Performance Comparison of Different OpenCL Implementations of LBM Simulation on Commodity Computer Hardware

Jelena TEKIC[2], Predrag TEKIC[1], Milos RACKOVIC[2]
[1]*Faculty of Information Technology, Vojvode Putnika 87, 21208 Sremska Kamenica, Serbia*
[2]*Faculty of Science, Trg Dositeja Obradovića 3, 21000 Novi Sad, Serbia*
radjenovic@uns.ac.rs

***Abstract*—Parallel programming is increasingly used to improve the performance of solving numerical methods used for scientific purposes. Numerical methods in the field of fluid dynamics require the calculation of a large number of operations per second. One of the methods that is easily parallelized and often used is the Lattice Boltzmann method (LBM). Today, it is possible to perform simulations of numerical methods not only on high performance computers (HPC) but also on commodity computers. In this paper is presented how to accelerate LBM implementation on commodity computers using characteristics of OpenCL specification. Simulation is executed simultaneously on multiple heterogeneous devices. Four different approaches for several commodity computer configurations are presented. Obtained results are compared for different types of commodity computers and advantages and disadvantages are discussed. In this paper it presented which LBM OpenCL code implementation, among four different presented, shows best simulation performance and should be used when solving similar CFD problems.**

***Index Terms*—Lattice Boltzmann methods, multicore processing, scientific computing, parallel programming, parallel algorithms.**

I. INTRODUCTION

For long time studying of numerical methods that require the calculation of a large number of operations per second was available only to researchers that had access to supercomputers and high performance computing (HPC), because of the low computational power of commodity computers CPU. Changes in architecture design of graphics cards processors (GPU) and processors (CPU) have enabled an increasing number of scientific researchers to study numerical methods using their personal (commodity) computers. Commodity computers nowadays usually have a couple of heterogeneous devices, putting in use all available devices can accelerate simulations of numerical methods.

OpenCL is a standard created for writing unify/portable programs for different hardware devices [1]. Implementing numerical methods using the OpenCL specification provides the ability to use all available devices on a single commodity computer to speed up the execution of numerical simulation.

The Lattice Boltzmann method (LBM) is one of computational fluid dynamics (CFD) methods for fluid simulation. LBM can simulate immense diversity of fluid flows [2-6]. Additionally LBM is appropriate for parallelization since leading equations of this method have local property [7-9]. Also, the method is very memory demanding and computationally intensive [10]. The benchmark problem of lid driven cavity is used to test developed algorithms in this paper. It is a classic CFD problem and it has been subject of comparison and validation in many theoretical models, numerical implementations, experimental calibrations and software implementations [11-15].

The main objective of this paper is accelerating OpenCL code that is executed simultaneously on multiple heterogeneous devices. Four different approaches are presented, each approach uses different OpenCL characteristics.

In the literature so far, the emphasis has always been on HPC devices/computers, in this research the emphasis is on commodity computers. The aim of this research is to show different LBM OpenCL simulation implementations that put in use all devices from different vendors that are part of one commodity computer platform and to achieve maximum simulation acceleration. The different characteristics of the devices used on commodity computers have led to the development of several algorithms presented in this paper. By selecting the appropriate algorithm for available devices the maximum acceleration of the simulation for the commodity computer is achieved.

All implementations are tested on several commodity computers configurations, results are compared and advantages and disadvantages are discussed for each approach for different configurations and vendors.

II. RELATED WORK

With development of multi core and many core architecture execution of LBM implementations is accelerated. LBM implementation that was executed on a GPU device was presented in [8]. Tölke and associate implemented LBM method using CUDA framework in 2008 [16]. They have shown that using GPU devices is more efficient than using CPU devices. Implementation done using CUDA framework can be executed only on NVIDIA devices, whereas implementation done using OpenCL specification can be executed on heterogeneous devices.

Multi GPU CUDA implementation of cavity flow is described in [17]. This implementation is tested on one node consisting of six Tesla C1060. D3Q19 lattice model and multi-relaxation-time (MRT) were used. In [18-19] OpenMP is used for parallelization of cavity flow implementation.

Digital Object Identifier 10.4316/AECE.2022.01008

D3Q19 model and MRT approximation were used. Implementation was tested for various depth–width aspect ratios on a single node multi GPU system, consisting of three nVIDIA M2070 devices or three NVIDIA GTX560 devices. In [20] CUDA and MPI were used for parallelization of LBM implementation for fluid flow through porous media. Moreover this research proposed optimization strategies based on the data structure and layout.

In [21] message passing interface (MPI) technique for GPU management for a cluster of GPUs was adopted. D3Q19 model and MRT approximation were used. Speed up of implementation for cavity flow was gained using overlapping of communication and computation. The flow around a sphere using D3Q19 model and MRT approximation was presented in [22]. CUDA and MPI library were used for parallelization. Partitioning method of solution domain or using the computation and the communication by multiple streams were used to reduce the size of communicational time. For testing of implementation a supercomputer equipped with 170 nodes of Tesla S1070 (680 GPUs) was used. Single-phase, multi-phase and multi-component LBM implementations on multi-GPU clusters using CUDA and OpenMP were presented in [23].

Tekic et al. have showed that implementation using OpenCL specification executed in parallel on GPU devices have better performances than sequential implementation executed on CPU devices [24]. Comparing performances of implementations done using CUDA and OpenCL it has been shown that LBM implementation using OpenCL can gain similar performances like implementation done using CUDA framework [25-26]. In 2018 Tekic et al. showed acceleration of execution of LBM simulation on more heterogeneous devices at the same time comparing to execution of LBM simulation on one heterogeneous device [27], results are also part of the J. Tekic PhD thesis "Optimization of CFD simulations on groups of many-core heterogeneous architectures" (in Serbian language).

III. Algorithm for LBM Implementation

Lattice Boltzmann method simulates the behavior of fluid flows by particle movement and collision. The algorithm and equations used to solve the Lattice Boltzmann method is presented in Fig. 1.

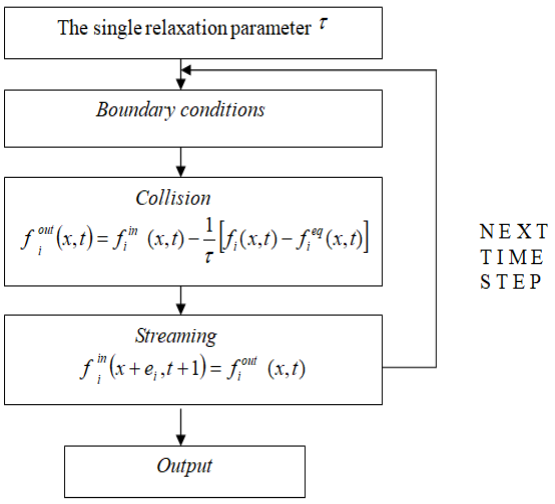

Figure 1. Algorithm for the Lattice Boltzmann method

At the beginning of the algorithm relaxation time -τ is given. Next step is calculation of boundary conditions. The bounce-back boundary conditions are used for the walls and the Equilibrium scheme [28-29] is used for the moving lid.

Next step is calculation of particle collision. Collision step describes collision with other particles in adjacent cells. f_i^{out} represents particle distribution function after collision, f_i distribution function, f_i^{eq} is equilibrium distribution function and

$$-\frac{1}{\tau}\left[f_i(x,t) - f_i^{eq}(x,t)\right] \quad (1)$$

represents the Bhatnagar–Gross–Krook (BGK) approximation [30] for the collision operator. Applied LBGK model is consistent with the Navier-Stokes equation [31].

Calculation of streaming step is the last part of one iteration of algorithm, f_i^{in} represents particle distribution function after streaming step and e_i is discrete velocity vector. The streaming step describes the transfer of information from one node to another.

IV. Implementation

This section describes the structure of different implementations for LBM. The implementation of boundary conditions and the Collision step is simple since there is no data dependence between the two iterations (time steps). Within the streaming step, the dependence of data between iterations occurs. The distribution function in the streaming step of the iteration (time step) t + 1 gets the value of the distribution function of the neighboring node from the previous iteration t, the data calculated in the current iteration depends on the data from the previous iteration. This dependency is solved by using an additional grid.

Lattice data are represented by D2Q9 lattice and accordingly 9 arrays for particle distribution functions are created. Also, to resolve the problem of data dependency between iterations additional arrays for particle distribution functions are created. Since OpenCL supports only one dimensional arrays all used arrays must be mapped to one dimensional arrays. For arrays that represent particle distribution functions the ArrayOfStructure scheme is used. This scheme is optimized for the collision step-this means that distribution functions arrays have all velocity vector values grouped for a single node.

Each implementation consists of two parts, first written in JAVA programming language that is executed on the HOST device and second written accordingly to OpenCL specification that is executed on OpenCL devices.

First part of implementation that is executed on the host device is similar for all 4 presented approaches. At the beginning of each implementation OpenCL objects needed for using OpenCL devices are created.

Frist step is creating one or more platforms depending of available hardware. More platforms are created if commodity computer consists of devices made by different vendors. One platform represents one OpenCL implementation, some vendors' support devices of third-party vendors by their implementations, however this implementations cannot take full advantage of third-party architectures. For each platform one context is created, also

for each used device one command Queue is created.

Last step is data initialization. One array is created for each particle distribution function, additional arrays are created to temporarily store the values and solve the problem of data dependencies between iterations, one array for data exchange between devices, size of segment and start point for the data segment processed on the observed device. The number of additional arrays depends on the type of implementation. Before kernel creation, data is split on segments that will be sent to compute devices and added to memory objects which are sent to appropriate compute devices. Memory objects are a wrapper around the initialized data, they send and distribute data to devices for parallel execution. At the end kernels are created and started. Different implementations have different numbers and structure of kernels.

A. Implementation 1

Implementation 1 consists of three kernels: Calculation, Exchange and Order, also it uses 8 additional global arrays for copies of particle distribution functions. This implementation is an improved version of implementation that is presented in paper [28].

Kernel Calculation calculates Boundary conditions, Collision step and Streaming step. Kernel Exchange collects data that will be exchanged between devices, data from all arrays that will be exchanged is saved in one array. After execution of kernel Exchange control is returned to the HOST device that exchanges data between devices. Last step is execution of kernel Order. This kernel adds exchanged data to appropriate arrays that represent particle distribution functions.

First difference between Implementation 1 and implementation that is presented in paper [28] is that data between original lattice and additional lattice are exchanged by direct copying on OpenCL devices in implementation from [28], whereas in the newer version of implementation direct copying is not used. When control is returned to HOST after kernel Exchange is done, next kernels are called with exchanged variables, on the place of variables that represent original lattice are placed variables that represent additional lattice and inversely on the place of variables that represent additional lattice are placed variables that represent original lattice.

Second difference between Implementation 1 and implementation that is presented in paper [28] is done for the part of implementation that executes data is exchange between devices. All data from different arrays is placed in only one array and one big array is exchanged between devices instead of exchanging six small arrays (one array for each function that needs to exchange data) like in previous implementation from [28]. Since all data is placed in one array, improved implementation is more complicated to operate but leads to improved simulation performance. OpenCL exchanges data placed in one large array faster than data placed in several smaller arrays.

B. Implementation 2

Implementation 2 consists of four kernels: Calculation, OrderByX, Exchange and Order, number of additional global arrays for copies of particle distribution functions is decreased to six.

Kernel Calculation calculate: Boundary conditions, Collision Step and Streaming step. Boundary conditions and Collision step are done the same as in Implementation 1. Streaming step is divided in two parts. In first part streaming by X axis is done using local memory. Local memory is faster but smaller than global memory and using local memory should accelerate the simulation. Two particle distribution functions perform streaming step only by X axis, for these functions streaming step is completed in this first part using local memory, and for these functions additional global arrays are not initialized and used. Second part of the streaming step is streaming by Y axis and it is done using 6 global additional arrays.

Since using local memory does not support sharing data between working groups, after the Streaming step global arrays are not in correct order by X axis. Next executed Kernel OrderByX will exchange data between work groups and put global arrays in correct order by X axis. Rest of Implementation 2 is the same as in Implementation 1. Kernel Exchange collects data for exchange between devices, on HOST device data is exchanged between used devices, and at the end of iteration exchanged data is placed in appropriate places to arrays for particle distribution functions by kernel Order.

C. Implementation 3

Implementation 3 consists of four kernels: Calculation, OrderByX, OrderByYAndExchange and Order. Additional global arrays for copies of particle distribution functions are not created.

Kernel Calculation performs Boundary conditions and Collision step the same way as in previously explained implementations. Streaming step is divided on two parts: first part is streaming by X axis and second part is streaming by Y axis. Data dependency is solved using variables created in faster local memory and additional global arrays are not created.

Since streaming is done using local memory global arrays are not in correct order. Kernel OrderByX will put global arrays in correct order by X axis, OrderByYAndExchange will put data in correct order by Y axis and after all global arrays are in correct order data will be added to array that will be exchanged between devices. These arrays (one array per compute device) will be exchanged by HOST device. Last step in the iteration is execution of kernel Order that will place exchanged data on their place in global arrays.

D. Implementation 4

Implementation 4 consists of four kernels: Calculation, StreamingEnd, Exchange and Order. Also number of additional global arrays for copies of particle distribution function is decreased to four. It is expected that for some configurations the decrease of number of used arrays variables speeds up the simulation even though the simulation uses one more kernel.

The fist part of Calculation kernel (Boundary condition and Collision) is the same as in previous implementations. Kernel Calculation performs the streaming for first four particle distribution functions. Next kernel StreamingEnd performs streaming for last four particle distribution functions. Dividing of streaming step on two parts allowed decrease of number of additional global arrays for particle

distribution functions but number of used kernels is increased by one compared to Implementation 1. Kernel Exchange create array for exchange between devices, HOST exchange the data and kernel Order places exchanged data on appropriate places in global arrays.

TABLE I. LIST OF CONFIGURATIONS USED FOR TESTING OF LBM IMPLEMENTATIONS

Configuration Number	Device name	Device type	Platform vendor	OpenCL Version	HasReadWriteCache
1	AMD Radeon R5 M430	GPU	AMD	1.2	true
	Intel Core i5-6200U	CPU	Intel	2.0	true
2	AMD Radeon R5 M420	GPU	AMD	2.0	true
	Intel Core i5-7200U	CPU	Intel	2.0	true
3	AMD Radeon R5 M430	GPU	AMD	1.2	true
	Intel HD Graphics 520	GPU	Intel	1.2	true
4	AMD Radeon R5 M420	GPU	AMD	1.2	true
	Intel HD Graphics 620	CPU	Intel	2.0	true
5	GeForce 940M	GPU	NVIDIA	1.2	true
	Intel Core i7-5500U	CPU	Intel	2.0	true
6	GeForce 940M	GPU	NVIDIA	1.2	true
	Intel HD Graphics 5500	GPU	Intel	2.0	true
7	GeForce GTX 960	GPU	NVIDIA	1.2	true
	GeForce GTX 960	GPU	NVIDIA	1.2	true
8	Tesla C2070	GPU	NVIDIA	1.1	true
	Tesla C2070	GPU	NVIDIA	1.1	true
9	AMD Radeon Pro 455	GPU	Apple	1.2	no
	Intel HD Graphics 530	GPU	Apple	1.2	no
10	Intel i9-9880H CPU	CPU	Apple	1.2	true
	AMD Radeon Pro 5500M	GPU	Apple	1.2	no
11	Intel UHD Graphics 630	GPU	Apple	1.2	no
	AMD Radeon Pro 5500M	GPU	Apple	1.2	No

V. RESULTS

Each of previously described implementation is tested on benchmark example lid driven cavity. Implementations are tested for Reynolds number (Re) of value 1000. Different grid sizes were used to analyze the results (1024^2, 1526^2, 2048^2, 2560^2, 3072^2, 3584^2, 4096^2 and 4608^2). Verification and comparison of simulation performance was performed based on the number of modified network nodes per second (MLUPS).

Configurations used for testing of implementations are given in Table 1. “One device implementation” is tested on a “computing device” with better performances for each configuration. When more compute devices were used for simulation the execution lattice was divided in segments, each segment was calculated on one compute device.

The results of solving the LBM simulation for the Reynolds number 1000 obtained using the implementations described in this paper were compared with the results from the literature and a satisfactory match was obtained. In Fig. 2, the mid-planes of the velocities along the x and y axes are shown. The results shown in Fig. 2 were calculated on Configuration 2. For other configurations and all variations of the implementation, the correctness of the results was validated by comparing the graphical representations of the velocity mid-planes as well as the graphical results of the streamlines. The results obtained for the velocity mid-planes are identical to those shown in Fig. 2 and they are not additionally shown in separate images.

When two different compute devices with different performances were used, the lattice was divided into segments of unequal sizes. Sizes of lattice segments were determined empirically. Using appropriately divided lattice accelerates the simulation execution and hides the difference in performances of used devices. The size of each segment was the same when identical compute devices were used.

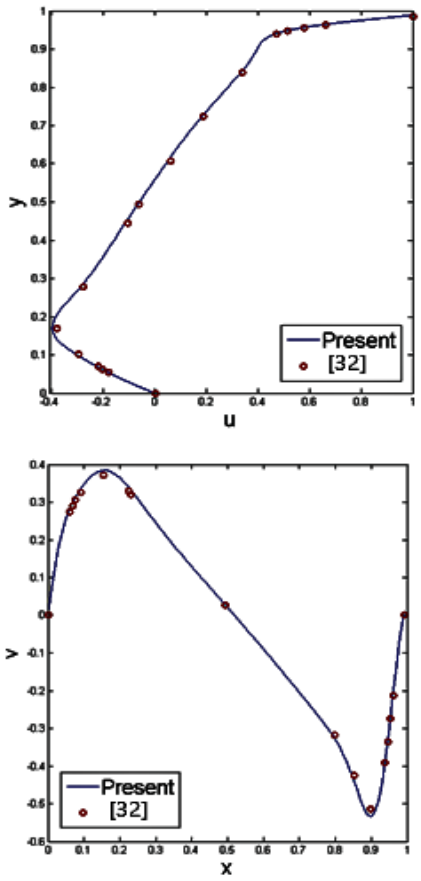

Figure 2. Vertical velocities for y = 0.5 for Re = 1000 (above) and Horizontal velocities for x = 0.5 for Re = 1000 (below) - the circles represent the data of Ghia et al., 1982 [32], and the line represents the result of parallel implementation in OpenCL on multiple heterogeneous devices

When configuration consisted of dedicated GPU and CPU device, or consisted of dedicated GPU and integrated GPU device, the size of lattice segment sent to dedicated GPU was significantly larger (for example the size was 80:20

percent ratio or even 90:10 percent ratio) but still the acceleration of the simulation was significant. Acceleration achieved for the configurations that had to use different platforms for each device was from 52% to 94%. In the case when all devices could use one platform gained acceleration was from 80% to 221%. Achieved acceleration for identical compute devices was 94% for Configuration 7 and 159% for Configuration 8.

Implementations described in this paper showed significant improvement of simulation speed compared to "single device implementation" and to implementations presented in [28]. Also it is noticed that improvement of performance for all implementations and configurations is more significant for larger lattices.

The improvement of simulation speed for different configurations in [28] was from 33.5% to 53% compared to "single device implementation". In this research acceleration was from 52% (simulation executed using two OpenCL platforms) to 221% (simulation executed using one OpenCL platform) and average improvement of acceleration was around 120% compared to "single device implementation".

Configuration 1 and Configuration 2 consist from AMD GPU and Intel processor. These configurations have two devices from different vendors and each vendor has its own platform for OpenCL. Since AMD OpenCL platform supports Intel processors both configurations can use one or two platforms/contexts. Results of simulation executions are presented on Fig. 3. Based on the presented results, it can be concluded that the newer version of Implementation 1 shows significant improvement of performance compared to Implementation 1a (presented at Tekić et al. 2018) [28].

Using one big array for exchange of data instead of a couple of smaller arrays and exchanging variables in calls of kernels instead of directly copying values from one variable to another on OpenCL devices significantly accelerates the speed of the simulations.

More over from Fig. 3, can be concluded that the simulation using one context gains better performances than the simulation that uses two contexts. Time needed for communication between two devices, if two platforms are used, decreases the performance of the simulation more than use of OpenCL implementation from third party vendor for a second (slower) device.

Configurations 3 and 4 contain AMD GPU and Intel integrated GPU. Since AMD platform does not support GPUs from other vendors these kinds of devices are tested by implementations that create two platforms and two contexts. The results obtained by testing these configurations are presented in Fig. 4. This type of configuration gains the best performance using Implementation 4. Decreased number of variables speeds up the simulation that runs on configurations which must use two OpenCL platforms. Implementation 4 showed better results although one more kernel is called and control is returned to the HOST one more time.

Configurations that have AMD devices have low performance for Implementation 2 and Implementation 3 that use local memory. Implementation 3 that uses local memory intensively shows very low performance on configurations that contain an AMD GPU. Time needed for simulation execution on two devices for Implementation 3 on these configurations is longer than the time needed for simulation execution on only one (faster) device. Because Implementation 3 has very poor performance when it is used on configurations with AMD devices results for this implementation are not shown in Fig. 3 and Fig. 4.

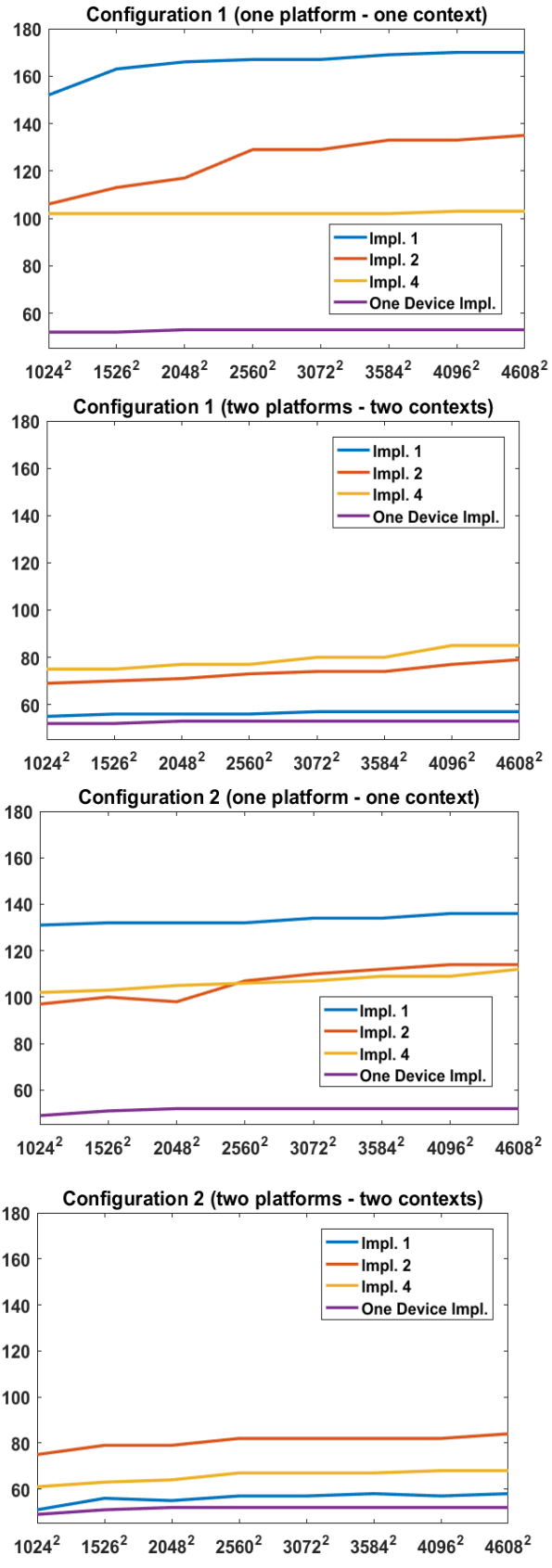

Figure 3. Results of simulation executions for Configuration 1 and Configuration 2

Configurations that contain NVIDIA GPU are presented

at Fig. 5. Configurations 5 contain NVIDIA GPU and Intel processor, Configuration 6 contain NVIDIA and INTEL integrated GPU. The best choice for this configuration is Implementation 3 that uses local memory for streaming by X and Y axis. Configuration 7 contains two NVIDIA GTX 960. The best performance for this implementation has Implementation 1. Configuration 8 contains two TESLA GPU. The best performance is achieved with Implementation 2 that uses local memory for streaming by X axis.

Apple has its platform for OpenCL that supports devices from different vendors. Results for devices that use Apple platform for OpenCL are presented on Fig. 6. GPU devices that are used on this Apple's platform don't support ReadWriteCache, Implementation 2 and Implementation 3 cannot be tested on these devices.

It can be concluded that simulation that uses Implementation 1 gains significant improvement of performance when two devices are used compared to the performance of the simulation executed on only one device. Implementation 4 that uses two devices at the same time also has better performance than simulation executed on only one device. However Implementation 4 does not have as good performance as Implementation 1 when only one context is used.

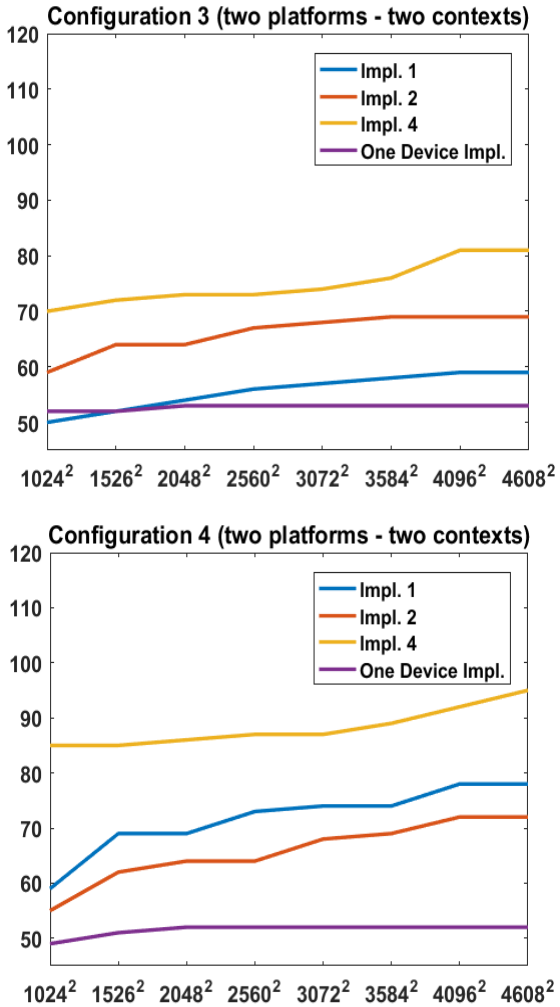

Figure 4. Results of simulation executions for Configuration 3 and Configuration 4

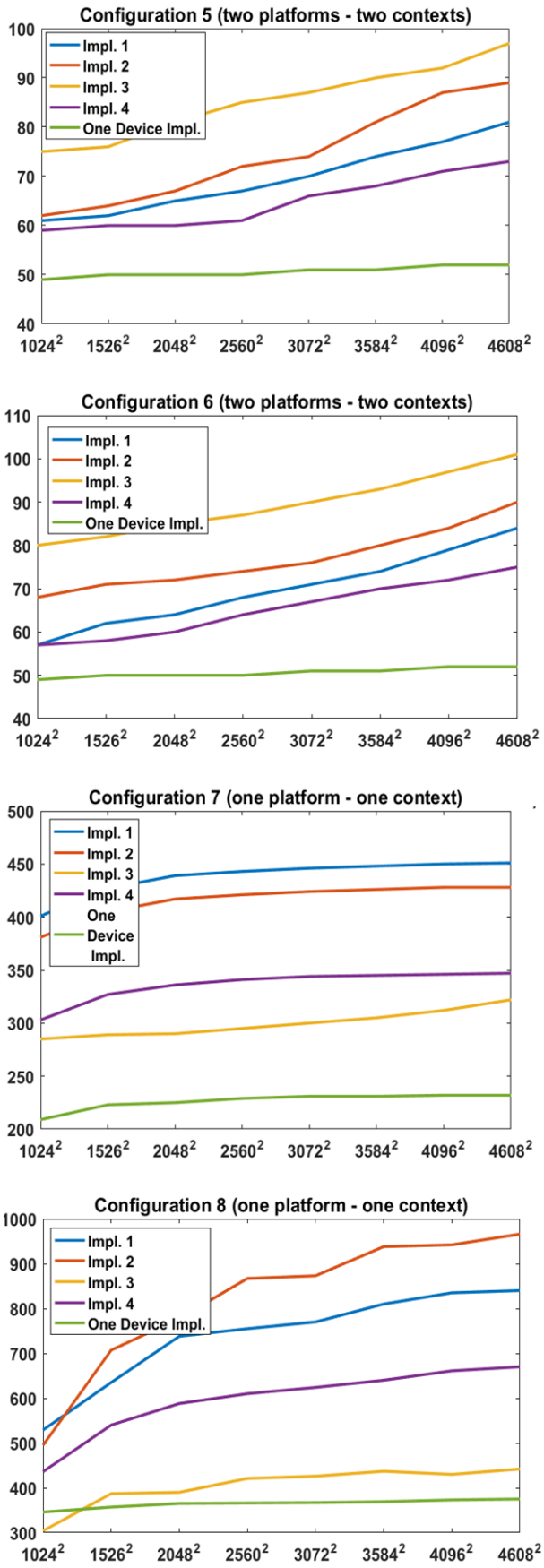

Figure 5. Results of simulation executions for Configuration 5, Configuration 6, Configuration 7 and Configuration 8

Presented results show that algorithms that put in use local memory are better solutions on NVIDIA graphics cards. When a newer generation of processors (e.g. Intel i7)

was used to run the HOST program, the best choice is an algorithm that uses two local counters. If an older generation processor (e.g. Intel i5) is used for the HOST device, it is better to use an algorithm with one local counter.

AMD graphics card platforms cannot make good use of local memory, so it is better for AMD platforms to use a basic algorithm or a basic algorithm with a reduced number of variables.

An algorithm with a reduced number of variables has proven to be a good choice if there is an AMD platform in combination with another platform, so it is necessary to form two contexts.

For commodity computers that have devices from different vendors, if possible, only one platform should be used to run the simulation. The achieved speeds will be higher if only one platform is used, even if the performance of one device decreases due to the implementation of another vendor platform. Data transfer between two platforms is complicated and affects the speed more than selecting a third party platform vendor. If it is necessary to use two contexts, an algorithm with a reduced number of auxiliary variables should be selected.

The performance of simulation execution on multiple heterogeneous multi-core devices depends not only on the performance of the devices used for parallel execution but also on the performance of the HOST device. After testing on different types of HOST devices, it can be concluded that by switching to newer generation processors (Intel i7 and higher) it is expected that the best performance will be achieved by an algorithm that uses local memory and two local variables. Also, it is expected that other manufacturers will improve the work with local memory and that on their devices in the future the algorithm that uses local memory and two local variables will have the best performance. Most current vendors have not yet implemented OpenCL 2.0. The implementation of OpenCL 2.0, followed by OpL 2.1 and OpenCL 2.2 should also contribute to improving simulation execution performance.

VI. Conclusion

In this paper is presented how general purpose LBM simulation can be accelerated on commodity computers using characteristics of OpenCL on different hardware configurations. Eleven configurations were used for testing of presented implementations. It has been shown that acceleration of simulation can be achieved in several ways.

It has been shown that swapping of variables (that represent ghost and original lattice) in calls of kernels when control is returned to the host is more efficient than direct copying from one lattice to another inside kernels executed on OpenCL devices. Also using one large array for data exchange between OpenCL devices instead of a couple of smaller arrays improves the performance of the simulation. Moreover it is described how the streaming step of LBM method can be calculated faster using local memory of OpenCL devices. Use of local memory accelerated simulation on configurations that support the use of local memory.

It is noticed that when configuration consists of devices from different vendors it is more efficient to use one context and platform if it possible. The speed of data transfer between devices that use different platforms affects the simulation performance more significantly than using a third party vendor OpenCL platform for one device.

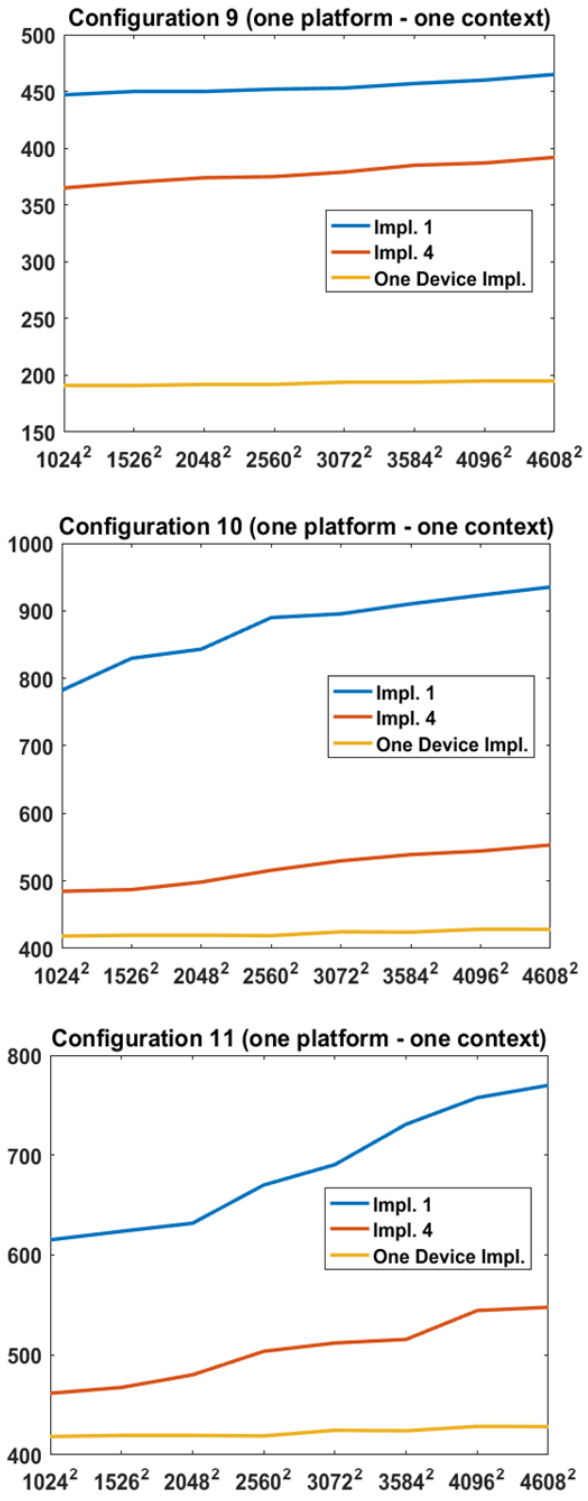

Figure 6. Results of simulation executions for Configuration 9, Configuration 10 and Configuration 11

Based on the research, four different OpenCL implementations have been created, presented, analyzed and tested on eleven different commodity hardware platforms. The obtained results have been discussed and it has been concluded that simulation of the Lattice Boltzmann method can be significantly accelerated using one of four proposed implementations. Each of the implementations has certain advantages on different types of

configurations and choosing appropriate implementation can significantly accelerate the LB simulation.

ACKNOWLEDGMENT

The authors acknowledge financial support of the Ministry of Education, Science and Technological Development of the Republic of Serbia (Grant No. 451-03-68/2022-14/200125).

REFERENCES

[1] P. L. Alvarez, S. Yamagiwa, "Invitation to OpenCL (Published Conference Proceedings style)," in Proc. Conference: Second International Conference on Networking and Computing, ICNC 2011, Osaka, Japan, November 30 - December 2, 2011, pp. 8-16. doi:10.1109/ICNC.2011.12

[2] W. S. R. M. Weiping Shi, "Finite-difference-based Lattice Boltzmann method for inviscid compressible flows," Numerical Heat Transfer, Part B: Fundamentals, vol. 40, pp. 1-21, no. 1, 2001. doi:10.1080/104077901300233578

[3] R. Mei, W. Shyy, D. Yu, L.-S. Luo, "Lattice Boltzmann method for 3-D flows with curved boundary," Journal of Computational Physics, vol. 161, pp. 680-699, no. 2, 2000. doi:10.1006/jcph.2000.6522

[4] Z. Guo, T. S. Zhao, "A Lattice Boltzmann model for convective heat transfer in porous media," Numerical Heat Transfer, Part B: Fundamentals, vol. 47, pp. 157-177, no. 2, 2005. doi:10.1080/10407790590883405

[5] P. M. Tekić, J. B. Rađenović, N. Lj. Lukić., S. S. Popović, "Lattice Boltzmann simulation of two-sided lid-driven flow in a staggered cavity," Int. J. Comput. Fluid Dyn., vol. 24, pp. 383–390, no. 9, 2010. doi:10.1080/10618562.2010.539974

[6] N. Lukić,, P. Tekic, J. Radjenovic, I. Sijacki, "Lattice Boltzmann simulation of two-sided lid-driven flow in deep cavities," Acta Periodica Technologica, pp. 157-168, 2015. doi:10.2298/APT1546157L

[7] S. Tomov, M. McGuigan, R. Bennett, G. Smith, J. Spiletic, "Benchmarking and implementation of probability-based simulations on programmable graphics cards," Computers & Graphics, vol. 29, pp. 71-80, no. 1, 2005. doi:doi.org/10.1016/j.cag.2004.11.008

[8] W. Li, X. Wei, A. Kaufman, "Implementing lattice Boltzmann computation on graphics hardware," The Visual Computer, vol. 19, pp. 444-456, no. 7, 2003. doi:10.1007/s00371-003-0210-6

[9] D. Vidal, R. Roy, F. Bertrand, "A parallel workload balanced and memory efficient Lattice-Boltzmann algorithm with single unit BGK relaxation time for laminar Newtonian flows," Computers & Fluids, vol. 39, pp. 1411-1423, no. 8, 2010. doi:10.1007/s00371-003-0210-6

[10] K. Mattila, J. Hyvaluoma, J. Timonen, T. Rossi, "Comparison of implementations of the Lattice-Boltzmann method," Computers & Mathematics with Applications, vol. 55, pp. 1514-1524, no. 7, 2008. doi:10.1016/j.camwa.2007.08.001

[11] G. K. Batchelor, "On steady laminar flow with closed streamlines at large Reynolds number," Journal of Fluid Mechanics, vol 1, pp. 177-190 , 1956. doi:10.1017/S0022112056000123

[12] F. Pan, A, Acrivos, "Steady flows in rectangular cavitie," Journal of Fluid Mechanics," vol. 28, pp. 643-655, 1967 doi:10.1017/S002211206700237X

[13] A. S. Benjamin, V. E. Denny, "On the convergence of numerical solutions for 2-D flows in a cavity at large Re," Journal of Computational Physics, vol. 33, pp. 340-358, 1979. doi:10.1016/0021-9991(79)90160-8

[14] P. N. Shankar, M. D. Deshpande, "Fluid Mechanics in the Driven Cavity," Annual Review of Fluid Mechanics, vol. 32, no.1 , pp. 93-136, 2000. doi:10.1146/annurev.fluid.32.1.93

[15] C.-H. Bruneau, M. Saad, "The 2D lid-driven cavity problem revisited. Computers & Fluids," vol. 35, no. 3, pp. 326-348, 2006. doi:10.1016/j.compfluid.2004.12.004

[16] J. Tölke, "Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by NVIDIA," Computing and Visualization in Science, vol. 13, no. 29, 2010. doi:10.1007/s00791-008-0120-2

[17] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, "Multi-GPU implementation of the Lattice Boltzmann method," Computers & Mathematics with Applications, vol. 65, pp. 252-261, no. 2, 2013. doi:10.1016/j.camwa.2011.02.020

[18] H.-W. Chang, P.-Y. Hong, L.-S. Lin, C.-A. Lin, "Simulations of three-dimensional cavity flows with multi relaxation time Lattice Boltzmann method and graphic processing units," Procedia Engineering, vol. 61, pp. 94-99, no. 2013. doi:10.1016/j.proeng.2013.07.099

[19] H.-W. Chang, P.-Y. Hong, L.-S. Lin, C.-A. Lin, "Simulations of flow instability in three dimensional deep cavities with multi relaxation time Lattice Boltzmann method on graphic processing units," Computers & Fluids, vol. 88, pp. 866-871, no. 2013. doi:10.1016/j.compfluid.2013.08.019

[20] C. Huang, B. Shi, N. He, Z. Chai, "Implementation of Multi-GPU based Lattice Boltzmann method for flow through porous media," Advances in Applied Mathematics and Mechanics, vol. 7, pp. 1-12, no. 1, 2015. doi:10.4208/aamm.2014.m468

[21] P.-Y. Hong, L.-M. Huang, L.-S. Lin, C.-A. Lin, "Scalable multi-relaxation-time Lattice Boltzmann simulations on multi-GPU cluster," Computers & Fluids, vol. 110, pp. 1-8, no. 2015. doi:10.1016/j.compfluid.2014.12.010

[22] W. Xian, A. Takayuki, "Multi-GPU performance of incompressible flow computation by Lattice Boltzmann method on GPU cluster," Parallel Computing, vol. 37, pp. 521-535, no. 9, 2011. doi:10.1016/j.parco.2011.02.007

[23] B. Massimo, F. Massimiliano, M. Simone, S. Sauro, K. Efthimios, "A flexible high-performance Lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries," Concurrency and Computation: Practice and Experience, vol. 22, pp. 1-14, no. 1, 2010.doi: 10.1002/cpe.v22:1

[24] P. M. Tekic, J. B. Radjenovic, M. Rackovic, "Implementation of the Lattice Boltzmann Method on heterogeneous hardware and platforms using OpenCL, " Advances in Electrical and Computer Engineering, vol. 12, no. 1, pp. 51-56, 2012. doi: 10.4316/AECE.2012.01009

[25] E. Calore, S.F. Schifano, R. Tripiccione, "A Portable OpenCL Lattice Boltzmann code for multi- and many-core processor architectures," Procedia Computer Science, vol. 29, pp. 40-49, 2014. doi:10.1016/j.procs.2014.05.004

[26] S. McIntosh-Smith, D. Curran, "Evaluation of a performance portable Lattice Boltzmann code using OpenCL," in Proc. Conference: International Workshop on OpenCL 2013 & 2014. Association for Computing Machinery, New York, NY, USA, 2014. doi:10.1145/2664666.2664668

[27] J. B. Tekic, P. M. Tekic, M. Rackovic, "Lattice Boltzmann method implementation on multiple devices using OpenCL," Advances in Electrical and Computer Engineering, vol. 3, pp. 3-8, 2018. doi:10.4316/AECE.2018.03001

[28] D. V. Patil, K. N. Lakshmisha, B. Rogg, "Lattice Boltzmann simulation of lid-driven flow in deep cavities," Computers & Fluids, vol. 35, pp. 1116-1125, no. 10, 2006. doi:10.1016/j.compfluid.2005.06.006

[29] S. Hou, Q. Zou,S. Chen, G. Doolen, A. C. Cogley, "Simulation of cavity flow by the Lattice Boltzmann method," Journal of Computational Physics, vol. 118, pp. 329-347, no. 2, 1995. doi:doi.org/10.1006/jcph.1995.1103

[30] P. L. Bhatnagar, E. P. Gross, M. Krook, "A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems," Physical Review, vol. 94, pp. 511-525, no. 3, 1954. doi:10.1103/PhysRev.94.511

[31] X, He, L. Luo, "Theory of the Lattice Boltzmann method: From the Boltzmann equation to the Lattice Boltzmann equation," Physical Review, vol. 56, pp. 6811-6817, no. 6, 1997. doi:10.1103/PhysRevE.56.6811

[32] U. Ghia, K. N. Ghia, C. T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," Journal of Computational Physics, vol. 48, pp. 387-411, 1982. doi:10.1016/0021-9991(82)90058-4