DOI: 10.15439/2015F305
ACSIS, Vol. 5

Innovative GPU accelerated algorithm for fast minimum convex hulls computation

Artem Potebnia
Email: potebnia@mail.ua
Kyiv National Taras Shevchenko University
in Kyiv, Ukraine

Sergiy Pogorilyy
Email: sdp@univ.net.ua
Kyiv National Taras Shevchenko University
in Kyiv, Ukraine

***Abstract*—Innovative algorithm for forming graph minimum convex hulls using the GPU is proposed. High speed and linear complexity of this method are achieved by distribution of the graph's vertices into separate units and their filtering. The key factor for improving the performance of innovative algorithm is the massively-parallel implementation of local hulls formation using video accelerators. A computational process is controlled by means of auxiliary matrices. A number of experimental studies of the algorithm have been carried out, and its suitability for application in the hull processing for large-scale problems has been demonstrated. The speed of the new method is 10 – 20 times higher compared to using functions of the professional mathematical package Wolfram Mathematica.**

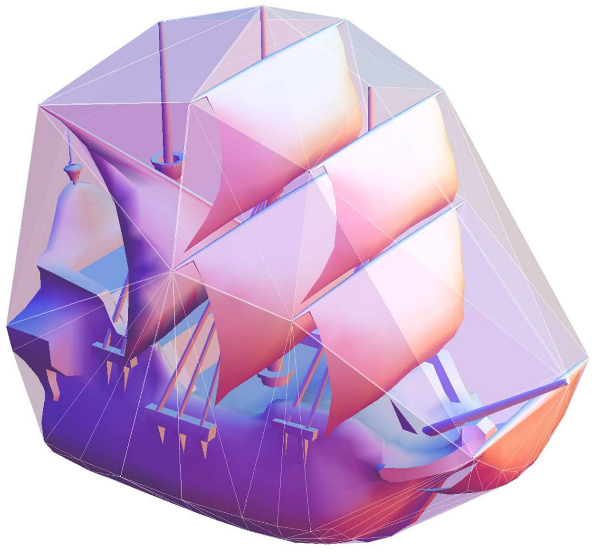
***Index Terms*—Minimum convex hull, CPU+GPU hybrid system, GPGPU technology, High-performance computing, CUDA, Graph**

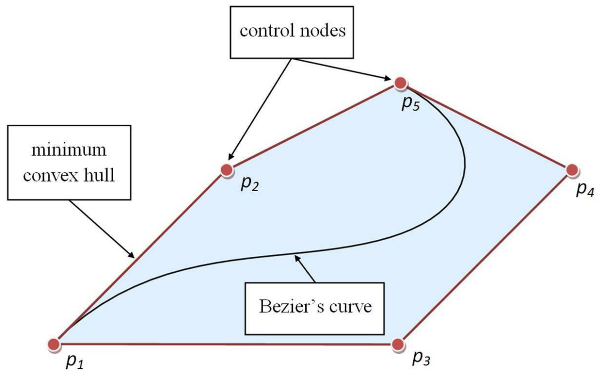
I. Introduction

FINDING the minimum convex hull (MCH) of the graph's vertices is a fundamental problem in many areas of modern research [7]. The solution of this task involves the formation of the minimum convex set containing all the nodes present in the graph (Fig. 1a). It is known that MCH is a common tool in computer-aided design and computer graphics packages [21]. For example, Bezier's curves used in *Adobe Photoshop*, *GIMP* and *CorelDraw* for modeling smooth lines fully lie in the convex hull of their control nodes (Fig. 1b). This feature greatly simplifies finding the points of intersection between curves and allows their transformation (moving, scaling, rotating, etc.) by appropriate control nodes [23]. The formation of some fonts and animation effects in the *Adobe Flash* package also uses splines composed of quadratic Bezier's curves [8].

It should be noted that convex hulls are used in *Geographical Information Systems* and routing algorithms in determining the optimal ways for avoiding obstacles. The paper [1] offers the methods for solving complex optimization problems using them.

Last decades are associated with rapid data volume growth in research processed by the information systems [19]. According to IBM, about 15 petabytes of new information are created daily in the world [14]. Therefore, in modern science, there is a separate area called *Big Data* related to the study of large data sets [13]. However, most of the known algorithms for MCH construction have time complexity $O(nlogn)$, making them useless when forming solutions to large-scale graphs. Therefore, there is a need to develop efficient algorithms with the complexity close to linear $O(n)$.


(a)



(b)

Fig. 1. Examples of the minimum convex hulls

TABLE I
COMPARISON OF THE COMMON ALGORITHMS FOR MCH CONSTRUCTION

Algorithm	**Complexity**	**Parallel versions**	**Multidimensional cases**
Jarvis's march	$O(nh)$, where h is a number of points on MCH	+	+
Graham's Scan	$O(nlogn)$	-	-
QuickHull	$O(nlogn)$, in the worst case – $O(n^2)$	+	+
Divide and Conquer	$O(nlogn)$	+	+

It is known that Wolfram Mathematica is one of the most powerful mathematical tools for the high performance computing. Features of this package encapsulate a number of algorithms and, depending on the input parameters of the problem, select the most productive ones [20]. Therefore, Wolfram Mathematica 9.0 is used to track the performance of the algorithm proposed in this article.

In recent years, CPU+GPU hybrid systems (GPGPU technology) allowing for a significant acceleration of computations have become widespread. Unlike CPU, consisting of several cores, the graphics processor is a multicore structure and the number of its components is measured in hundreds [16]. In this case, the sequential steps of algorithm are executed on the CPU, while its parallel parts are implemented on the GPU [20]. For example, the latest generation of NVIDIA Fermi GPUs contains 512 computing cores, allowing for the introduction of new algorithms with large-scale parallelism [9]. Thus, the usage of NVIDIA GPU ensures the conversion of standard workstations to powerful supercomputers with cluster performance [17].

The paper goal is to develop a high-speed algorithm for finding the minimum convex hulls using GPU. The time complexity of the proposed method is close to linear. The optimal values of algorithm's parameters, with which the usage of GPU resources is the most effective, are established in this paper.

II. A REVIEW OF ALGORITHMS FOR FINDING THE MINIMUM CONVEX HULLS

Despite intensive research, which lasted for the past 40 years, the problem of developing efficient algorithms for MCH formation is still open. The main achievement is the development of numerous methods based on the extreme points determination of the original graph and the link establishment among them [6]. These techniques include the *Jarvis's march* [15], *Graham's Scan* [11], *QuickHull* [4], *Divide and Conquer* algorithm and many others. The main features of their practical usage are given in Table I.

For parallelization the *Divide and Conquer* algorithm is the most suitable. It provides a random division of the original vertex set into subsets, formation of partial solutions and their connection to the general hull [21]. Although the hull connection phase has linear complexity, it leads to a significant slowdown of the algorithm, and as a result, to the unsuitability of its application in the hull processing for large-scale graphs.

Chan's algorithm, which is a combination of slower algorithms, has the lowest time complexity $O(nlogh)$. However, it can work by the known number of vertices contained in the hull [3]. Therefore, currently, its usage in practice is limited [5].

Study [2] gives a variety of acceleration tools for known MCH formation algorithms by cutting off the graph's vertices falling inside an octagon or rectangle and appropriate reducing the dimensionality of the original problem. The paper [12] suggests numerous methods of convex hull approximate formation, which have linear complexity. Such algorithms are widely used for tasks where speed is a critical parameter. But linearithmic time complexity of the fastest exact algorithms demonstrates the need for the introduction of new high-speed methods of convex hulls formation for large-scale graphs.

III. INNOVATIVE ALGORITHM FOR FORMATION OF THE CONVEX HULLS

We shall consider non-oriented graph $G = (V, E)$. The new algorithm provides a division of the original graph's vertex set into a set of output units $U = \langle U_1, U_2, ..., U_n \rangle$, $U_i \subseteq V$. However, unlike the *Divide and Conquer* method, this division is not random, but it is based on the spatial distribution of vertices. All nodes of the graph should be distributed by the formed subsets, i.e. $\bigcup\limits_{i=1}^{n} U_i = V$. This allows the presence of empty units, which do not contain vertices. Additionally, the condition of orthogonality division is met, i.e. one vertex cannot be a part of the different blocks: $U_i \cap U_j = \varnothing$, $\forall i \neq j$. Fig. 2a shows an example of division taking into account the above requirements.

The next stage of the proposed algorithm involves the formation of an auxiliary matrix based on the distribution of nodes by units. The purpose of this procedure is the primary filtration of the graph's vertices, which provides a significant decrease in the original problem dimensionality. In addition, the following matrices define the sets of blocks for the calculation in the subsequent stages of the algorithm and the sequence of their connection to the overall result. An auxiliary matrix formation involves the following operations:

1) Each block of the original graph must be mapped to one cell of the supporting matrix. Accordingly, the dimension of this matrix is $n \times m$, where n and m are the numbers of blocks allocated by the relevant directions.
2) The following operations provide the necessary coding of matrix's cells. Thus, the value of cell $c_{i,j}$ is zero if the corresponding block $U_{i,j}$ of original graph contains no vertices. Coding of blocks that contain extreme nodes (the highest, rightmost, lowest and leftmost points) of

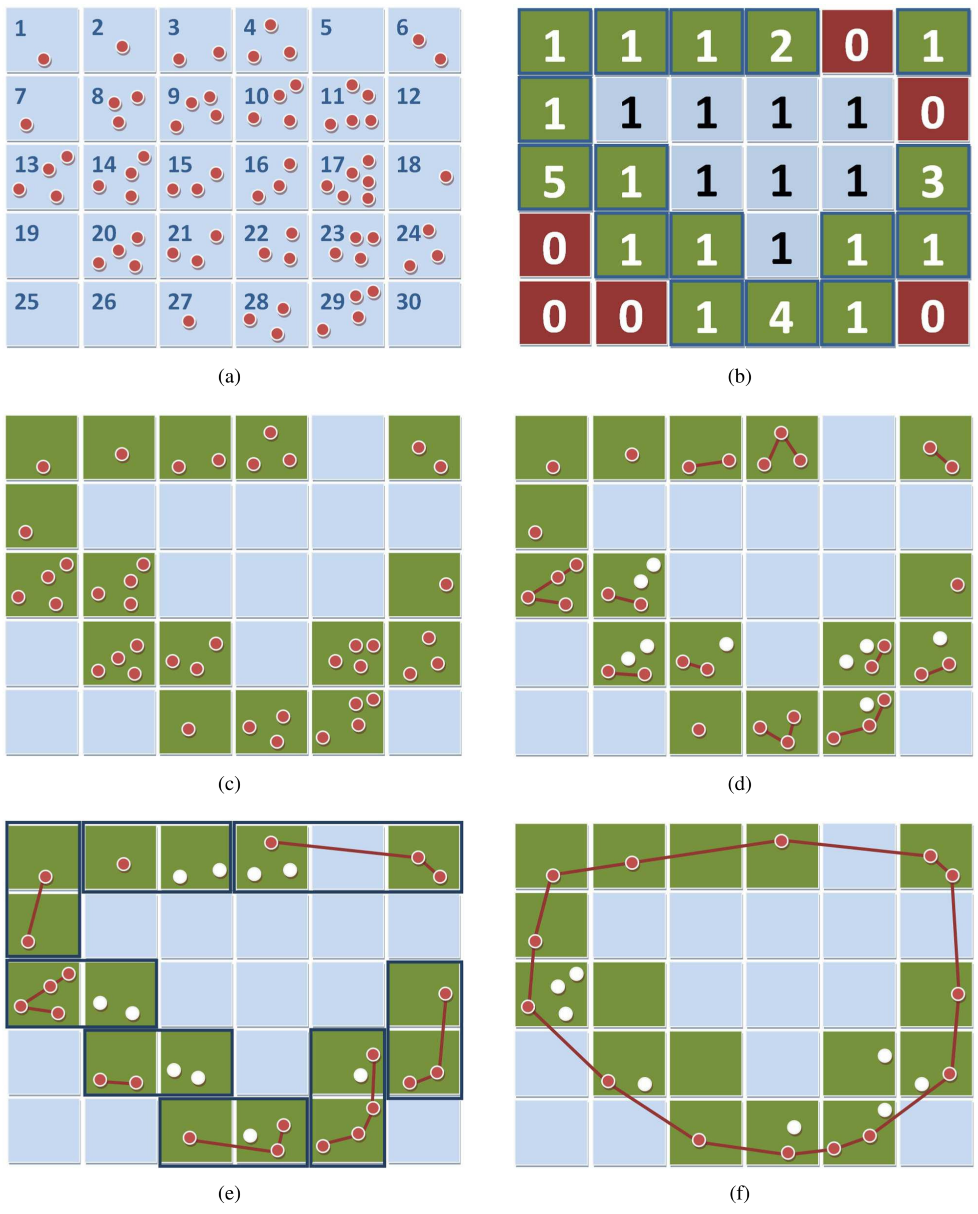


Fig. 2. Example of the algorithm execution

a given set is important for the algorithm. Appropriate cells are filled with numbers from 2 to 5. Other units that are filled, and contain no extreme peaks, shall be coded with ones in auxiliary matrix.

3) Further, primary filtration of allocated blocks is carried out using the filled matrix. Empty subsets thus shall be excluded from consideration. Blocks containing extreme vertices shall determine the graph division into parts for which the filtration procedure is applied. We shall consider the example of the block selection for the section limited with cells 2 – 3. If $c_{i,j} = 2$, then the next non-zero cell is searched by successive increasing of j. In their absence, the next matrix's row $i+1$ is reviewed. Selection of blocks is completed, if the value of another chosen cell is $c_{i,j} = 3$. The study of the other graph's parts is based on a similar principle.

Partial solutions are formed for selected blocks. Such operations require the formation of fragments rather than full-scale hulls that provides secondary filtration of the graph's vertices. The last step of the algorithm involves the connection of partial solutions to the overall result. Thus the sequential merging of local fragments is done on a principle similar to *Jarvis's march*. It should be noted that at this stage filtration mechanism leads to a significant reduction in the dimensionality of the original problem. Therefore, when processing the hulls for large graphs, combination operations constitute about 0.1% of the algorithm total operation time.

We shall consider the example of this algorithm execution. Let the set of the original graph's vertices have undergone division into 30 blocks (Fig. 2a). Auxiliary matrix calculated for this case is given in Fig. 2b. After application of primary filtration, only 57% of the graph's nodes were selected for investigation at the following stages of the algorithm (Fig. 2c). The next operations require the establishment of local hulls (Fig. 2d) and their aggregations are given in Fig. 2e. After performing of pairwise connections, this operation is applied repeatedly until a global convex hull is obtained (Fig. 2f).

IV. The development of hybrid CPU-GPU algorithm

We know that the video cards have much greater processing power compared to the central processing elements. GPU computing cores work simultaneously, enabling to use them to solve problems with the large volume of data. CUDA (*Compute Unified Device Architecture*), the technology created by NVIDIA, is designed to increase the productivity of conventional computers through the usage of video processors computing power [22].

CUDA architecture is based on SIMD (*Single Instruction Multiple Data*) concept, which provides the possibility to process the given set of data via one function. Programming model provides for consolidation of threads into blocks, and blocks – into a grid, which is performed simultaneously. Accordingly, the key to effective usage of GPU hardware capabilities is algorithm parallelization into hundreds of blocks performing independent calculations on the video card [24].

It is known that GPU consists of several clusters. Each of them has a texture unit and two streaming multiprocessors, each containing 8 computing devices and 2 superfunctional units [10]. In addition, multiprocessors have their own distributed memory resources (16 KB) that can be used as a programmable cache to reduce delays in data accessing by computing units [22]. From these features of CUDA architecture, it may be concluded that it is necessary to implement massively-parallel parts of the algorithm on the video cards, while sequential instructions must be executed on the CPU. Accordingly, the stage of partial solutions formation is suitable for implementation on the GPU since the operations for each of the numerous blocks are carried out independently.

It is known that function designed for executing on the GPU is called a kernel. The kernel of the innovative algorithm contains a set of instructions to create a local hull of any selected subset. In this case, distinguishing between the individual subtasks is realized only by means of the current thread's number. Thus, the developed hybrid algorithm has the following execution stages:

1) Auxiliary matrix is calculated on the CPU. The program sends cells' indexes that have passed the primary filtration procedure and corresponding sets of vertices to the video card.
2) Based on the received information, particular solutions are formed on the GPU, recorded to its global memory and sent to the CPU.
3) Further, the procedure of their merging is carried out and the overall result is obtained.

It should be noted that an important drawback of hybrid algorithms is the need to copy data from the CPU to the GPU and vice versa, which leads to significant time delays [16], [18]. Communication costs are considerably reduced by means of filtration procedure.

When developing high-performance algorithms for the GPU it is important to organize the correct usage of the memory resources. It is known that data storage in the global video memory is associated with significant delays in several hundred GPU cycles. Therefore, in the developed algorithm, the global memory is used only as a means of communication between the processor and video card. The results of intermediate calculations for each of the threads are recorded in the shared memory, access speed of which is significantly higher and is equal to 2 – 4 cycles.

V. Experimental studies of the proposed algorithm

In the current survey, experimental tests were run on a computer system with an Intel Core i7-3610QM processor (2.3 GHz), 8 GB RAM and DDR3-1600 NVIDIA GeForce GT 630M video card (2GB VRAM). This graphics accelerator contains 96 CUDA kernels, and its clock frequency is 800 MHz.

It is known that the number of allocated blocks increases linearly with enhancing of processed graphs dimensionality. The complexity of calculating the relevant auxiliary matrices grows by the same principle. The stages of multi-step filtration and local hulls construction provide a significant simplification of final connection procedure. Thus, the complexity of the developed algorithm is linear $O(n)$ for uniformly distributed data.

MCH instances composed of all graph's vertices are the worst for investigation. In this case, the filtration operations do not provide the required acceleration and the algorithm complexity is equal to $O(nlogn)$. However, these examples have purely theoretical significance and almost never occur in practice.

Fig. 3 shows the dependence of the innovative algorithm execution time on the graph dimensionality and the number of vertices in the selected blocks. These results confirm the linear complexity of the proposed method. In addition, it is important to set the optimal dimensionality of the subsets allocated in the original graph. A selection of smaller blocks (up to 1000 nodes) leads to a dramatic increase in the algorithm operation time.

This phenomenon is caused by the significant enhancing of the auxiliary matrices dimensionality, making it difficult to control the computing process (Fig. 4). Per contra, the allocation of large blocks (over 5000 vertices) is associated with the elimination of the massive parallel properties, mismanagement of the video card resources, and as a consequence, increasing of the algorithm execution time. Thus, the highest velocity of the proposed method is observed for intermediate values of

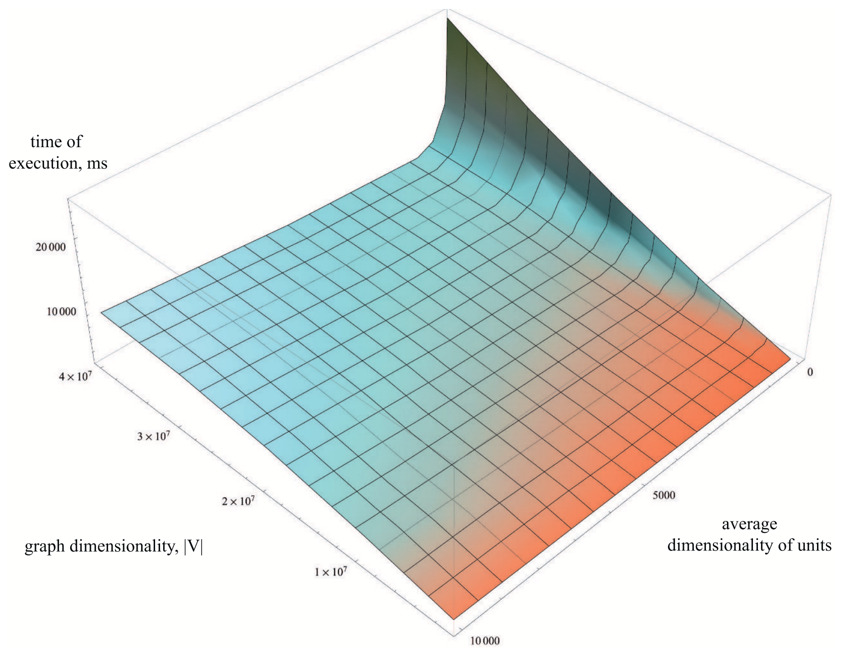


Fig. 3. Dependence of the innovative algorithm performance on the graph dimensionality and the number of vertices in the selected blocks

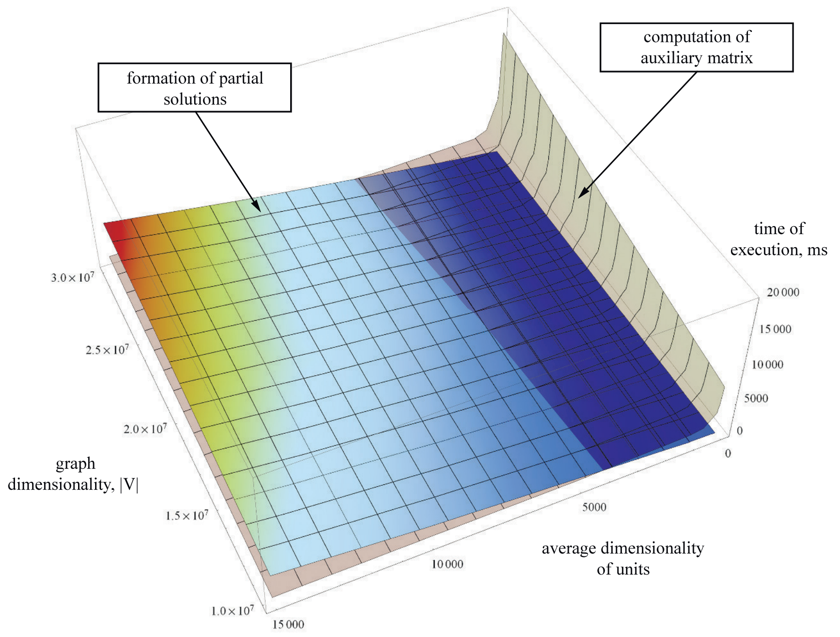
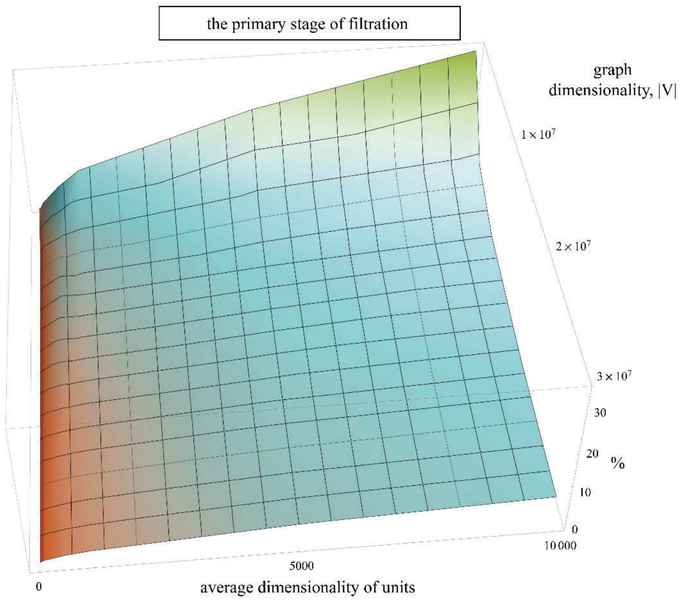


Fig. 4. Dependences of the various stages performance on the graph dimensionality and the number of vertices in the selected blocks

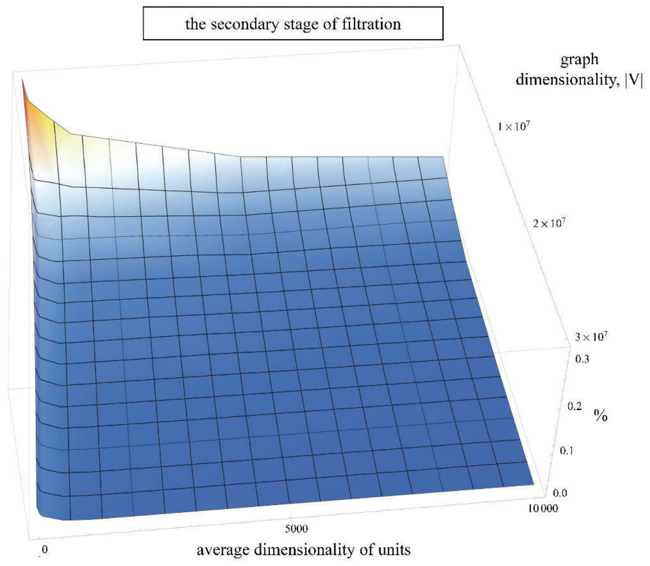
the blocks dimensionality (1000 – 5000 vertices). In this case, auxiliary matrices are relatively small, and the second stage of the algorithm preserves the properties of massive parallelism.

One of the most important means to ensure the algorithm's high performance is the multi-step filtration of the graph's vertices. Fig. 5a shows the dependence of the primary selection quality on the dimensionality of the original problem and allocated subsets. These results show that such filtration is the most efficient with the proviso that the graph's vertices are distributed into small blocks. Furthermore, the number of selected units increases with the raising of the problem's size, providing rapid solutions to graphs of extra large dimensionality. By virtue of a riddance from the discarded blocks, the following operations of the developed algorithm are applied only to 1 – 3% of the initial graph's vertices.

However, the results of the secondary filtration (Fig. 5b) are the opposite. In this case, the highest quality of the selection is obtained on the assumption that the original



(a)



(b)

Fig. 5. The influence of filtration procedure over the reduction in the problem's size

vertices are grouped into large subsets. Withal, the secondary filtration is much slower than the primary procedure, so the most effective selection occurs at intermediate values of the blocks dimensionality. As a result of these efforts, only 0.05 – 0.07% of the initial graph's vertices are involved in the final operations of the proposed algorithm.

In order to determine the efficiency of the developed algorithm, its execution time has been compared with the built-in tools of the mathematical package *Wolfram Mathematica 9.0*. All choice paired comparison tests were conducted for randomly generated graphs. The MCH formation in *Mathematica* package is realized by the instrumentality of *ConvexHull[]* function, while the *Timing[]* expression is used to measure the obtained performance. The results of the performed comparison are given in Fig. 6. They imply that the new algorithm computes the hulls up to 10 – 20 times faster than *Mathematica's* standard features.

VI. Conclusions

The paper suggests an innovative algorithm for finding the minimum convex hulls, which is based on the GPGPU technology and uses graphic accelerators. Unlike its predecessors, this algorithm is adapted to fast solving of the large-scale problems and, therefore, is suitable for using with respect to *Big Data*

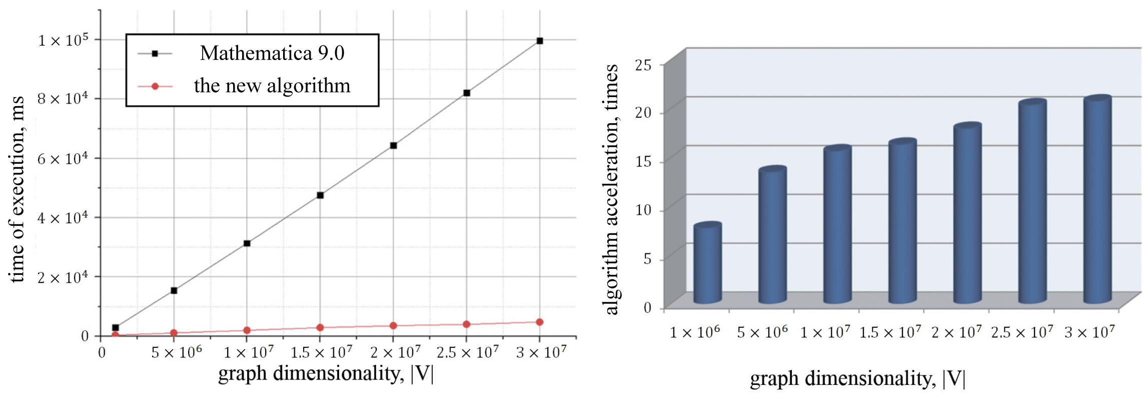


Fig. 6. A performance comparison between the new algorithm and built-in tools of the mathematical package Wolfram Mathematica 9.0

direction. Compared to the classical methods it has a number of the following benefits:

1) **High speed of operation and linear complexity.** Algorithm performance is increased by the steps of vertices' distribution into blocks, filtration and using of the auxiliary matrices. As a result, the speed of the new method is 10 – 20 times higher in contrast to the usage of professional mathematical package *Mathematica*.
2) **Massive parallelism.** Formation of partial hulls is carried out independently, which contributes to the implementation of these calculations by using graphics processors.
3) **The ability of hulls' dynamic adjustment.** When adding new vertices to the initial set, calculations are executed only for units that have undergone modification. These operations require only local updating of the convex hulls, because the results for intact parts of the original graph are invariable.
4) **The ability of generalization for the multidimensional problem instances.** In these cases, the selected subsets are the n-dimensional cubes to which operations of the developed method are applied.

These advantages may be the basis for the inclusion of the innovative algorithm in the professional mathematical packages to promote the high-performance computations among their users. A further direction of research is related to the development of hybrid CPU+GPU versions of this algorithm for complex systems with many processors and video cards.

References

[1] Aardal K., Van Hoesel S., Polyhedral Techniques in Combinatorial Optimization I: Theory. Statistica Neerlandica 50, pp. 3–26, 1995. DOI: 10.1111/j.1467-9574.1996.tb01478.x.

[2] Akl S.G., Toussaint G.T., A fast convex hull algorithm. Information processing letters 7, pp. 219–222, 1978.

[3] Allison D.C.S., Noga M.T., Some performance tests of convex hull algorithms. BIT 24(1), pp. 2–13, 1984. DOI: 10.1007/BF01934510.

[4] Barber C.B., Dobkin D.P., Huhdanpaa H., The Quickhull Algorithm for Convex Hulls, ACM Trans. Math. Softw. 22(4), pp. 469–483, 1996. DOI: 10.1145/235815.235821.

[5] Chan T.M., Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete & Computational Geometry 16, pp. 361–368, 1996. DOI: 10.1007/BF02712873.

[6] Cormen T.H., Leiserson C.E., Rivest R.L., Stein C., Introduction to Algorithms, Second Edition. Section 33.3: Finding the convex hull. MIT Press, pp. 947–957, 2001.

[7] De Berg M., Cheong O., van Kreveld M., Overmars M., Computational Geometry: Algorithms and Applications. Springer-Verlag, Heidelberg, 2008. DOI: 10.1007/978-3-540-77974-2

[8] Duncan M., Applied Geometry for Computer Graphics and CAD. Springer-Verlag, London, 2005. DOI: 10.1007/b138823

[9] Glaskowsky P.N., NVIDIA's Fermi: The First Complete GPU Computing Architecture. NVIDIA, 2009.

[10] Govindaraju N.K., Larsen S., Gray J., Manocha D., A memory model for scientific algorithms on graphics processors. Proceedings of the ACM/IEEE conference on Supercomputing, 2006. DOI: 10.1109/SC.2006.2.

[11] Graham R.L., An efficient algorithm for determining the convex hull of a finite planar set, Info. Proc. Lett. 1(1), pp. 132–133, 1972. DOI: 10.1016/0020-0190(72)90045-2.

[12] Hossain M., Amin M., On Constructing Approximate Convex Hull. American Journal of Computational Mathematics 3(1A), pp. 11–17, 2013. DOI: 10.4236/ajcm.2013.31A003.

[13] IBM: Analytics: The real-world use of big data, 2013.

[14] IBM: Storage strategies that deliver business value, 2011.

[15] Jarvis R.A., On the identification of the convex hull of a finite set of points in the plane, Info. Proc. Lett. 2(1), pp. 18–21, 1973. DOI: 10.1016/0020-0190(73)90020-3.

[16] Lee C., Ro W.W., Gaudiot J.-L., Boosting CUDA Applications with CPU-GPU Hybrid Computing. International Journal of Parallel Programming 42(2), pp. 384–404, 2014. DOI: 10.1007/s10766-013-0252-y.

[17] Nickolls J., Dally W., The GPU computing era. Micro IEEE 30(2), pp. 56–69, 2010. DOI: 10.1109/MM.2010.41.

[18] Novakovic V., Singer S., A GPU-based hyperbolic SVD algorithm. BIT 51(4), pp. 1009–1030, 2011. DOI: 10.1007/s10543-011-0333-5.

[19] Pogorilyy S.D., Potebnia A.V., Formation and investigation of Kruskal's algorithm parallel scheme for shared memory systems. Scientific Papers of Donetsk National Technical University "Informatics, Cybernetics and Computer Science" 16(204), pp. 82–89, 2012. DOI: 10.5281/zenodo.16440 (in Ukrainian).

[20] Potebnia A.V., Pogorilyy S.D., Exploration of data coding methods in wireless computer networks. Proceedings of the Fourth International Conference on Theoretical and Applied Aspects of Cybernetics (TAAC), Kyiv, pp. 17–31, 2014. DOI: 10.13140/RG.2.1.3186.3844.

[21] Preparata F.P., Shamos M.I., Computational Geometry: An Introduction. Springer-Verlag, New York, 1985. DOI: 10.1007/978-1-4612-1098-6.

[22] Sanders, J., Kandrot, E., CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 2010.

[23] Sederberg T. W., Computer aided geometric design course notes, 2011. Available: http://tom.cs.byu.edu/ 557/text/cagd.pdf.

[24] Sklodowski, P., Zorski, W., Movement Tracking in Terrain Conditions Accelerated with CUDA. Proceedings of the Federated Conference on Computer Science and Information Systems, FedCSIS 2014, Warsaw, Poland, pp. 709–717. DOI: 10.15439/2014F282.