Performance of Linear Algebra Symmetric and Hermitian k and $2k$ Rank Update in Multi-Accelerator Architectures

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

***Abstract*—Symmetric and Hermitian rank-k and rank-2k updates are crucial in various linear algebra problems. This paper presents implementations of symmetric and Hermitian updates of rank-k and rank-2k on two Volta 100 graphics processors, tested in both single and double precision. The implementations were performed using the cublasXt and Magma libraries.**

The objective of this paper isto provide performance estimates for symmetric and Hermitian updates of rank-k and rank-2k on two GPUs and to determine which of the cublasXt or Magma libraries is more efficient when applied to these problems.

***Keywords*— Multiple GPUs, cuBLASXt, MAGMA.**

I. Introduction

In recent years, general-purpose graphics processors have emerged as one of the most widely used topics in high-performance computing. The popularity of hybrid GPU-based systems began with the advent of the NVIDIA CUDA (Compute Unified Device Architecture) architecture [1] and the extensions of standard programming languages such as C, C++, and Fortran.

In the hybrid system, linear algebra problems are addressed using the cuBlas [2] and MAGMA [3] libraries, which include matrix-vector and matrix-matrix operations, as well as various linear algebra problems, including factorizations, solutions of systems of linear equations, finding eigenvalues and vectors, and performing various transformations.

For solving linear algebra problems in multi-GPU architectures, the cuBlasXt [4] library from cuBlas and the MAGMA library subroutines, which are designed for multi-GPUs, are used.

The cuBLASXt API serves as a host interface and supports several graphics processors. The cuBLASXt API supports only the BLAS3 [5,6] subroutines, which consist of matrix-matrix operations, symmetric and Hermitian k-rank and 2k-rank update problems.

This paper outlines the algorithmic steps for implementing symmetric and Hermitian rank-k and rank-2k update problems on two GPUs using the cuBLASXt and MAGMA libraries. It includes performance graphs for these problems in both single and double precision on two GPUs using the Magma and cuBLASXt libraries, as well as performance comparison graphs for these two libraries. The goal is to find out which of these libraries is more efficient to use and under what circumstances.

II. Implementation Steps in a Multi-GPU Architecture

The implementations of symmetric and Hermitian k and 2k rank updates on 2 GPUs were performed using the cublasXt and Magma 2.6.0 libraries.

Below, we outline the algorithmic steps of these implementations for symmetric 2k rank updates in the cases of using these two libraries: cublasXt and Magma.

In case of using the cublasXt library:

1. We include the following header files:
   #include <cuda.h>
   #include <cuda_runtime_api.h>
   #include <cublas.h>
   #include <cublas_v2.h>
   #include <cublasXt.h>
2. The cuBLASXt API context is initialized:
   cublasXtHandle_t handle,
   cublasXtCreate(&handle).
3. We give the number of GPU devices used and their corresponding IDs. In the case of two GPUs, it will be:
   constintnDevices = 2,
   intdeviceId[nDevices] = {0, 1}.
4. We call the cublasXtDeviceSelect() function, which allows the user to specify the number of GPU devices and their corresponding IDs. This function creates a cuBLAS context for each GPU in the list:
   cublasXtDeviceSelect(handle, nDevices, deviceId).
5. Before calling the cublasXt function, we use the start-time function:
   cudaEventRecord(start, 0).

https://doi.org/10.51408/csit2025_20

6. We call the calculation subroutine:

cublasXtSsyr2k(handle, CUBLAS_FILL_MODE_LOWER, CUBLAS_OP_N, n, n, &alpha, a, n, b, n,&beta, c, n), where we have already introduced the necessary parameters to be included in the subroutine.

7. Once the subroutine concludes, the end time function is called:

cudaEventRecord(stop, 0).

8. To calculate the time elapsed between the start and end of the program, we use the following function:

cudaEventElapsedTime(&time_seconds, start, stop), which returns the first argument of the function.

9. After the program concludes, the cuBLASXt API context should be destroyed:

cublasXtDestroy (handle).

In case of using Magma library:

1. We include the following header files:

```
#include <cuda.h>
#include <cuda_runtime_api.h>
#include <magma.h>
#include <magma_v2.h>
#include "magma_lapack.h"
```

2. The MAGMA library is initialized:

magma_init().

3. Memory is allocated for matrices on the CPU:

```
magma_smalloc_cpu(&hC, lda*n );
magma_smalloc_cpu(&hR, lda*n );
magma_smalloc_cpu(&hA, lda*k );
magma_smalloc_cpu(&hB, lda*k ).
```

4. Local memory is allocated for the matrices distributed on the GPUs, moving from GPU to GPU in a cycle. In each, the magma_setdevice(dev) function is first called, then the memory allocation functions:

```
magma_smalloc(&dC[dev], ldda*nlocal );
magma_smalloc(&dA[dev], ldda*k*2 ):
```

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

5. The C matrix is moved to the GPU memory using the following function:

magma_ssetmatrix_1D_col_bcyclic(n, n, hC,lda, dC, ldda, opts.ngpu, nb).

6. The A and B matrices are transferred to the GPU memory by cycling between GPUs using the following functions:

```
magma_ssetmatrix( n, k, hA, lda, dA[dev], ldda ),
magma_ssetmatrix( n, k, hB, lda, dB[dev], ldda ).
```

7. Next, we fix the execution time using the

gpu_time =magma_wtime() function.

8. The

magmablas_ssyr2k_mgpu2(MagmaLower, MagmaNoTrans, n, k, alpha, dA, ldda, 0, dB, ldda, 0, beta, dC, ldda, offset, ngpu, nb, queues, nqueue)

subroutine is then called, which performs the symmetric 2k rank update in parallel across the GPUs.

Note that all the necessary parameters required for the subroutine have been defined in the program beforehand.

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

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

magma_sgetmatrix_1D_col_bcyclic(n, n, dC, ldda, hR, lda, opts.ngpu, nb).

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

```
magma_free_cpu(hC );
magma_free_cpu(hR );
magma_free_cpu(hA );
magma_free_cpu(hB ).
```

12. We free the memory allocated on the GPUs by moving from GPU to GPU in a loop, first calling the magma_setdevice(dev), then the magma_free(dC[dev]) and magma_free(dA[dev]) functions.

13. We terminate MAGMA using the magma_finalize() function.

III. Experimental Results

The research was conducted on two NVIDIA Tesla V100-PCIE graphics processors using the cublasXt and Magma libraries. The cuda-10.2 platform was used for parallel calculations. To install the MAGMA 2.6.0 library, the BLAS, LaPack, cLaPack, ATLAS libraries, and 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, gfortran compilers were used to compile the MAGMA library.

We will present the results of the experiments in the form of graphs.

Figures 1 and 2 illustrate the performance graphs of symmetric k and 2k rank updates on 2 GPUs using the Magma library, respectively.

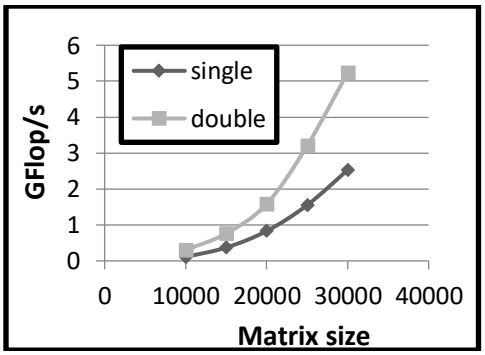

Fig. 1. k-rank

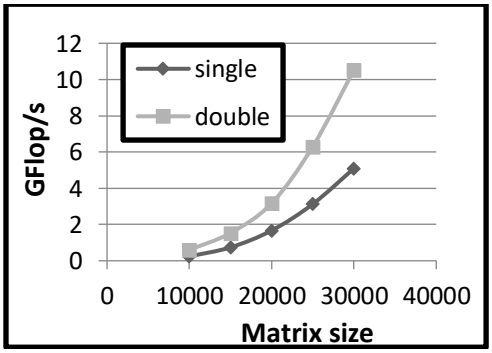

Fig. 2. 2k-rank

Figures 3 and 4 display the performance graphs of symmetric k and 2k rank updates on 2 GPUs using the cublasXt library, respectively.

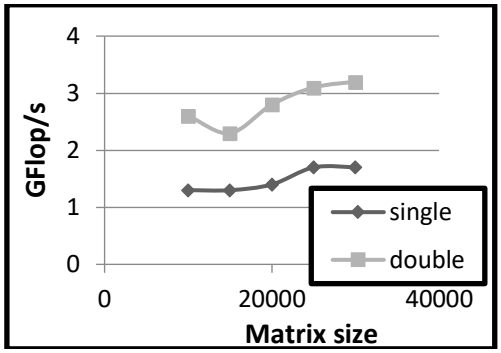

Fig. 3. k-rank

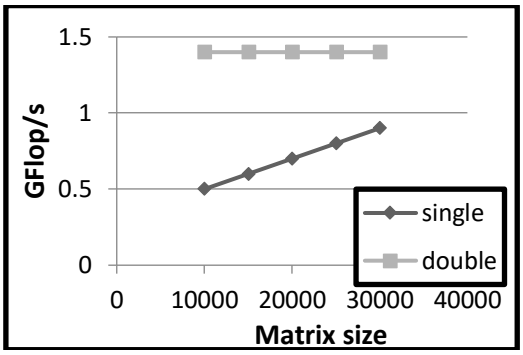

Fig. 4. 2k-rank

Figures 5 and 6 illustrate the performance difference graphs for symmetric k and 2k rank updates on 2 GPUs using the Magma library, presented in both single and binary precision, respectively.

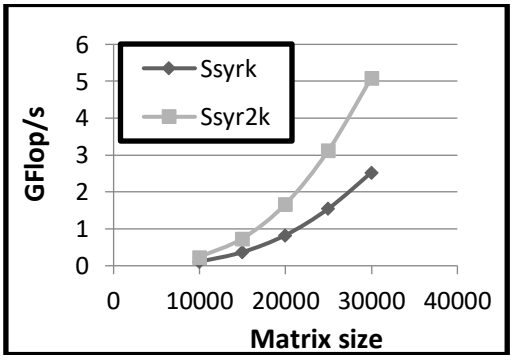

Fig. 5. Single presicion

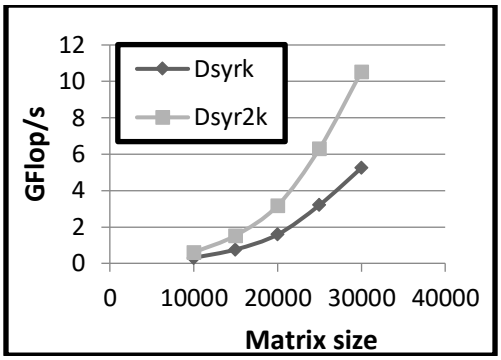

Fig. 6. Double presicion

Figures 7 and 8 display the performance difference graphs for Hermitian k and 2k rank updates on 2 GPUs using the Magma library in single and binary precision, respectively.

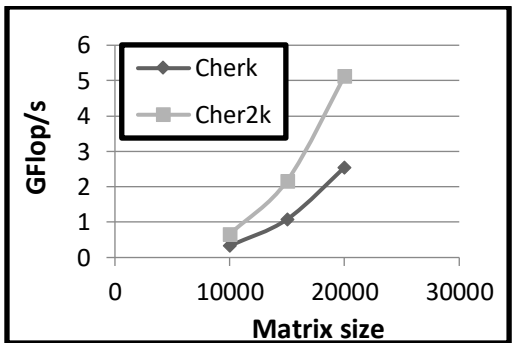

Fig. 7. Single complex presicion

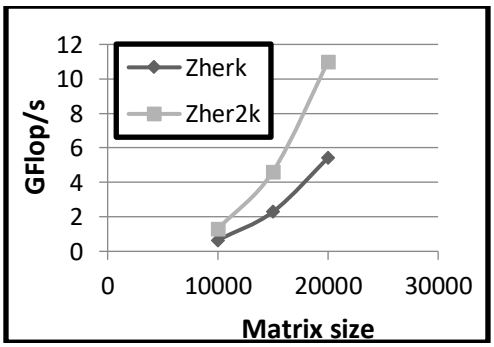

Fig. 8. Double complex presicion

Figures 9 and 10 provide the performance difference graphs for symmetric k rank updates on 2 GPUs using the Magma and cublasXt libraries in single and binary precision, respectively.

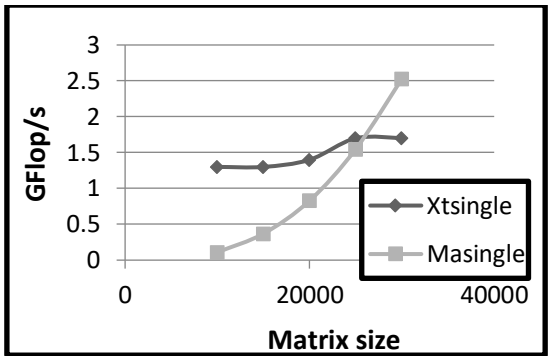

Fig. 9. Single k-rank

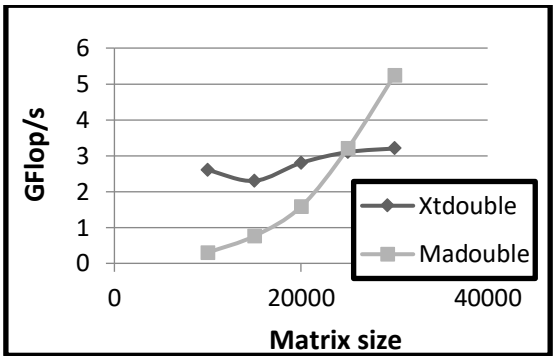

Fig. 10. Double k-rank

Figures 11 and 12 show the performance difference graphs for symmetric 2k rank update on 2 GPUs using the Magma and cublasXt libraries in both single and binary precision, respectively.

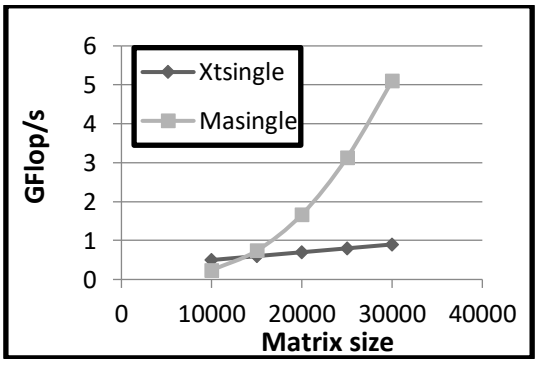

Fig. 11. Single 2k-rank

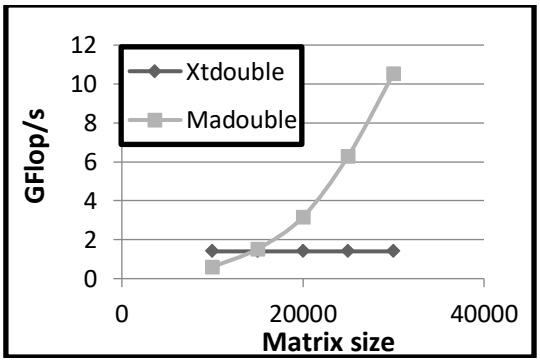

Fig. 12. Double 2k-rank

IV. Conclusion

As mentioned above, the implementations were performed on two Volta 100 GPUs using the MAGMA 2.6.0 library.

The following results were obtained from the experiments:

- When using the Magma library, the performance of binary precision for symmetric k-rank and 2k-rank updates is 2 times higher than the performance of single precision.
- When using the cublasXt library, the performance of binary precision for symmetric updates of rank k is 2 times higher than that of single precision, and for updates of rank 2k, the performance of binary precision is initially 2 times higher than that of single precision, and then it decreases to 1.5 times.
- When using the Magma library, the performance of symmetric updates of rank 2k for both single and binary precision is 2 times higher than that of k.
- In the case of Hermitian rank, the performance of rank 2k updates for single and double precision is 2 times higher than that of rank k updates.
- In the case of symmetric rank k updates in single and double precision, the cublasXt library is 2 times more efficient for matrices with dimensions up to 25000, and the Magma library proves to be much more efficient for matrices with larger dimensions.
- In the case of symmetric rank 2k updates in single and double precision, the cublasXt library is 1.5 times more efficient for matrices with dimensions up to 15000. Meanwhile, the Magma library is 2-5 times more efficient for matrices with larger dimensions.

References

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

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

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

[4] [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html#using-the-cublasxt-api.

[5] 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, March 1990.

[6] 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, March 1990.