show the graphs of the difference in Hermitian matrix-vector multiplication performance when using one and two graphics processors in single and double precision, respectively.

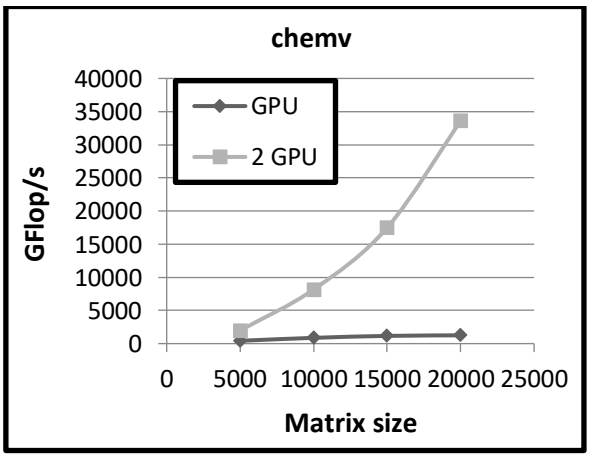

Fig. 5. Hermitian Single precision

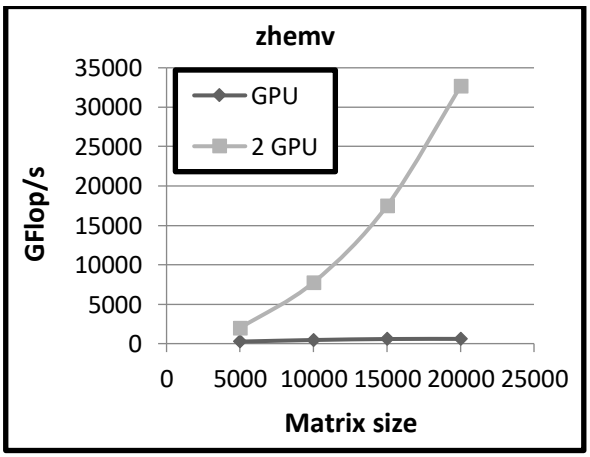

Fig. 6. Hermitian Double precision

	chemv	
x	GPU(GFlop/s)	2 GPU
5000	422,94	1979,04
10000	872,35	8165,32
15000	1163,58	17478,01
20000	1253,53	33641,05

	zhemv	
x	GPU(GFlop/s)	2 GPU
5000	258,51	1979,04
10000	459,84	7768,39
15000	584,09	17478,01
20000	815,45	32658,83

4. Conclusion

As a result of the experiments, we have obtained the following results:

- On two GPUs, in both single and double precision, the performance of the Hermitian matrix-vector multiplication is 3.5 times higher than the performance of the symmetric matrix-vector multiplication.
- In single precision, the performance of the symmetric matrix-vector multiplication of up to 30,000 dimensions on two GPUs is at least 10 times and up to 30 times higher than the performance on a single GPU.
- In double precision, the performance of a symmetric matrix-vector multiplication of up to 20,000 dimensions on 2 GPUs is at least 10 times and at most 25 times higher than that of 1 GPU.
- In single precision, the performance of a Hermitian matrix-vector multiplication of up to 20,000 dimensions on 2 GPUs is at least 10 times and at most 25 times higher than that of 1 GPU.
- In double precision, the performance of a Hermitian matrix-vector multiplication of up to 20,000 dimensions on 2 GPUs is at least 10 times and at most 40 times higher than that of 1 GPU.

References

[1] NVIDIA, “NVIDIA CUDA Parallel Computing Platform”, [Online]. Available: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*, vol. 7, no. 4, pp. 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.*, vol. 5, no. 3, pp. 308–323, 1979.

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

[6] J. Dongarra and P. Luszczek, “Linpack benchmark”, *Encyclopedia of Parallel Computing*, pp. 1033–1036, 2011.

[7] J. Dongarra, P. Luszczek and A. Petitet, The LINPACK benchmark: Past, present, and future. Concurrency and Computation: Practice and Experience. Concurrency and Computation: Practice and Experience, 15:2003, 2003.
[8] TOP500 Supercomputer Site. http://www.top500.org.
[9] 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), vol.14, no. 1, pp. 18–32, 1988.
[10] J. Dongarra, J. Du Croz, S. Hammarling and R. J. Hanson, "An extended set of fortran basic linear algebra subprograms", *ACM Trans. Math. Softw.*, vol.14, no. 1, pp.1–17, 1988.
[11] J. Dongarra, J. Cruz, S. Hammerling and I. S. Duff, "Algorithm 679: A set of level 3 basic linear algebra subprograms: model implementation and test programs", ACM *Trans. Math. Softw.*, vol. 16, no. 1, pp. 18–28, 1990.
[12] J. Dongarra, J. Du Croz, S. Hammarling and I. S. Duff, "A set of level 3 basic linear algebra subprograms", *ACM Trans. Math. Softw.*, vol. 16, no.1, pp. 1–17, 1990.
[13] 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.
[14] CUDA Nvidia. Cublas library. NVIDIA Corporation, Santa Clara, California, 15, 2008.
[15] "MAGMA Matrix Algebra on GPU and Multicore Architectures", [Online]. Available: http://icl.cs.utk.edu/magma/, 2014.
[16] [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html#using-the-cublasxt-api.

Մատրից-վեկտոր արտադրյալի արտադրողականությունները բազմաքանակ արագացուցիչների ճարտարապետությունում

Է. Գիչունց

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

Ամփոփում

Այս աշխատանքում ներկայացված են սիմետրիկ և Հերմիտյան մատրից-վեկտոր արտադրյալի արտադրողականությունները երկու Volta 100 գրաֆիկական պրոցեսորների վրա՝ մեկական և երկուական ճշգրտություններում: Իրականացումները կատարվել են ՝ կիրառելով Magma գրադարանը:
Աշխատանքի նպատակն է ներկայացնել սիմետրիկ և Հերմիտյան մատրից-վեկտոր արտադրյալի իրականացումները և արտադրողականությունների գնահատականները 2

GPU-ների վրա, և ցույց տալ նաև արտադրողականության տարբերությունը 1 GPU-ի կիրառման նկատմամբ:
Բանալի բառեր՝ բազմակի GPU-ներ, MAGMA, մատրից-վեկտոր բազմապատկում:

Производительность матрично-векторных произведений в мультиускорительных архитектурах

Э. Гичунц
Институт проблем информатики и автоматизации НАН РА, Ереван, Армения
e-mail: editagich@iiap.sci.am

Аннотация

В данной статье представлены результаты оценки производительности симметричных и эрмитовых матрично-векторных произведений на двух графических процессорах Volta 100 в одинарной и бинарной точности. Реализации выполнены с использованием библиотеки Magma.

Цель данной работы — представить реализацию и оценку производительности симметричного и эрмитова матрично-векторного умножения на двух графических процессорах, а также показать разницу в производительности по сравнению с реализацией на одном графическом процессоре.
Ключевые слова: множественные GPU, MAGMA, матрично-векторное умножение