Tuning of a Matrix-Matrix Multiplication Algorithm for Several GPUs Connected by Fast Communication Links

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

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

Abstract. The usage of one of the latest high-performance hardware types (nodes with several GPUs connected by high bandwidth and low latency communication links), requires algorithms where the CPU is used only to manage the program execution, and GPUs are used for computations. In this work, we study an original GPU-only parallel matrix-matrix multiplication algorithm ($C = \alpha A * B + \beta C$) for servers with multiple GPUs. The algorithm is implemented using CUDA. The performance of this multi-GPU GEMM algorithm and the method defining the optimal tile size using the hardware parameters and the matrix size are considered. The usability of the developed performance model by benchmarking two types of GPU servers is verified.

Keywords: Parallel computing · CUDA · GEMM · high-speed GPU interconnect · multi-GPU programming

1 Introduction

Today, accelerators, especially GPUs, have arisen as an important component of supercomputers. Therefore, many algorithms and programs have been modified accordingly: molecular dynamics codes such as GROMACS [1], particle-in-cell plasma simulation codes such as PICADOR [2], electronic structure calculation codes such as Quantum Espresso [3,4], or astrophysical hydrodynamics codes such as GPUPEGAS [5,6]. The main idea of these modifications is to offload some possible workload to be computed on GPUs. The strategy is to organize parallel computing across the nodes of a supercomputer using MPI and inside the node with GPUs using OpenCL/CUDA/HIP (see e.g. [7–9]). The common issue then is data transfer bandwidth, which restricts the GPU utilization rate [10,11], hence, there arises a need for very high-performance links such as NVlink or Infinity Fabric. Servers with GPU devices connected with such links generally show high efficiency, especially in the cases of GPU-only algorithms (e.g. [12–14]).

Many problems are based or rely on linear algebra, particularly, on matrix operations. Matrix multiplication is a more or less costly operation that might be

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
L. Sokolinsky and M. Zymbler (Eds.): PCT 2022, CCIS 1618, pp. 158–171, 2022.
https://doi.org/10.1007/978-3-031-11623-0_12

appraised as a benchmark to study the performance of computers (e.g. [15–18]). While GPUs are used for such computations, we can distinguish cases when only GPUs are used for it or, otherwise, other types of computing units may participate in it. On the other hand, contemporary compute nodes have fast links (e.g. NVlink), such as those described earlier, between GPUs and slower links (e.g. PCIe) between the CPU and GPU. Thence, if we have to compute several consecutive mathematically heavy operations, it will be less costly to launch it on GPUs without giving requests to the host memory for data transfer. In addition, during multi-GPU computing, devices have to continuously send and receive data, therefore, the reuse of data on the units involved, if possible, will make the process faster and less complicated.

2 Related Work

Parallel matrix multiplication algorithms have a long history of development. For example, after the introduction of the MPI standard, a Scalable Universal Matrix Multiplication Algorithm (SUMMA) for parallel matrix multiplication was published [19]. Different parallel GEMM algorithm models and structures were under research [20,21]. There were attempts to redesign the core of the algorithm [22]. One of the recent works showing approximate peak performance is the Communication Optimal S-partition-based Matrix multiplication Algorithm (COSMA) [23]. Particular attention there is paid to the importance of I/O operations and communications management, which is the key to reach comparably fast performance in relation to the best existing solutions such as ScaLAPACK [23].

Analytical models help to find optimal parameters for GPU algorithms. Such a model for a single GPU is presented in Tran, Lee, and Choi [24].

Since we work with multiple GPUs, we also have to pay attention to the synchronization process between the devices [25].

The optimal partitioning of a computational domain over several heterogeneous processors, processor load balancing and the minimization of inter-processor communication costs are crucial for data-parallel dense linear algebra and other applications that have a similar communication pattern in modern hybrid servers. One of the most recent works in this field is devoted to the optimal partitioning of a square computational domain over three heterogeneous processors [26].

An algorithm showing high efficiency for hybrid platforms that have fast communication links installed between the CPU and GPU is built over the PaRSEC runtime system [27]. It stores data in the host memory, and GPUs are supplied with the necessary data chunks by the CPU-GPU interconnect.

In our previous work [28], we introduced a matrix multiplication algorithm for a multi-GPU node that uses only GPUs for data storage and computation. The algorithm is aimed for the use in nodes with a fast interconnect between GPUs. It was shown that the standard cuBLAS-XT function from Nvidia CUDA SDK provided suboptimal performance on a multi-GPU node. The asynchronous

data transfer and compute organization in the algorithm allowed us to get much higher performance. In this work, we present an analytical model that estimates the time-to-solution for the proposed algorithm using basic hardware parameters. This analytical model can be used to optimize the algorithm verified in this work by benchmarking two different multi-GPU servers.

3 Testing Platforms

The results reported in this study are obtained on the K-type nodes of the cHARISMa supercomputer at HSE University [29,30]. The nodes are based on DELL PowerEdge C4140 servers with two Intel Xeon Gold 6152 CPUs, and four NVidia Tesla V100 GPUs (Fig. 1a). Each GPU has 32 GB of HBM2 memory, and the four GPUs are connected by NVLINK 2.0, forming a fully connected ('all-to-all') topology.

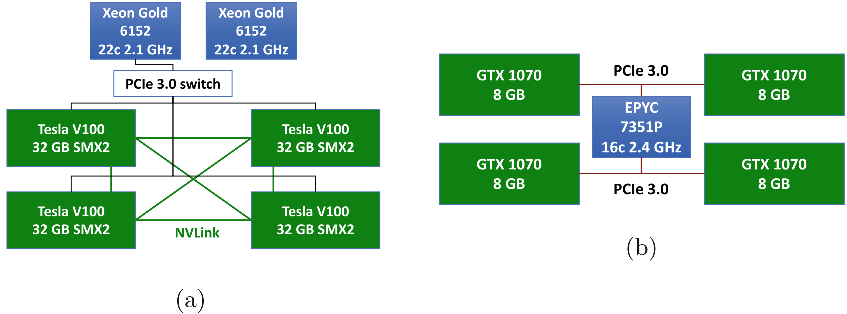

Fig. 1. Topology of the DELL PowerEdge C4140M node of the cHARISMa supercomputer with two CPUs and four Nvidia Tesla V100 GPUs connected by NVLINK 2.0 (a) and the TYAN B8021G88V2HR-2T server with one CPU and four Nvidia GTX1070 GPUs connected by PCIe 3.0 (b).

The benchmarking studies presented in this work are carried out using the standard HPC software stack based on CentOS Linux release 7.6.1810, GNU compilers 7.3, and CUDA Version 10.2.89 with driver ver. 440.33.01.

The second platform is the TYAN B8021G88V2HR-2T server at JIHT RAS. It has the EPYC 7351P CPU and four NVidia GeForce GTX1070 GPUs connected by PCIe 3.0 (Fig. 1b). Each GPU has 8 GB of GDDR5 memory.

4 Parallel Matrix-Matrix Multiplication Algorithm for Multiple GPUs

The studied algorithm is an improved version of the algorithm presented previously [28]. The research is based on the general matrix-matrix multiplication algorithm for the following matrix operation

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

The main idea of the process is to calculate the tiles of the resulting matrix C on each device with the reuse of possible data of the input matrix bands [28]. In the research, we observe a simpler case with 2 square matrices A and B, where $N * N$ elements are in both. Each matrix is divided into some number of equal-sized bands to share the computational load between different GPUs. Then, from the pairs of the bands from A and B, we calculate the appropriate tiles of the resulting matrix C.

The GEMM function from the cuBLAS library is used as the core of the proposed algorithm. It normally works with column-oriented matrices, but is also available for differently oriented matrices. In our case, if we perceive the column orientation as a default configuration, we store the data of the matrix A transposed. This simplifies the data transfer operation because we can just send one long data line from the source memory.

The algorithm is GPU-oriented, thus, the CPU gives only instructions. The source data are located in GPUs, and transfers are sent only between GPUs. For computation in devices, 2 bands are allocated for the same matrix.

The classical SUMMA algorithm [19] is developed using MPI for distributed memory systems. SUMMA does not use asynchronous data transfers, which could help overlapping computations and communications. The algorithm proposed here works with the "rows" of the matrix A and the "columns" of the matrix B. However, GPUs perform computational operations reasonably quickly. To supply data in time, high-speed communication links and data division into a sufficiently large number of chunks are required. To manage this issue, we organize asynchronous communications and computations. Then, we do not supply the original data of the matrix C, but we add βC due to the copy kernel of the results received in the final stage.

5 Theoretical Optimal Tile Size

For a better performance of the algorithm, we regulate the tile size one by one and find the best one. The performance time of the GEMM kernel can be approximated [31] as

$$T_{GEMM} = \max\left(T_{mem}, T_{math}, T_{instructions}\right), \tag{1}$$

where T_{mem} is the data management time in the frame of a device, T_{math} is the time spent on mathematical operations, and $T_{instructions}$ is the time during which instructions are given by the CPU. In the basis of the algorithm, we assume in advance that $T_{instructions} \ll T_{math}$ since, otherwise, a multi-GPU implementation would slow down the execution by waiting for instructions. To satisfy this condition, the matrices involved in GEMM should not be too small. To understand if $T_{math} > T_{mem}$, we can find the arithmetic intensity [32] of

GEMM operations, which should be greater than the defined value for the devices involved, i.e., the math and memory bandwidth proportion. In the case of single precision (32-bit), it can be found by the proportion

$$Intensity = \frac{FLOPS}{bytes} = \frac{2MNK}{4(MN + NK + MK)} > BW_{math}/BW_{mem} = k_{BW}, \tag{2}$$

where M, N, K are the numbers of elements in the columns or rows of the matrices A, B, C; BW is the processor math or memory bandwidth, appropriately. In our case, Eq. (2) becomes

$$Intensity = \frac{N_i^2 N}{2(N_i N + N_i N + N_i^2)} = \frac{N_i N}{4N + 2N_i} > k_{BW}. \tag{3}$$

After simple arithmetic we get $(N > 2k_{BW})$

$$N_i > 4k_{BW}N/(N - 2k_{BW}),\ T_{math} > T_{mem}. \tag{4}$$

For example, for single precision of the Nvidia V100 GPU ($k_{BW} = 16.6$) if $N = 2^{12}$, $N_i > 66.9$ needed to $T_{math} > T_{mem}$, if $N = 2^{13}$, then $N_i > 66.6$, if $N = 2^{14}$, then $N_i > 66.5$. Each time can be found by the equations

$$T_{math} = FLOPS/BW_{math},\ T_{mem} = bytes/BW_{mem}, \tag{5}$$

which for the algorithm transform into

$$T_{math} = 2N_i^2 N/BW_{math},\ T_{mem} = (8NN_i + 4N_i^2)/BW_{mem}. \tag{6}$$

On the other hand, we have to keep data supply from memory storing original matrices simultaneously. The GPU device, where the original matrix is located, sends bands to the other GPUs, so $Num_{GPUs} - 1$ data transfer operations need be sent. If we are not dealing with a small number of tiles, the reuse of data in the model can approximate the time needed for one kernel launch (T_{kernel}) as follows:

$$T_{kernel} = \max(T_{GEMM}, (Num_{GPUs} - 1)T_{transfer}). \tag{7}$$

Otherwise, the transfer time ($T_{transfer}$) can increase by up to 3 times (3 matrices A, B, C). This effect is conspicuously demonstrated when we store all matrices in one device [28], where the transfer time influences the performance of the whole task and has decreased efficiency compared to the case with the spread store of the matrices.

Furthermore, if we suppose that we have reached a fully parallel model, then the full task performance time (T_{task}) will be approximately

$$T_{task} = 3T_{transfer} + N/(N_i Num_{GPUs})T_{kernel}. \tag{8}$$

Particularly, the data transfer time has a linear form [33]

$$T_{transfer} = bytes/BW_{transfer} + T_{latency}. \tag{9}$$

We work with sufficiently large matrices such that the term with transfer bandwidth between GPUs ($BW_{transfer}$) dominates [33]

$$bytes/BW_{transfer} \gg T_{latency}. \quad (10)$$

Accordingly, expression (9) together with condition (10) in single precision can be converted into

$$T_{transfer} = 4N_iN/BW_{transfer}. \quad (11)$$

Today, high-performance computing environments have exceptionally fast math bandwidth (BW_{math}) or in-device memory bandwidth (BW_{mem}) comparing with data transfer ($BW_{transfer}$) from another device. It means that we will slow down the execution of the task whenever we make GPUs wait for data supply. There are two possible situations, if $Intensity > k_{BW}$, we determine from Eqs. (1), (6), (7), (11)

$$T_{kernel} = T_{GEMM} = T_{math},\ N_i > 2(Num_{GPUs} - 1)BW_{math}/BW_{transfer}, \quad (12)$$

and if $Intensity < k_{BW}$,

$$T_{kernel} = T_{mem},\ N_i > N\left((Num_{GPUs} - 1)BW_{mem}/BW_{transfer} - 2\right). \quad (13)$$

Interesting remarks can be made here from conditions (12) and (13). If for some reason we have $BW_{transfer} \gg BW_{math\ or\ mem}$, we will also have $T_{kernel} = T_{GEMM}$ since N_i should be a natural number; but, probably, some reversed situation can be exposed for small N_i and will have a comparably fast $BW_{transfer}$.

We also minimize the performance time in Eq. (8) by regulating N_i. With (6), (11), (12), (13), we determine for $Intensity > k_{BW}$

$$T_{task}(N_i) = (3N/BW_{transfer} + 2N^2/(Num_{GPUs}BW_{math}))N_i, \quad (14)$$

otherwise

$$T_{task}(N_i) = (3N/BW_{transfer} + 4N/(Num_{GPUs}BW_{mem}))N_i + 8N^2/(Num_{GPUs}BW_{mem}). \quad (15)$$

Expressions (14) and (15) show that the performance will be better for lower N_i. However, we still have to satisfy three conditions (12), (13), and (4) at once.

6 Tile Size Tuning for Different Platforms

To come up with the required tile size using the materials presented in Sect. 5, we have to pay attention to arithmetical intensity (4) and conditions (12) or (13). Importantly, we see that if the algorithm is math limited ($Intensity > k_{BW}$), then N_i is independent from N. In the work frame, we observe only matrices with $N \geq 8192$, and both N and N_i being some natural number power of 2.

6.1 Expected tile Size for Experimental Environments

The first computing system is composed by Nvidia V100 GPUs connected by NVLink 2.0. The V100 GPU single precision parameters are presented in Table 1. To find N_i, we use the maximal expected $BW_{math\ or\ mem}$, but for the transfer one, the rate is found by the bandwidth test. We have from (4) at least $N_i > 66$ to have a mathematical limited condition and from (12) $N_i > 616$ for 2 GPUs and $N_i > 1849$ for 4 GPUs. Thus, the optimal sizes are $N_i = 1024$ for 2 GPUs and $N_i = 2048$ for 4 GPUs.

The second system is Nvidia GeForce GTX 1070 GPUs connected by PCIe 3.0 (see Table 1). From the intensity condition we get $N_i > 90$, from (12) $N_i > 1352$ for 2 GPUs and $N_i > 4058$ for 4 GPUs. Thus, the optimal sizes are $N_i = 2048$ for 2 GPUs and $N_i = 4096$ for 4 GPUs.

Table 1. Test platform parameters and best predicted tile sizes

Hardware Parameters	Nvidia V100	Nvidia GTX1070
BW_{math} (Gflops/sec)	14899	5783
BW_{mem} (Gb/sec)	900	256
test $BW_{transfer}$ (Gb/sec)	48.33	8.55
ideal $BW_{transfer}$ (Gb/sec)	50	16
Algorithm Parameters	Nvidia V100	Nvidia GTX1070
N_i for 2 GPUs	1024	2048
N_i for 4 GPUs	2048	4096

6.2 Experimental Results

In this section, we would like to present the performance time and tile size dependence. For each test platform, we observed 2 cases of data store, when all square matrices with N^2 elements are located in the memory of one GPU, or the matrices A, B, C are located in 3 GPUs, one in each. We increased the testing data by 2, thus, N and N_i have the value being some power of 2. In addition, the size ranges are $N \geq 8192$ and $N_i \geq 512$. The matrix size is chosen consciously large to avoid the influence of minor parameters such as $T_{instructions}$ (Eq. (1)), and the tile size to show the behavior when it goes over the appointed value in Table 1. If we do not meet extra conditions to the tile size, the largest one we can deal with is $N_i = N/Num_{GPUs}$ due to equal task division between the implemented GPUs.

Each server we tested has 4 GPUs, which we could use when launching the program. For each, we analyzed the cases with 2 or 4 GPUs. In addition, we wanted to present some testing profiles to describe how GPU performance relates to the tile size (N_i). While working with CUDA, we got profiles using nvprof and visualized them by the Nvidia Visual Profiler. In Fig. 2, we can see the involved processor in the column on the left side and the type of operation described, such as instructions given on the CPU, data transfer, or kernel (GEMM) execution.

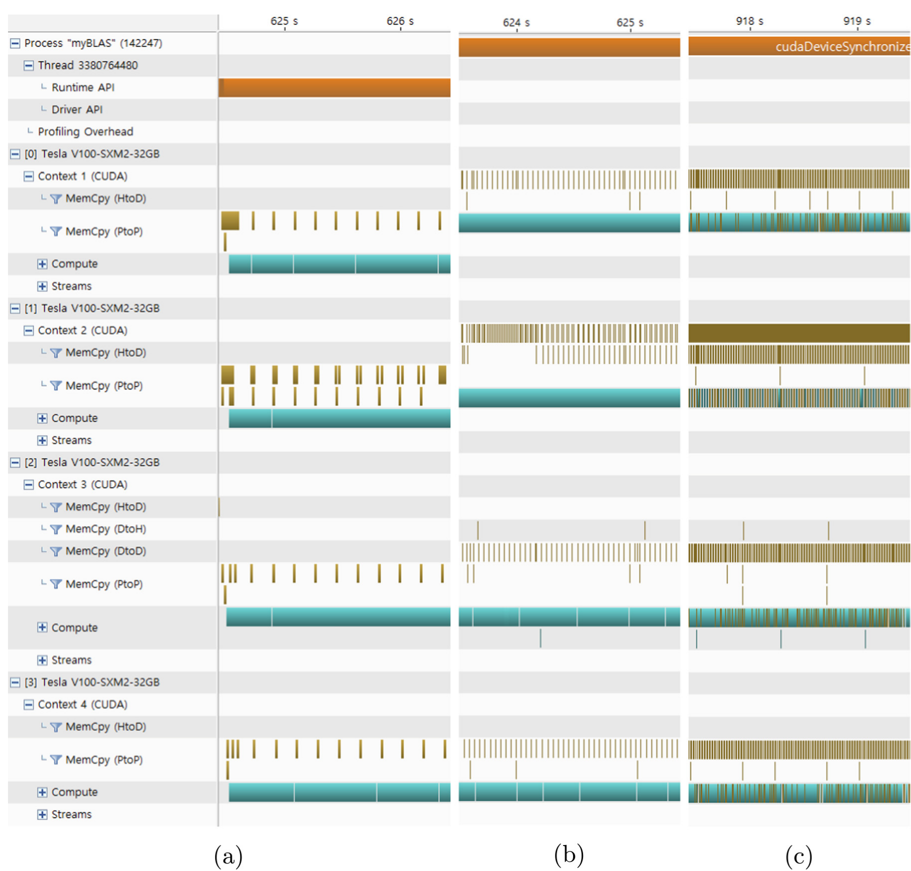

Fig. 2. Profile parts of the multi-GPU GEMM operation on V100 with the proposed algorithm performing on 4 GPUs. Number of elements ($N = 65536$) in a row (column) of matrices and different tile sizes ($N_i = 4096$ (a), $N_i = 2048$ (b), $N_i = 1024$ (c)). The matrices A, B, and C are stored in devices with id 0, 1, and 2.

Four V100 Connected by NVLink. The GPUs we worked with have 32 GB of in-device memory. For our experiments, the maximum matrices with $N = 32768$ fit in one device memory together and with $N = 65536$ separately. However, in the second case, we met the memory limit for allocating additional band matrices involved in the computing process, thus, the maximum tile size we could use was $N_i = 4096$. In Fig. 3, we can find the best performance with $N_i = 1024$ for 2 GPUs and $N_i = 2048$ for 4 GPUs. These numbers match with those defined in Sect. 6.1.

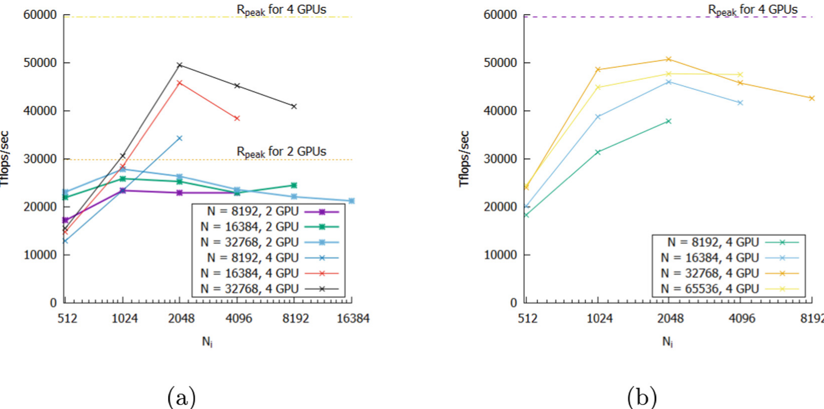

Fig. 3. Graph of the multi-GPU GEMM operation on V100 with the proposed algorithm performance speed on 2 and 4 GPUs by tile size (N_i) for a different number of elements (N) in a row (column) of matrices. The matrices A, B, and C are stored in device 0 (a) or in devices 0, 1, 2 (b), respectively. The dashed lines show the total single precision peak performance of 2 and 4 GPUs, respectively.

Figure 2 for $N = 65536$ demonstrates the execution of 4 Volta 100 GPUs for different tile sizes (N_i). When using the appointed size N_i (see Fig. 2b), we achieve the most frequent data transfer (for example, comparing with Fig. 2a), not taking longer than the kernel time (see Fig. 2c). This matches with the propositions given in Sect. 5. Therefore, the benchmark results verify the model.

Four GTX1070 Connected by PCIe 3.0. This testing platform has 8 GB of local memory in each GPU. We could perform benchmarks up to $N = 16384$ for a single device located case and $N = 32768$ and up to $N_i = 4096$ for a distributed storage case. The results are presented in Fig. 4.

The best performance with 2 GPUs is achieved with $N_i = 2048$ as proposed (Fig. 4a). For 4 GPUs, we have a lack of data due to the memory limit, however, the performance keeps growing to $N_i = 4096$, so we can expect it to be the best. Hence, the found tile size according to the arithmetical model matches for 2 GPUs, and, however, the tile size for 4 GPUs is shown to be the best only on one point due to the memory limit; the behavior of the graphs matches with the propositions given beforehand, that is why we can assume that the results fully comply with the mathematical algorithm.

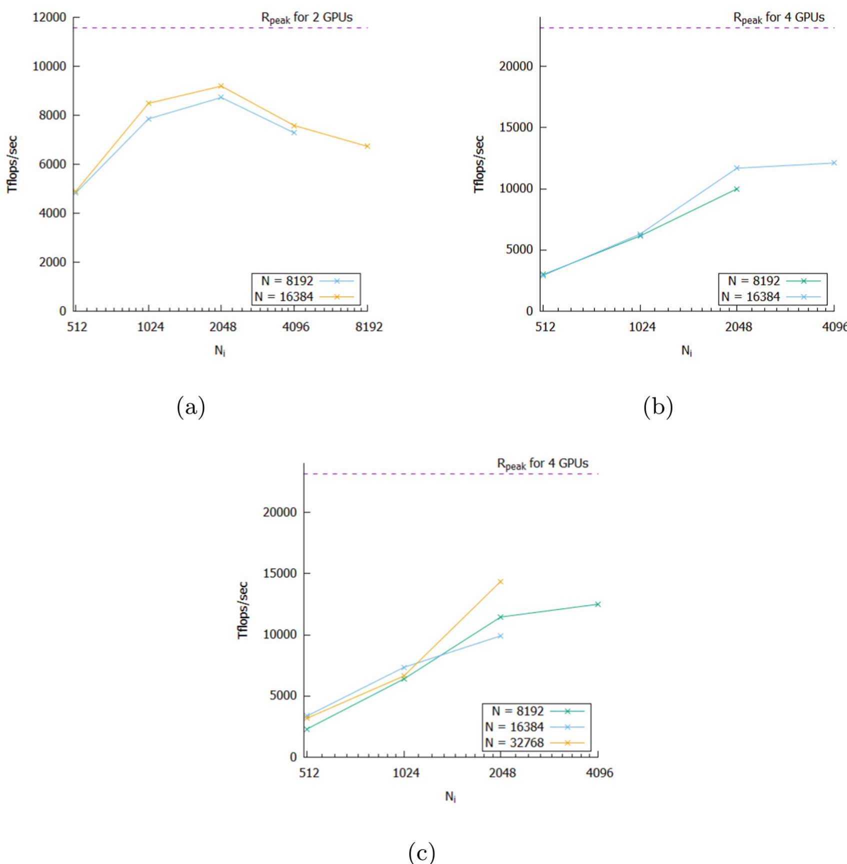

Fig. 4. Graph of the multi-GPU GEMM operation on GeForce GTX1070 with the proposed algorithm performance speed on 2 (a) and 4 GPUs (b,c) by tile size (N_i) for a different number of elements (N) in a row (column) of matrices. The matrices A, B, and C are stored in device 0 (a,b) or in devices 0, 1, 2 (c), respectively. The dashed lines show the total single precision peak performance of 2 and 4 GPUs, respectively.

7 Discussion

The performance of the proposed multi-GPU general matrix multiplication algorithm is dependent on the size of tile matrices. We determined tile size values for definite device and communication link properties, as well as the number of GPUs involved, to achieve the best performance in the tests. Then we performed experiments with large matrices to reach the performance limited by the computing ability of the GPU.

In the most general case, the computation of GEMM functions on a GPU can be limited by computational performance or memory bandwidth, depending on the matrix size. However, in our work, we observe only a computationally

limited case. There are secondary, but still important conditions that affect the performance. The first is the initialization latency of kernels on the GPU called by the CPU, and the consumed time for it should be much less than the time of the actual workload. The second is the data transfer time, which should be less than the computing time to keep the continuous operation of multiple devices.

We propose a mathematical model to define the optimal tile size based on known hardware parameters. The model is based on some assumptions about the system. In particular, the system should include similar GPUs connected by links with the same and fairly stable throughput values. In practice, determining the real properties of communication networks may be a non-trivial task, in particular, the behavior of these links under a certain load may require additional tests and profiling.

The proposed algorithm with a high degree of probability will not be optimal for exotic and specific cases (see e.g. [34]). In this paper, more typical and common cases are considered. Further development of the algorithm may include increased flexibility and versatility to work effectively on a wide range of configurations.

8 Conclusion

An analytical model that takes into account the hardware parameters of the GPU server (data transfer bandwidth and GPU performance) and predicts the optimal tile size for the multi-GPU GEMM algorithm is developed. The benchmarks on two different GPU servers (one with V100 GPUs and NVlink and another with GTX1070 GPUs and PCIe links) confirm the applicability of the analytical model. The profiling of the algorithm execution on the GPU server with NVlink also verifies the model .

Acknowledgment. The article was prepared within the HSE University Basic Research Program. The research was partially supported by the resources of the supercomputer facilities provided by NRU HSE.

References

1. Abraham, M.J., et al.: GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX **1–2**, 19–25 (2015). https://doi.org/10.1016/j.softx.2015.06.001
2. Bastrakov, S., et al.: Particle-in-cell plasma simulation on heterogeneous cluster systems. J. Comput. Sci. **3**(6), 474–479 (2012). https://doi.org/10.1016/j.jocs.2012.08.012
3. Romero, J., Phillips, E., Ruetsch, G., Fatica, M., Spiga, F., Giannozzi, P.: A performance study of quantum ESPRESSO's PWscf code on multi-core and GPU systems. In: Jarvis, S., Wright, S., Hammond, S. (eds.) PMBS 2017. LNCS, vol. 10724, pp. 67–87. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72971-8_4

4. Spiga, F., Girotto, I.: phiGEMM: a CPU-GPU library for porting Quantum ESPRESSO on hybrid systems. In: 2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing, pp. 368–375 (2012). https://doi.org/10.1109/PDP.2012.72
5. Akimova, E., Misilov, V., Kulikov, I., Chernykh, I.: Hydrodynamical simulation of astrophysical flows: high-performance GPU implementation. J. Phys. Conf. Ser. **1336**, 012014 (2019). https://doi.org/10.1088/1742-6596/1336/1/012014/meta
6. Kulikov, I.: GPUPEGAS: a new GPU-accelerated hydrodynamic code for numerical simulations of interacting galaxies. Astrophys. J. Suppl. Ser. **214**(1), 12 (2014). https://doi.org/10.1088/0067-0049/214/1/12
7. Nikolskiy, V.P., Stegailov, V.V.: GPU acceleration of four-site water models in LAMMPS. In: Advances in Parallel Computing, vol. 36: Parallel Computing: Technology Trends, Proceedings of PARCO-2019, pp. 565–573 (2019). https://doi.org/10.3233/APC200086
8. Stegailov, V., et al.: Angara interconnect makes GPU-based Desmos supercomputer an efficient tool for molecular dynamics calculations. Int. J. High Perform. Comput. Appl. **33**(3), 507–521 (2019). https://doi.org/10.1177/1094342019826667
9. 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. **35**(4), 312–324 (2021). https://doi.org/10.1177/10943420211008288
10. Smirnov, G.S., Stegailov, V.V.: Efficiency of classical molecular dynamics algorithms on supercomputers. Math. Models Comput. Simul. **8**(6), 734–743 (2016). https://doi.org/10.1134/S2070048216060156
11. Morozov, I., Kazennov, A., Bystryi, R., Norman, G., Pisarev, V., Stegailov, V.: Molecular dynamics simulations of the relaxation processes in the condensed matter on GPUs. Comput. Phys. Commun. **182**(9), 1974–1978 (2011). https://doi.org/10.1016/j.cpc.2010.12.026
12. Anderson, J.A., Lorenz, C.D., Travesset, A.: General purpose molecular dynamics simulations fully implemented on graphics processing units. J. Comput. Phys. 227(10), 5342–5359 (2008). https://doi.org/10.1016/j.jcp.2008.01.047
13. Luehr, N., Ufimtsev, I.S., Martínez, T.J.: Dynamic precision for electron repulsion integral evaluation on graphical processing units (GPUs). J. Chem. Theory Comput. 7(4), 949–954 (2011). https://doi.org/10.1021/ct100701w
14. Rojek, K., Wyrzykowski, R., Kuczynski, L.: Systematic adaptation of stencil-based 3D MPDATA to GPU architectures. Concurr. Comput. **29**(9), e3970 (2017). https://doi.org/10.1002/cpe.3970
15. Dongarra, J., Pineau, J.F., Robert, Y., Vivien, F.: Matrix product on heterogeneous master-worker platforms. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 53–62 (2008). https://doi.org/10.1145/1345206.1345217
16. DeFlumere, A., Lastovetsky, A.: Searching for the optimal data partitioning shape for parallel matrix matrix multiplication on 3 heterogeneous processors. In: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, pp. 17–28. IEEE (2014). https://doi.org/10.1109/IPDPSW.2014.8
17. Rohr, D., Lindenstruth, V.: A flexible and portable large-scale DGEMM library for Linpack on next-generation multi-GPU systems. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 664–668. IEEE (2015). https://doi.org/10.1109/PDP.2015.89

18. Ryu, S., Kim, D.: Parallel huge matrix multiplication on a cluster with GPGPU accelerators. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 877–882. IEEE (2018). https://doi.org/10.1109/IPDPSW.2018.00139
19. Van De Geijn, R.A., Watts, J.: SUMMA: scalable universal matrix multiplication algorithm. Concurr. Pract. Exp. **9**(4), 255–274 (1997). https://doi.org/10.1002/(SICI)1096-9128(199704)9:4⟨255::AID-CPE250⟩3.0.CO;2-2
20. Goto, K., Geijn, R.A.v.d.: Anatomy of high-performance matrix multiplication. ACM Trans. Math. Softw. **34**(3), 12–1 - 12–25 (2008). https://doi.org/10.1145/1356052.1356053
21. 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. 24–1- -24–22. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3295500.3356181
22. Lai, P.W., Arafat, H., Elango, V., Sadayappan, P.: Accelerating Strassen-Winograd's matrix multiplication algorithm on GPUs. In: 20th Annual International Conference on High Performance Computing, pp. 139–148. IEEE (2013), https://doi.org/10.1109/HiPC.2013.6799109
23. 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, pp. 1–22 (2019). https://doi.org/10.1145/3295500.3356181
24. Tran, N.P., Lee, M., Choi, J.: Parameter based tuning model for optimizing performance on GPU. Cluster Comput. **20**(3), 2133–2142 (2017). https://doi.org/10.1007/s10586-017-1003-4
25. Zhang, L., Wahib, M., Zhang, H., Matsuoka, S.: A study of single and multi-device synchronization methods in Nvidia GPUs. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 483–493. IEEE (2020). https://doi.org/10.1109/IPDPS47924.2020.00057
26. Malik, T., Lastovetsky, A.: Towards optimal matrix partitioning for data parallel computing on a hybrid heterogeneous server. IEEE Access **9**, 17229–17244 (2021). https://doi.org/10.1109/ACCESS.2021.3052976
27. 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). https://doi.org/10.1109/ScalA49573.2019.00010
28. 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). https://doi.org/10.1109/GloSIC50886.2020.9267865
29. 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.) Communications in Computer and Information Science. Supercomputing, pp. 597–609. Springer, Cham (2019), https://doi.org/10.1007/978-3-030-36592-9_49

30. 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
31. Kelefouras, V., Kritikakou, A., Mporas, I., Kolonias, V.: A high-performance matrix-matrix multiplication methodology for CPU and GPU architectures. J. Supercomput. **72**(3), 804–844 (2016). https://doi.org/10.1007/s11227-015-1613-7
32. Li, X., Liang, Y., Yan, S., Jia, L., Li, Y.: A coordinated tiling and batching framework for efficient GEMM on GPUs. In: Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, pp. 229–241 (2019). https://doi.org/10.1145/3293883.3295734
33. Boyer, M., Meng, J., Kumaran, K.: Improving GPU performance prediction with data transfer modeling. In: 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum, pp. 1097–1106. IEEE (2013). https://doi.org/10.1109/IPDPSW.2013.236
34. Tang, H., Komatsu, K., Sato, M., Kobayashi, H.: Efficient mixed-precision tall-and-skinny matrix-matrix multiplication for GPUs. Int. J. Netw. Comput. **11**(2), 267–282 (2021). https://doi.org/10.15803/ijnc.11.2_267