Multi-GPU GEMM Algorithm Performance Analysis for Nvidia and AMD GPUs Connected by NVLink and PCIe

Yea Rem Choi[1](✉) and Vladimir Stegailov[1,2,3]

[1] HSE University, Moscow, Russia
echoj@hse.ru
[2] Joint Institute for High Temperatures of RAS, Moscow, Russia
[3] Moscow Institute of Physics and Technology, Dolgoprudny, Russia

Abstract. Modern types of multi-GPU servers combine up to 8 A100 GPUs connected by NVLink 3.0 links through NVSwitch. This connectivity provides unprecedented capabilities for multi-GPU algorithms. In this work, we analyze the performance of matrix-matrix multiplication algorithm developed by us previously. Tuning principles and limits for maximum performance are discussed. Algorithm performance for much more affordable 4 AMD Radeon RX 6900 XT based server with PCI 4.0 working under ROCm HIP is described for comparison.

Keywords: Parallel computing · CUDA · HIP · GEMM · High-speed GPU interconnect · Multi-GPU programming

1 Introduction

Modern high performance computing shows an evident trend of the growing use of specialized computational elements. Among the top 10 systems in the current Top500 list (June 2022) there is only one CPU-only supercomputer Fugaku with the number 2. The number 6 is the Sunway TaihuLight supercomputer based on the special Sunway SW26010 processors (each with 260 computing cores) requiring special programming techniques. The number 9 is the Tianhe-2A supercomputer combining Intel Xeon CPUs with the Matrix-2000 accelerators. All other systems (the numbers 1, 3–5, 7, 8 and 10) provide the major share of computing power via GPU accelerators: the numbers 1 (Frontier), 3 (Lumi), 4 (Summit), 5 (Sierra), 7 (Perlmutter), 8 (Selene) and 10 (Adastra).

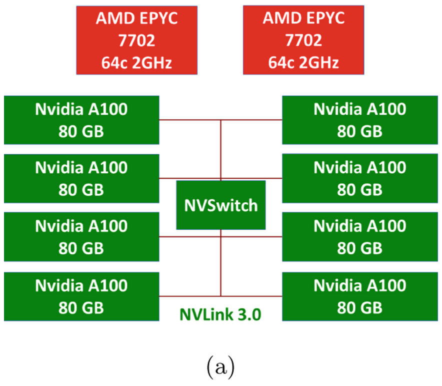
After the long period of Nvidia dominance in GPU computing technologies, now the real competition of vendors is developing. New GPUs made by AMD can be found in several largest supercomputers of the current top 10 systems (Frontier, Lumi and Adastra). The Aurora supercomputer that is to be commissioned soon in Argonne National Laboratory will be based on the new GPUs made by Intel. Each of these major GPU vendors proposes its own programming framework. In 2007 Nvidia pioneered the CUDA technology. In 2015 AMD

D. Balandin et al. (Eds.): MMST 2022, CCIS 1750, pp. 281–292, 2022.
https://doi.org/10.1007/978-3-031-24145-1_23

proposed the ROCm infrastructure and HIP for its GPUs as a nearly complete substitute for CUDA. In 2019 Intel announced the oneAPI infrastructure that uses the DPC++ cross-architecture language based on the SYCL standard for GPU programming. Interoperability of CUDA and the new technologies of AMD and Intel is crucial developing portable HPC software [1–3]. Many specific middle layers focused on performance portability are under active development (e.g. OpenMP, OpenACC, KOKKOS, Alpaka). Porting of linear algebra libraries is among the first priorities for the proper introduction of new technology. For example, the ROCm framework provides the hipBlas library that is a very close analogue of cuBlas and has similarly high efficiency [4].

Computing nodes of GPU-based supercomputers have multiple GPUs per node. The systems with AMD GPUs Frontier, Lumi and Adastra have very similar design with 4 GPUs/node. The systems with Nvidia GPUs differ: Summit has 6 GPU/node, Sierra and Perlmutter have 4 GPU/node and Selene has 8 GPU/node. In all these systems GPUs are interconnected by ultrafast communication links (Nvidia NVLink or AMD Infinity Fabric). For example, one A100 accelerator has 12 NVLink 3.0 links with 50 GB/s peak bandwidth each. One MI250X accelerator has 8 Infinity Fabric links with 100 GB/s peak bandwidth each. Such connectivity between GPUs within a supercomputer node opens unprecedented opportunities for parallel computations.

In this paper, we analyse the performance of the multi-GPU matrix-matrix multiplication algorithm developed and implemented by us previously [5] for two systems: a node with 8 A100 GPUs connected by NVLink 3.0 links through NVSwitch and a node with much more affordable AMD Radeon RX 6900 XT GPUs connected by PCIe 4.0. The SGEMM variant of the algorithm is considered. The accuracy of the previously proposed theoretical model [6] for performance tuning is validated. The performance influence of the tensor cores available in A100 [7,8] is described. The peculiarities of porting the algorithm from CUDA to HIP and running it on the AMD GPUs are described.



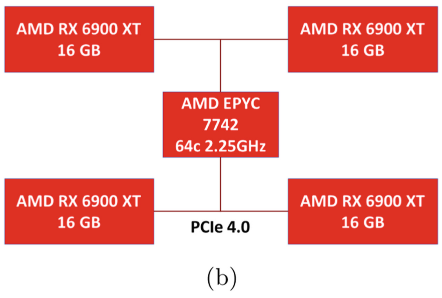


Fig. 1. The topology of the A100-equipped node of the cHARISMa supercomputer with two CPUs and eight Nvidia A100 GPUs by NVLink 3.0 (a) and the server with one CPU and four AMD RTX 6900 XT connected by PCIe 4.0 (b).

2 Related Work

Parallel algorithms for matrix multiplication evolve together with the development of the parallel computing technologies. The MPI algorithm for parallel matrix multiplication has been published soon after the MPI standard was introduced [9]. A runtime system called SuperMatrix that parallelize matrix operations for SMP and/or multi-core architectures was described in [10]. SuperMatrix introduced the concept of a set of tiles for work distribution among multiple threads. The PaRSEC framework [11] introduced the direct acyclic graph scheduling for dense algebraic operations. The subsequent PaRSEC implementation of the GEMM algorithm showed very high efficiency [12]. The near optimal parallel matrix-matrix multiplication algorithm COSMA was introduced based on the red-blue pebble game ideas [13]. Matrix multiplication is a generic test case for evaluation new programming models in HPC, e.g. the use for the Rust programming language [14].

Along with the academic projects SuperMatrix, PaRSEC and COSMA, there is a commercial multi-GPU Level-3 BLAS library cuBlas-XT developed by Nvidia. It was shown, however, that cuBlas-XT provides sub-optimal performance [15]. The multi-GPU level-3 BLAS library BLASX with improved scheduling was developed by Wang et al. [15]. The problem of communication optimal partitioning of a square computation domain over three heterogeneous processors has been considered recently [16].

PaRSEC, BLASX and COSMA are complex and multipurpose software projects. These projects (as well as cuBlas-XT) allow making calculations for matrices that are stored in the CPU memory (via special scheduling of CPU-GPU data transfers). The aim of this work is to analyze a much simpler matrix-matrix multiplication algorithm for matrices stored in GPU memory only [5,6]. Such an algorithm suits better for the purpose of benchmarking different types of GPUs and GPU-GPU interconnects.

3 Performance Model Overview

Here, we give a brief overview of the performance model developed in our previous work [5,6].

The GEMM algorithm is solving the equation

$$C = \alpha A * B + \beta C.$$

The uniqueness of the developed algorithm is that it uses only the resources of GPUs for its work, avoiding the necessity to wait for the data from the host CPU during the algorithm execution. This makes it possible to deploy such high bandwidth links as Nvlink between GPUs available in GPU servers nowadays (e.g., in DGX-like systems). Also, the asynchronous data transfers and computation overlap have been organized in the algorithm, providing the high performance rate.

For the case when we work with big matrix sizes the algorithm performance is limited by the computational abilities of GPUs. In the observed experiments the sizes of the tiles could be expected to be optimal for algorithm performance if they match the following conditions [6]

$$\begin{cases} N_i > 4k_{BW}N/(N-2k_{BW}), & N > 2k_{BW} \\ N_i > 2(Num_{GPUs}-1)BW_{math}/BW_{transfer}, \\ N_i > 2(Num_{GPUs}-1)BW_{math}/BW_{transfer}, \end{cases}$$

where N and N_i are the sizes of original matrices and tile matrices, $k_{BW} = BW_{math}/BW_{mem}$ is the bandwidth coefficient, BW_{math} and BW_{mem} are the mathematical and memory bandwidths of a GPU device, and Num_{GPUs} is the number of implemented in computation GPUs.

4 Testing Platforms

The results reported in this study are obtained on the nodes of the cHARISMa supercomputer at HSE University [17,18]. The nodes are based on the 8x Nvidia A100 GPU "Delta" platform with NVSwitch (Fig. 1a). Each GPU has 80 Gb of HBM2 memory, and eight GPUs are connected by NVLINK 3.0 via NVSwitch.

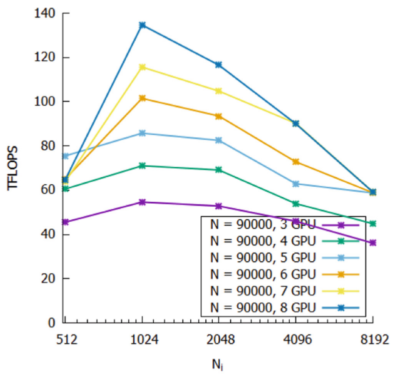
The benchmarking studies on the A100 equipped node are carried out using the standard HPC software stack based on CentOS Linux release 7.9.2009, GNU compilers 8.3.0, and CUDA Version 11.7.64 with the driver ver. 515.43.04.

The second platform is the server with 4 AMD RX 6900 XT GPUs connected by PCIe 4.0 (Fig. 1b). Each GPU has 16 Gb of GDDR6 memory. The server is based on the ASRock ROMED8-2T single socket motherboard with one AMD EPYC 7742 CPU. The benchmarking studies on this server are carried out using Ubuntu 20.04 Linux with AMD ROCm 5.2.1. RX 6900 XT GPUs have RDNA2 architecture that is a close relative of CDNA2 architecture of MI250X GPUs used in Frontier, Lumi and Adastra supercomputers.

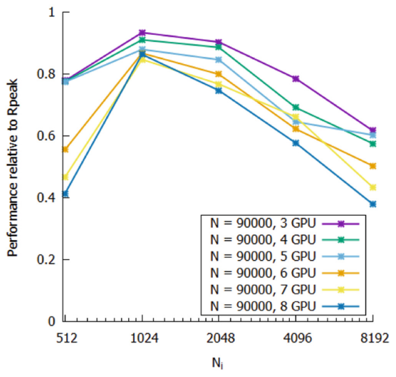
Table 1 summarizes the key features of two type of GPUs considered in terms of the parameters used in the performance model proposed in [6].

Table 1. Test platforms parameters

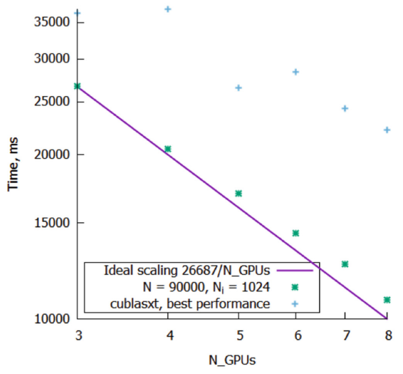
Hardware parameters	Nvidia A100	AMD RX 6900 XT
Peak FP32 performance (TFLOPS)	19.5	22.5
Real FP32 performance (TFLOPS)	18.4	21.4
Peak FP32 tensor core performance (TFLOPS)	156	–
Real FP32 tensor core performance (TFLOPS)	124	–
Peak GPU memory bandwidth (GB/s)	2039	512
Peak GPU-GPU bandwidth (GB/s)	300	32
Real GPU-GPU bandwidth (GB/s)	281	25.5



(a)



(b)



(c)

Fig. 2. Graph of the multi-GPU SGEMM operation on 3, 4, 5, 6, 7, and 8 A100 GPUs by tile size (N_i) for ($N = 90000$) elements in a row (column) of matrices without tensor core in absolute TFLOPS (a) and in relative data (b). The graph (c) is the time dependency by number of GPUs for tile size ($N_i = 1024$). The matrices A, B, and C are stored in devices 2, 1, 0 respectively.

5 Results

The possibility of reaching high levels of performance of the matrix multiplication algorithm considered ($C = \alpha A * B + \beta C$) is based on overlapping of computation and communication [5]. It is the size of the tiles N_i subdividing the matrices A, B and C that regulates the efficiency of this overlap, and hence the overall performance of the algorithm. Figure 2 shows the benchmark results for the A100 equipped node without using tensor cores. The results are presented for one of the largest possible matrix sizes $N = 90000$ where NxN is the size of the square matrices considered in this work.

We see that the higher is the number of GPUs N_{GPUs} the stronger is the dependence of the algorithm performance on the tile size N_i. This behavior is quite reasonable since the role of GPU-GPU communications increases for higher N_{GPUs}. For all $N_{GPUs} = 3$–8 we see the optimum $N_i = 1024$. At this tile size the efficiency of the algorithm (attained FLOPS over theoretical peak performance) reaches more than 80% for all N_{GPUs}.

The strong scaling for this case of $N = 90000$ is shown on Fig. 2 too and it is pretty close to the ideal scaling. Figure 2c shows the performance of cuBlas-XT for the exactly same problem ($N = 90000$, the points correspond to the minimum execution times at the variation of the tile size). One can see that the performance of cuBlas-XT is significantly worse that the performance of our algorithm.

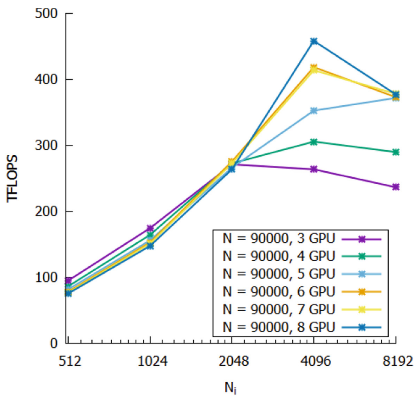
A100 accelerators have the tensor cores that speed up multiplications of small matrices. While FP32 peak performance of A100 is 19.5 TFLOPs, tensor cores boosts it to 156 TFLOPs. Since our parallel matrix-matrix multiplication algorithm uses cuBlas for multiplications of tiles within each GPU, the algorithm can benefit from using tensor cores. The use of tensor cores in single precision can be switched on and switched off using CUDA calls (in double precision tensor cores can not be switched off and are deployed automatically whenever possible).

Figure 3 shows the benchmark results for our SGEMM algorithm with tensor cores switched on. One can see that the optimum values of N_i move to larger values. Despite significant acceleration in absolute values, the level of efficiency with tensor cores becomes lower (even lower than 40% $N = 8$).

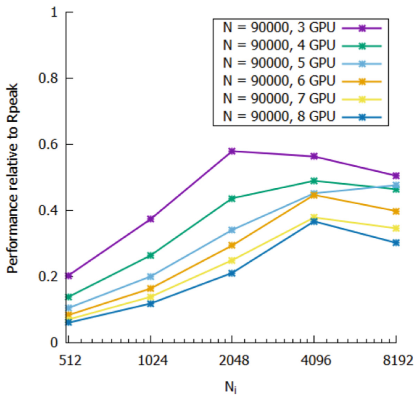
Figure 4 shows the results for the server with AMD RX 6900 XT GPUs. The CUDA code of our algorithm has been ported to HIP using the Perl-based hipify tool available in the ROCm framework. hipBlas GEMM function calls are used instead of cuBlas. The results show surprisingly modest performance and low efficiency that is lower than 40% for 3 and for 4 GPUs. It is a strange fact since even for the quite old Nvidia GTX1070 GPUs the similar benchmark showed efficiency over 50% (see [6]).

6 Discussion

The algorithm had to be improved to manage with any size of the matrices. In the ending part of the algorithm the storing A and B devices exchange and



(a)



(b)

Fig. 3. Graph of the multi-GPU SGEMM operation on 3, 4, 5, 6, 7, and 8 A100 GPUs by tile size (N_i) for ($N = 90000$) elements in a row (column) of matrices with tensor core in absolute TFLOPS (a) and in relative data (b). The matrices A, B, and C are stored in devices 2, 1, 0 respectively.

compute the left matrices part after division them into bands. This way has been chosen firstly because we had an idea to improve the algorithm which requires it. Secondly, it is the one of less computationally expensive solution to manage with left parts at the same time. However, in some cases the effect of this step is too strong to be able to move the performance maxima. Moreover, for small block sizes the GPUs have to multiply tall-and-skinny matrices. Improvement of this issue has not been implemented in the algorithm yet.

Figure 5 shows the profiles of the algorithm execution with and without tensor cores. While we use tensor core, we achieve reasonably fast computational speed, but also we moderately, but sensibly lose the accuracy. The observed performance on each computation kernels are unduly low from peak for example in comparison

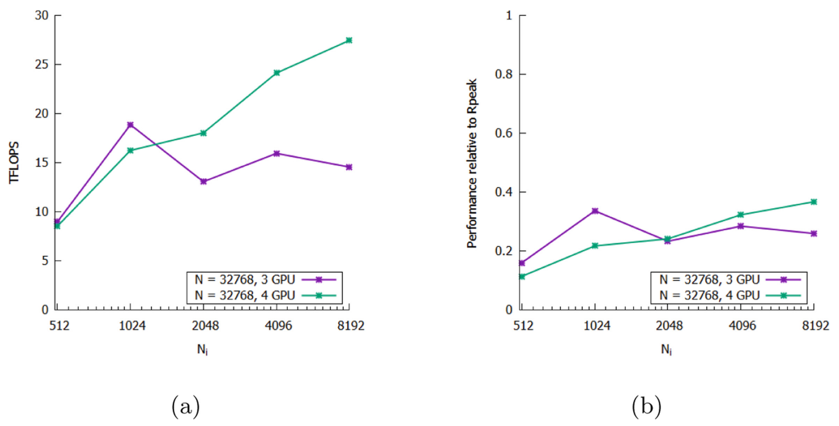


Fig. 4. Graph of the multi-GPU SGEMM operation on 3 and 4 Radeon RX 6900XT GPUs by tile size (N_i) for ($N = 32768$) elements in a row (column) of matrices without tensor core in absolute TFLOPS (a) and in relative data (b). The matrices A, B, and C are stored in devices 2, 1, 0 respectively.

with case without tensor core. In double precision the accuracy loss is much less detectable in comparison with the same matrix size in single precision (FP64 tensor cores are IEEE-compliant).

The cuBLAS library allows to use tensor core in single precision on A100 GPUs, but we could not enable FP32 tensor cores in the cuBLAS-XT library calls. Probably, the cuBLAS-XT supports only the default settings in this issue.

The difficulties in using new GPU technologies can be illustrated by the following fact. From experiments it comes out that version 2021.3.2.4-027534f and earlier Nvidia Nsight Systems Profiler gives incorrect synchronization profiles for multiple GPUs. In newer versions this problem has been corrected. Due to this problem a lot of efforts have been spend looking for the possible reasons of improper synchronization on A100 GPUs (while no problems were observed on V100 in our previous work).

We could not achieve the peak performance for RX 6900 XT GPUs during the algorithm. However, in a single launch of SGEMM in one GPU we do achieve the value very close to the peak. We marked (in Fig. 6) the time needed to compute a band multiplication with $2^{32} \approx 4.3$ GFLOPS, thus, the performance is 8 TFLOPS from the peak 22.5 TFLOPS. Figure 6 shows also considerable delays between data transfers. We suppose that these problems might be explained by the difficulty for the GPU (that is a consumer GPU, not a server grade GPU) to perform computations and data transfers simultaneously. This observation points to the fact that the multi-GPU performance of the algorithm considered can not be transferred easily to the consumer-grade GPU systems.

The results of the benchmarks of the A100-equipped node give us the possibility to test the performance model developed previously [6]. Table 2 summarizes the empirical values and the predictions. We show the threshold values "theoretical threshold" and the next larger $N_i = 2^n$ "theoretical N_i". The overall

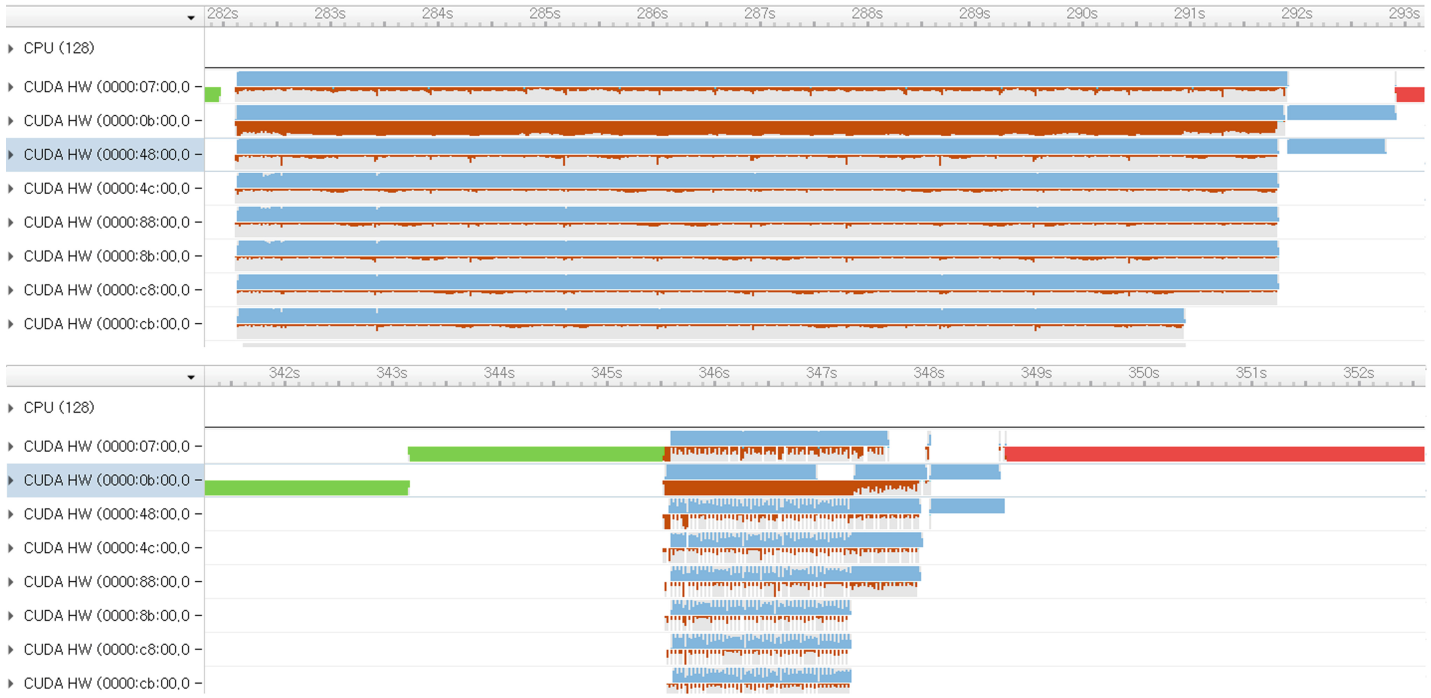


Fig. 5. The profiles of the multi-GPU SGEMM operation on 8 A100 GPUs without tensor core (upper) and with tensor core (lower) usage. Number of elements ($N = 90000$) in a row (column) of matrices and tile size ($N_i = 1024$) in case without tensor core and ($N_i = 4096$) in case with tensor core. The matrices A, B, and C are stored in devices 2, 1, 0 respectively. The green bars are the data supply from the host to devices for execution, the red bars are the resulting data supply to the host, the blue bars are the computations in GPUs, and the brown bars are the peer-to-peer data transfer operations between devices. (Color figure online)

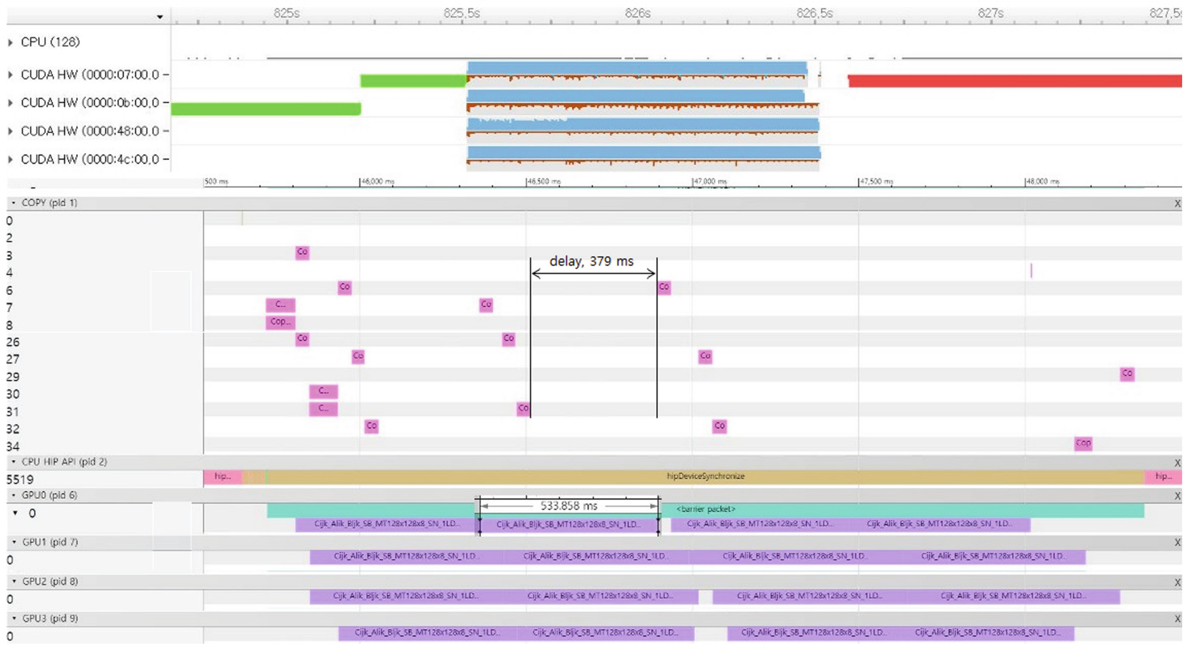


Fig. 6. The profiles of the multi-GPU SGEMM operation on 4 A100 GPUs without tensor core (upper) and 4 RX 6900 XT GPUs (lower). Number of elements ($N = 32768$) in a row (column) of matrices and tile size ($N_i = 1024$) in case on A100 GPUs and ($N_i = 8192$) in case on RX 6900 XT GPUs. The matrices A, B, and C are stored in devices 2, 1, 0 respectively. The green bars (upper, in lower omitted) are data supply from the host to devices for execution, the red bars (upper, in lower omitted) are result data supply to the host, the blue bars (upper) and purple bars (lower) are computations in GPUs, and the brown bars (upper) and pink bars (lower) are peer-to-peer data transfer operations between devices. (Color figure online)

accuracy of the predictions is higher than 50%. There are two factors that limit this accuracy: 1) the model does not take into account the possible last stage of the algorithm with poor load balancing, 2) the GPU performance for smaller tile sizes can be much lower than the maximum performance for larger tiles.

Table 2. The optimal tile sizes (N_i): the empirical values from the benchmarks of A100 system (see Fig. 2 and Fig. 3) and the predictions of the model [6].

A100 with NVLink		N_{GPUs}					
		3	4	5	6	7	8
Without tensor cores	Empirical	1024	1024	1024	1024	1024	1024
	Theoretical threshold	262	393	524	655	786	917
	Theoretical N_i	512	512	1024	1024	1024	1024
With tensor cores	Empirical	2048	4096	4096	4096	4096	4096
	Theoretical threshold	1766	2648	3531	4413	5296	6178
	Theoretical N_i	2048	4096	4096	8192	8192	8192

In the future, it would be interesting to see how the performance of the our algorithm compares with the performance of the SuperMatrix, ParSEC and COSMA implementations. Deployment of our algorithm in such a standard test as, for example, High Performance Linpack (HPL) would be another interesting problem for further study.

7 Conclusions

The empirical optimum parameters for our multi-GPU SGEMM algorithm obtained for 8 A100 based server with NVLink are compared with the theoretical model predictions. It is shown how using tensor cores changes the balance between communication and computation. The benchmark results obtained using the server with 4 AMD RDNA2-type GPUs connected by PCI 4.0 reveals certain peculiarities of porting our multi-GPU SGEMM algorithm from CUDA to HIP.

Acknowledgment. The article was prepared within the framework of the HSE University Basic Research Program. This research was supported in part through resources of supercomputer facilities provided by HSE University. This research was supported in part through computational resources of HPC facilities at HSE University [18].

References

1. Kondratyuk, N., Nikolskiy, V., Pavlov, D., Stegailov, V.: GPU-accelerated molecular dynamics: state-of-art software performance and porting from Nvidia CUDA to AMD HIP. Int. J. High Perform. Comput. Appl., p. 10943420211008288 (2021)
2. Williams-Young, D.B., et al.: Achieving performance portability in gaussian basis set density functional theory on accelerator based architectures in NWChemEx. Parallel Comput. **108**, 102829 (2021)

3. Cojean, T., Tsai, Y.H.M., Anzt, H.: Ginkgo-A math library designed for platform portability. Parallel Comput. **111**, 102902 (2022)
4. Brown, C., Abdelfattah, A., Tomov, S., Dongarra, J.: Design, optimization, and benchmarking of dense linear algebra algorithms on AMD GPUs. In: 2020 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7. IEEE (2020)
5. Choi, Y.R., Nikolskiy, V., Stegailov, V.: Matrix-Matrix Multiplication using multiple GPUs connected by NVLink. In: 2020 Global Smart Industry Conference (GloSIC), pp. 354–361. IEEE (2020)
6. Choi, Y.R., Nikolskiy, V., Stegailov, V.: Tuning of a matrix-matrix multiplication algorithm for several GPUs connected by fast communication links. In: Sokolinsky, L., Zymbler, M. (eds.) PCT 2022. CCIS, vol. 1618, pp. 158–171. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-11623-0_12
7. Markidis, S., Der Chien, S.W., Laure, E., Peng, I.B., Vetter, J.S.: Nvidia tensor core programmability, performance and precision. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 522–531. IEEE (2018)
8. Dakkak, A., Li, C., Xiong, J., Gelado, I., Hwu, W.m.: Accelerating reduction and scan using tensor core units. In: Proceedings of the ACM International Conference on Supercomputing, pp. 46–57 (2019)
9. Van De Geijn, R.A., Watts, J.: SUMMA: scalable universal matrix multiplication algorithm. Concurrency Pract. Experience **9**(4), 255–274 (1997)
10. Chan, E., et al.: SuperMatrix: a multithreaded runtime scheduling system for algorithms-by-blocks. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 123–132 (2008)
11. Wu, W., Bouteiller, A., Bosilca, G., Faverge, M., Dongarra, J.: Hierarchical DAG scheduling for hybrid distributed systems. In: 2015 IEEE International Parallel and Distributed Processing Symposium, pp. 156–165. IEEE (2015)
12. Herault, T., Robert, Y., Bosilca, G., Dongarra, J.: Generic matrix multiplication for multi-GPU accelerated distributed-memory platforms over PaRSEC. In: 2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), pp. 33–41. IEEE (2019)
13. Kwasniewski, G., Kabić, M., Besta, M., VandeVondele, J., Solcà, R., Hoefler, T.: Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, pp. 1–22. Association for Computing Machinery, New York, NY, USA (2019)
14. Bychkov, A., Nikolskiy, V.: Rust language for supercomputing applications. In: Voevodin, V., Sobolev, S. (eds.) RuSCDays 2021. CCIS, vol. 1510, pp. 391–403. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92864-3_30
15. Wang, L., Wu, W., Xu, Z., Xiao, J., Yang, Y.: BLASX: a high performance level-3 BLAS library for heterogeneous multi-GPU computing. In: Proceedings of the 2016 International Conference on Supercomputing, pp. 1–11 (2016)
16. Malik, T., Lastovetsky, A.: Towards optimal matrix partitioning for data parallel computing on a hybrid heterogeneous server. IEEE Access **9**, 17229–17244 (2021)
17. Kondratyuk, N., et al.: Performance and scalability of materials science and machine learning codes on the state-of-art hybrid supercomputer architecture. In: Voevodin, V., Sobolev, S. (eds.) RuSCDays 2019. CCIS, vol. 1129, pp. 597–609. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36592-9_49
18. Kostenetskiy, P.S., Chulkevich, R.A., Kozyrev, V.I.: HPC resources of the higher school of economics. J. Phys. Conf. Ser. **1740**, 012050 (2021). https://doi.org/10.1088/1742-6596/1740/1/012050