Performance for General, Symmetric and Triangular Matrix Multiplication for Multiple GPUs

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

***Abstract*— The paper presents implementations of the product of general, symmetric and triangular type matrices on two graphics processors. Product performances of the mentioned matrices on one and two Volta 100 graphics processors with single and double precision are presented using the cuBLASXt library.**

***Keywords*—Graphics processors, matrix multiplication, cuBLASXt.**

I. INTRODUCTION

Along with the emergence of the hybrid system, libraries were created implementing the most important problems of linear algebra on the graphics processor. One of such libraries is cuBLAS [1], thanks to the subroutines of which the vector-vector, matrix-vector and matrix-matrix operations are performed in the CPU-GPU hybrid system. This paper [2] presents the efficiency of using the cuBLAS library for matrix multiplication.

cuBLASXt [3] API of cuBLAS is a Host interface that supports multiple graphics processors.The cuBLASXt API provides memory allocation between the mentioned GPUs as well as the workload distribution between them. The cuBLASXt API ensures only BLAS3 subroutines, that is, it provides only matrix-matrix operations. To share the load among multiple GPUs, the cuBLASXt API uses a tiling strategy, dividing each matrix into square tiles of BlockDim x BlockDim dimension. To compute the first brick of the resulting matrix, the CPU thread 0 in charge of GPU0, loads the three bricks of the first row of the first matrix and the three bricks of the first column of the second matrix to perform the memory transfer and calculations.

II. MATRIX MULTIPLICATION PROCESS USING CUBLASXT

Let us present the algorithmic steps of the matrix multiplication process on two GPUs in the case of symmetric matrices. It consists of the following steps:

1. *We include cuBLASXt's own cublasXt.h header file:*
#include <cublasXt.h>

2. *The cuBLASXt API context should be initialized using cublasXtCreate(). The cublasXtHandle_t type is a pointer type to an opaque structure holding the cuBLASXt API context:*
cublasXtHandle_t handle,
cublasXtCreate(&handle).

3. *The cublasXtDeviceSelect() function allows the user to specify the number of GPU devices and their corresponding IDs. With two GPUs it will be:*
const int nDevices = 2,
int deviceId[nDevices] = {0, 1},
cublasXtDeviceSelect(handle, nDevices, deviceId).

4. *The start time function is applied before the calculation:*
cudaEventRecord(start, 0).
The calculation is performed using the following subprogram:
cublasXtSsymm(handle,CUBLAS_SIDE_LEFT,
CUBLAS_FILL_MODE_LOWER , n, n, &alpha, a, n, b, n, &beta, c, n).
After the subroutine is finished, the end time function is applied:
cudaEventRecord(stop, 0).
cudaEventElapsedTime(&time_seconds, start, stop): the first argument to this function returns the passed time between the start and end of the program.

5. *At the end of the program, the context should be destroyed with*
cublasXtDestroy(),
cublasXtDestroy(handle).

III. EXPERIMENTAL RESULTS

The tests were carried out on two Volta 100 GPUs in cuda-10.2 environment. The product of general, symmetric and triangular matrices with single and double precision is implemented, and performance estimates are given on one

and two GPUs. Graphical images of the obtained results are as follows:

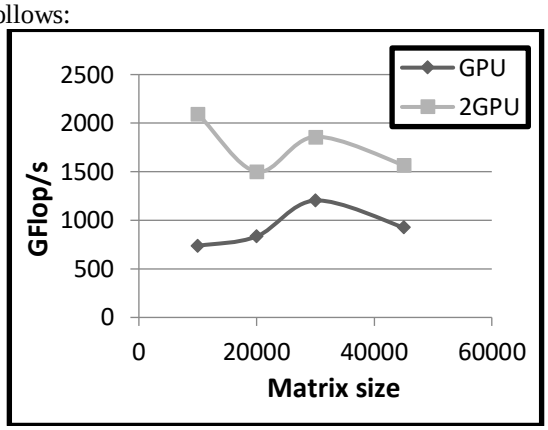

Fig. 1. Sgemm

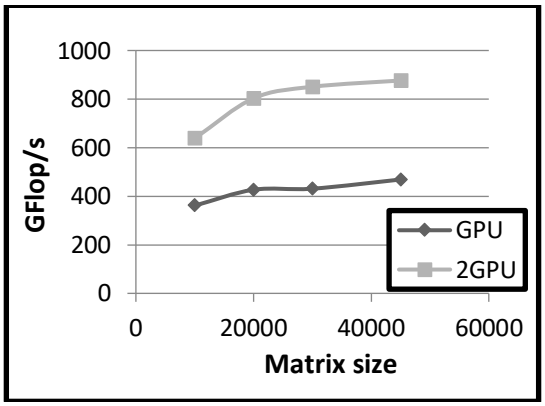

Fig. 2. Dgemm

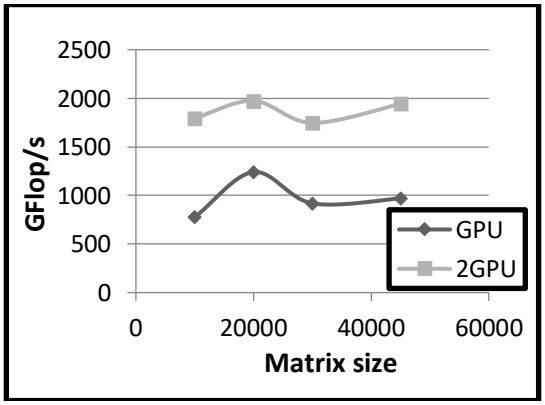

Fig. 3. Ssymm

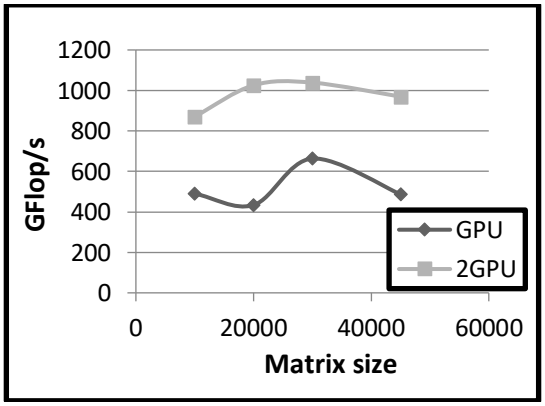

Fig. 4. Dsymm.

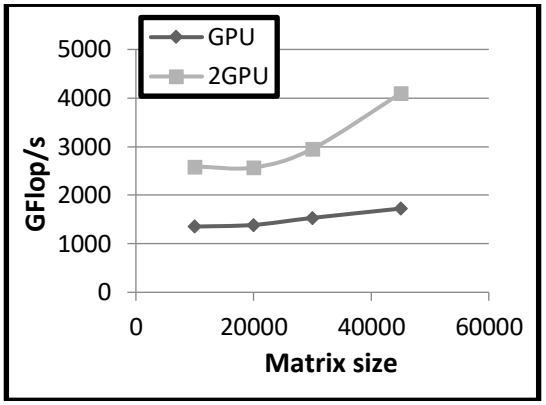

Fig. 5. Strmm.

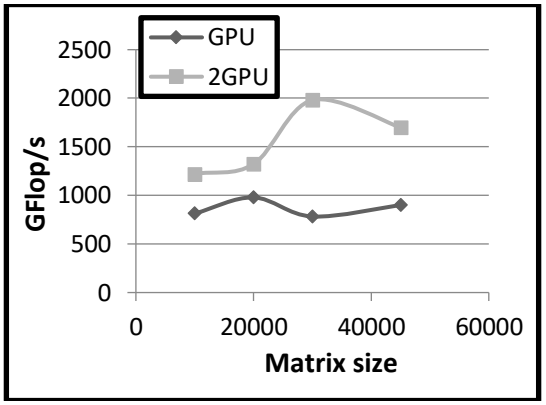

Fig. 6. Dtrmm

IV. Conclusion

Studies were carried out on matrices with 45000*45000 dimension, and we got the following results:

- In the case of general matrices, both for single and double precisions, performance on two GPUs is twice as high as on one GPU.
- For symmetrical matrices, both for single and double precisions, performance on two GPUs is twice higher than that on a single GPU.
- In the case of triangular matrices with single precision, performance on two GPUs is twice higher than on a single GPU, whereas with double precision - 1.5-2 times.
- On both GPUs, the performance of the product of general, symmetric and triangular matrices for single precision is twice higher than that for double precision.

References

[1] [Online]. Available: http://docs.nvidia.com/cuda/cublas/index
[2] E. E. Gichunts, "Benchmarking of GPU NVIDIA CUDA, CUBLAS and MAGMA Libraries Based on Matrix Multiplication Problem", *Transactions of IIAP NAS RA, Mathematical Problems of Computer Science*, vol. 42, pp. 121—126, 2014.
[3] [Online]. Available: https://docs.nvidia.com/cuda/cublas/index.html#using-the-cublasxt-api