UDC 004.032.24:519.612.2
https://doi.org/10.31891/csit-2022-3-6

OLEG ZHULKOVSKYI, INNA ZHULKOVSKA,
VOLODYMYR SHEVCHENKO, HLIB VOKHMIANIN
Dniprovsky State Technical University

EVALUATION OF THE EFFICIENCY OF THE IMPLEMENTATION OF PARALLEL COMPUTATIONAL ALGORITHMS USING THE <thread> LIBRARY IN C++

Progressive hardware and software mean of paralleling and synchronization of calculations on modern computers with multicore architecture allow to increase the efficiency of computer modeling by increasing (by an order or more) the performance of calculations. The purpose of this work is to increase the efficiency of computational algorithms for the computer implementation of the sweep method by using modern advanced parallel programming techniques. The study used methods of matrix algebra, parallel computations, as well as analysis of the efficiency of algorithms and programs. As a result of the work, computational algorithms for sequential and parallelized in two threads sweep method were developed, and a comparative evaluation of the effectiveness of their implementation by means of thread control library <thread> C++ was performed. The order of SLAE in this case was up to 5×107. As a result of computational experiments, it was possible to achieve an increase in computational speed of 1.88-2.86 times. The results obtained correspond with similar data from available literature sources. The scientific novelty of the work lies in the subsequent development of promising approaches to increase the efficiency of computer simulation through the use of modern technologies and principles of parallel programming with computational experiments on modern hardware and software architectures. For the first time, estimates of the time of software implementation of algorithms for sequential and parallelized by means of the <thread> C++ library computational algorithms for the sweep method for a significant order of SLAE were obtained. The expediency of this paralleling is demonstrated for SLAEs of the order over 2.5×105. The main significance of the work lies in the practical application of the results obtained in computer simulation of engineering problems, the most resource-intensive stage of which is the multiple solution of SLAE of a significant order. Further prospects of research assume in-depth paralleling of algorithms for numerical solution of SLAE by using scalable variations of applied methods, choosing the most productive software technologies, paralleling the program code to the maximum (in terms of the number of processor cores) number of threads.

Key words: computational algorithm, numerical solution of SLAE, sweep method, performance, parallel computation, thread, computation acceleration, multithreading.

ОЛЕГ ЖУЛЬКОВСЬКИЙ, ІННА ЖУЛЬКОВСЬКА,
ВОЛОДИМИР ШЕВЧЕНКО, ГЛІБ ВОХМЯНІН
Дніпровський державний технічний університет

ОЦІНКА ЕФЕКТИВНОСТІ РЕАЛІЗАЦІЇ ПАРАЛЕЛЬНИХ ОБЧИСЮВАЛЬНИХ АЛГОРИТМІВ ЗАСОБИМИ <thread> C++

Прогресивні апаратні та програмні засоби розпаралелювання та синхронізації обчислень на сучасних ЕОМ з багатоядерною архітектурою дозволяють підвищити ефективність комп'ютерного моделювання за рахунок збільшення (на порядок та вище) продуктивності обчислень. Метою цієї роботи є підвищення ефективності обчислювальних алгоритмів комп'ютерної реалізації методу прогонки за рахунок використання сучасних прогресивних технологій паралельного програмування. У ході дослідження використано методи матричної алгебри, паралельних обчислень, а також аналізу ефективності алгоритмів та програм. В результаті роботи було розроблено обчислювальні алгоритми послідовного та розпаралеленого на два потоки методу прогонки, а також виконано порівняльну оцінку ефективності їх реалізації за допомогою управління потоками засобами бібліотеки <thread> C++. Порядок СЛАР при цьому становив до 5×107. В результаті обчислювальних експериментів вдалося досягти збільшення швидкості обчислень у 1,88-2,86 разів. Отримані результати кореспондуються з аналогічними даними із доступних літературних джерел. Наукова новизна роботи полягає у подальшому розвитку перспективних підходів до підвищення ефективності комп'ютерного моделювання за рахунок застосування сучасних технологій та принципів паралельного програмування з проведенням обчислювальних експериментів на сучасних програмно-апаратних архітектурах. Вперше було отримано оцінки часу програмної реалізації алгоритмів послідовного та розпаралеленого засобами бібліотеки <thread> C++ обчислювальних алгоритмів методу прогонки для значного порядку СЛАР. При цьому продемонстровано доцільність застосування даного розпаралелювання для СЛАР порядку близько 2,5×105. Основна значущість роботи полягає у практичному застосуванні отриманих результатів під час проведення комп'ютерного моделювання інженерних задач, найбільш ресурсомістким етапом яких є множинне розв'язування СЛАР значного порядку. Подальші перспективи досліджень припускають поглиблене розпаралелювання алгоритмів чисельного розв'язання СЛАР за рахунок використання масштабованих варіацій застосовуваних методів, вибору найбільш продуктивних програмних технологій, розпаралелювання програмного коду на максимальну (за кількістю ядер процесора) кількість потоків.

Ключові слова: обчислювальний алгоритм, чисельне рішення СЛАУ, метод прогонки, швидкодія, паралельні обчислення, thread, прискорення обчислень, багатопоточність.

Introduction

The architecture and performance of computers are continuously improving. At the same time [1], the main emphasis of development is placed on parallel computing due to the increased number of processors (cores) in modern computers. By performing parallel computations, an opportunity is given to solve a wide range of tasks which initially require great computational resources, which was impossible for the previous decade's generation of computers [2].

The above-mentioned concepts became prerequisites for appearance of such notion as multithreaded programming. This type of programming means, first of all, acceleration and increase of efficiency of executed

algorithm due to free management of threads between which the executed program code can be distributed [3]. At present there are many methodologies in modern high-level programming languages thanks to which it is possible to perform parallel computations programmatically.

The purpose of this work is to increase the efficiency of computer modeling by increasing the performance of computing algorithms of the sweep method (TDMA - Tridiagonal matrix algorithm) by using modern parallel programming technologies by means of the C++ programming language.

In order to achieve the described goal, the following tasks are set for the work:

– to implement by software the algorithms of sequential and paralleled in two threads sweep method;

– to make a comparative characterization of the efficiency (speed) of the implementation of the above algorithms using the selected C++ tools for a significant order of SLAEs;

– to develop a concept for further development of approaches to improve the efficiency of computer modeling, using parallelized computational algorithms of the sweep method by their implementation on computers with multicore architecture.

Related work

The considered sources of information devoted to the topic of parallel computing highlight the monograph of basic principles and rules [1] which will allow designing effective parallel programs for solving various computational tasks on modern computers.

In [2], not only the existing system architectures are described in more detail, but also the tools used for their programming. Modern multiprocessor systems according to generally accepted standards are considered. In addition to this material, the educational complex [4] with a review of the architecture of modern multicore processors, which includes a comparative characteristic of modern processors, deserves attention.

Speaking about multithreaded programming within the program code itself, one of the most famous works is about the practice of multithreaded programming [3] which describes in detail the approaches to implementation of parallel computing and also considers and analyzes the problems of using common algorithms and then solving them.

Materials and Methods

Multithreading programming tools and support in C++ first appeared in C++ 11 standard in 2011. There are a number of traditional, different notions and varieties of programming which are based on the property of competitiveness [3, 5]. Often when writing a competitive program, a combination of several different methods is used, when it is possible to mix different forms of competitiveness, using the most appropriate tool separately for each block of the application.

All existing architectures are commonly classified according to the general taxonomy of computer architectures based on the presence of parallelism in instruction and data threads. This classification was proposed by Michael Flynn in 1966 and later extended in 1972 [2, 5, 6]. This taxonomy includes four classes of computing systems, of which the system MIMD (Multiple Instruction, Multiple Data) can be distinguished within the problem under consideration [1, 2, 4, 6-8]. It is worth noting that Flynn's taxonomy actually assigns all kinds of parallel systems to the MIMD class, despite their possible significant differences.

Multiprocessor systems, which are part of the MIMD classification, allow organizing a hardware implementation of parallel processing. They are represented by two types [2]: symmetric multiprocessor system SMP (Symmetric Multiprocessors) and the system with mass parallelism MPP (Massive Parallel Processing). SMP is the most common system. Processors contained in it have equal performance, as well as access to shared memory, which ultimately gives equal access time to shared memory [2, 9]. MPP systems are supercomputers with a large number of processors and shared memory, which are built as a number of separate computing nodes that interact with each other using high-speed communication channels [2, 10]. In this system, the application, which is to be executed in parallel, is divided into processors, which weakly interact with each other and exchange information by sending and receiving messages. The continuation of the ideas behind the MPP system are cluster systems. These systems also consist of multiple nodes, each of which can act as a single hardware and software system. Each node has its own local memory. Clusters are divided into two types: homogeneous and heterogeneous. If all nodes of a cluster have the same performance and architecture, a cluster is considered homogeneous [2].

All of the above computing systems can be used to solve a wide range of different tasks. Multiprocessor systems, including supercomputers, cluster systems are used to solve problems that require processing of significant amounts of data [2, 10]. Multiprocessor SMP systems are used for real-time data processing [2, 9]. The most common example is a web server.

The most widespread and one of the first programming tools for parallel processing computer systems with shared memory is MPI (Message Passing Interface) [1, 2, 11]. The first such standard was approved in 1993. Later, in 1997, the second version was approved, expanding the capabilities of the first one. The MPI interface provides compatibility between many UNIX-based operating systems (OS), and is also supported in programming languages such as C, C++, and Fortran. The MPI library provides a list of various functions that can be used to organize

interaction between processors: message exchange between processors, regulation of message transmission mode, indication of data types transferred by messages etc. [2, 11].

Another set of standards that describes the interfaces between the OS and the application program, which is a system API (Application Programming Interface) is POSIX (Portable Operating System Interface), described in the standard ISO/IEC 9945. It additionally describes the C programming language library. Like MPI, this standard also provides compatibility between multiple UNIX-based operating systems [3, 12-14]. Any UNIX process has a component of a certain number of threads of control, which contain common address space, but different command threads. It is assumed that the simplest process consists of only one thread [6]. POSIX interface can be considered as an organization of Pthreads (POSIX threads), which is supported by most UNIX systems. Unfortunately, its practical application for organizing parallel computations is reduced to a minimum due to the lack of data parallelism, and the fact that initially the mechanism of using threads was not developed for the further organization of parallel computations [15].

A more modern and widely used methodology for organizing parallel computing is the OpenMP shared-memory multiprocessor system [15]. This technology allows to introduce parallel processing in those algorithms which are executed sequentially. It does not require any changes in the source code of the program. It is enough just to add necessary parallel processing directives to those code fragments which are to be executed in parallel. After that the code will be executed simultaneously on several processors [15, 16]. The OpenMP methodology can be used together with MPI interface to program computing clusters that contain multiprocessor nodes. In this case, OpenMP implements the tasks on individual nodes, and MPI provides interaction between the nodes [1, 11, 15].

The C++ programming language has a separate library for working with threads <thread>, which is also investigated in this paper. The library contains the std::thread class for full-fledged work with threads. The mentioned class includes a lot of various features thanks to which any task can be solved in a necessary and optimal way. At the moment when a program performs many tasks, threads allow to increase the performance to a great extent. The creation and destruction of a thread is not time-consuming. Thanks to threads, many algorithms can easily be represented as independent subtasks that can be executed in separate threads, thereby improving performance. In the framework of this problem, the use of threads is the most appropriate approach, because their use allows to implement the left sweep method in one thread, and the right sweep method in another [3, 17].

The main features and methods for working with threads can be defined as follows [3, 17, 18].

1. Identification. Each thread has its own unique identifier, which can be obtained by a special method get_id(). It is used to check in which of the threads the code is executed and whether it is executed in parallel by comparing the displayed identifiers with each other.

2. Waiting. Each thread has two main methods of waiting: join() - it waits until the created thread is finished and blocks the calling thread, detach() - it does not wait until the created thread is finished and does not block the calling thread, which allows the main function main() to continue.

3. Suspension. Each thread can be suspended for a user-defined time, if desired. To provide this functionality, use the sleep_for() methods - to suspend thread execution for a certain time, and sleep_until() - to suspend thread execution until a certain time passed with the argument.

The aforementioned basic methods and features provide flexible thread control, which in fact allows you to parallelize the task in a convenient way, obtaining on the output a minimum time of program code execution. But despite all the advantages of the methodology, the problem of «data race» is still urgent. Parallel access to a thread object, a thread buffer object or a library thread can lead to it. That is, two threads that run in the same address space can independently access an object in ways that produce undefined results. For example, if one writes an object and the other reads data from that object, there is a «race», i.e., setting the order of execution of the corresponding operation. The results obtained may not only be undefined, they are often completely unpredictable [3, 17].

There are several ways to solve the described problem [3, 17]:

– encapsulating the data structure in a protection mechanism can ensure that intermediate states with a violation of invariance are visible only to the thread that is currently making any changes;

– using «lock-free programming» – a concept that involves changing the design of the data structure and invariants in such a way that changes are made as a series of indivisible modifications, each of which preserves the invariants;

– using the method of «Software Transactional Memory» (STM - Software Transactional Memory) – updating the data structure in the transaction form itself in the same way as for database updates (the sequence of changes is stored in the transaction log, and then fixed within a single operation [3, 17]).

The <thread> library, along with many advantages has disadvantages, the main of which is the problem of «data race». The solution to this problem is possible, but in practice, depending on the task at hand, this problem may develop into an extreme degree of complexity.

Experiments

To carry out experiments of this kind it is necessary to have a computer which contains processors equal in performance, as well as having the same access time and equal rights when accessing shared memory. Modern multicore processors have independently functioning computing modules (cores).

The present research was conducted using a purely multicore architecture (Table 1).

Table 1

Architecture of the computational experiment

CPU	Intel Core i5-8400 (6 cores, 2.8 GHz), cache 9 MB
RAM	Goodram DDR4 (4 GB, 2666 MHz, 21300 MB/s)×4
Operating system	Microsoft Windows 10
Integrated development environment	Microsoft Visual Studio C++
Programming technology	<thread>

As part of the research, a software functional was developed that implements the classical algorithm of the sequential right sweep method and the counter sweep method paralleled in two threads. The counter sweep algorithm is a combination of left and right sweeps. The size of the SLAE during the study was varied in the range from 1×10^5 to 5×10^7. The values of the equation coefficients were randomly generated into variables of the standard type double, observing the conditions of diagonal predominance of the matrix.

To measure the execution time of the algorithm the class steady_clock from the <chrono> C++ library was used, which has the necessary set of classes to represent moments in time. The specified class provides access to a stable clock, and is also best suited for measuring intervals. One of the key methods of the class steady_clock are the new and duration methods, which allow to get the current time and keep the difference between two instants of time, respectively [3].

Results

The obtained results of calculations (Table 2), depending on the order of SLAE, show the time of software implementation of sequential algorithms of right and counter sweep, as well as parallelized in two threads version of counter sweep using <thread> C++ technology. The key results demonstrating the relevance and novelty of the work are shown in Fig. 1-3.

Table 2

Results of the computational experiment

SLAE order	Sequential algorithms			Parallel algorithm		
	right sweep ($t1$, s)	counter sweep ($t2$, s)	$s1=t1/t2$	counter sweep ($t3$, s)	$s2=t1/t3$	$s3=t2/t3$
100000	0,001969	0,001070	1,84019	0,010454	0,18835	0,10235
200000	0,003778	0,002325	1,62495	0,011972	0,31557	0,19420
300000	0,005328	0,003411	1,56201	0,012134	0,43910	0,28111
400000	0,007292	0,004712	1,54754	0,012992	0,56127	0,36268
500000	0,009424	0,005836	1,61480	0,013747	0,68553	0,42453
600000	0,010809	0,007042	1,53493	0,014496	0,74565	0,48579
700000	0,012564	0,008276	1,51812	0,015192	0,82701	0,54476
800000	0,014579	0,009442	1,54406	0,015766	0,92471	0,59888
900000	0,016272	0,011038	1,47418	0,016102	1,01056	0,68550
1000000	0,018275	0,011839	1,54363	0,016677	1,09582	0,70990
2500000	0,047037	0,030118	1,56176	0,025979	1,81058	1,15932
5000000	0,092932	0,058827	1,57975	0,04137	2,24636	1,42197
10000000	0,184777	0,119720	1,54341	0,071688	2,57752	1,67001
15000000	0,301776	0,198740	1,51845	0,105439	2,86209	1,88488
20000000	0,370866	0,238921	1,55225	0,136108	2,72479	1,75538
35000000	0,654413	0,418034	1,56545	0,234742	2,78780	1,78082
50000000	0,942063	0,610584	1,54289	0,342137	2,75347	1,78462

According to the experimental results, application of the counter sweep method on single-core architectures even without using parallelism means allows to increase performance of the computational algorithm (Fig. 1, 2). Thus, computational speed-up (s1) due to this replacement in the considered range of SLAE order change was 1.47-1.84. The maximum computation time in conditions of the used computational experiment architecture in the considered range of SLAE order variation did not exceed one second for any of the investigated implementations of the sweep method (Fig. 1).

The best approximation of the obtained data is provided by their linear approximation of the represented regression functions (Fig. 1).

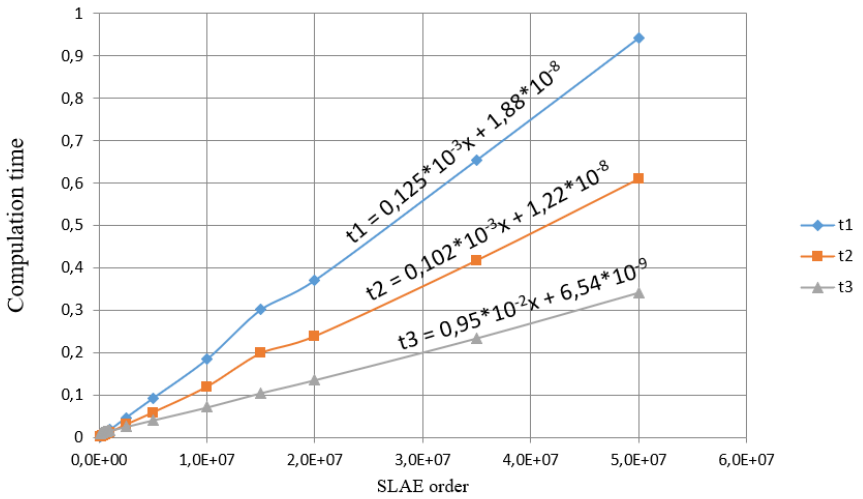

Fig. 1. Dependence of the solution time on the order of SLAE (x) in the range 1×10^5-5×10^7 for the implementation of the sweep method: t1 - right (sequential calculation); t2 - counter (sequential calculation); t3 - counter (parallel calculation)

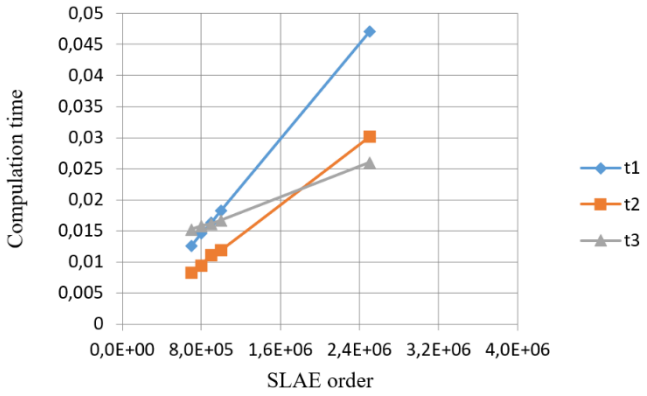

Fig. 2. Dependence of the solution time on the order of SLAE in the range 1×10^5-4×10^5 for the implementation of the sweep method: t1 - right (sequential calculation); t2 - counter (sequential calculation); t3 - counter (parallel calculation)

The conclusion about the performance of the counter-parallelized algorithm for two threads and the use of specialized tools of software parallelism in comparison with the sequential analogue in the investigated range of matrix order variation is ambiguous. Parallel processing time in the counter-parallelization method for the SLAE order less than 2.5×10^5 will be longer than in the traditional method of sequential calculations, i.e., t3> t2 (Fig. 2) and s3<1 (Fig. 3).

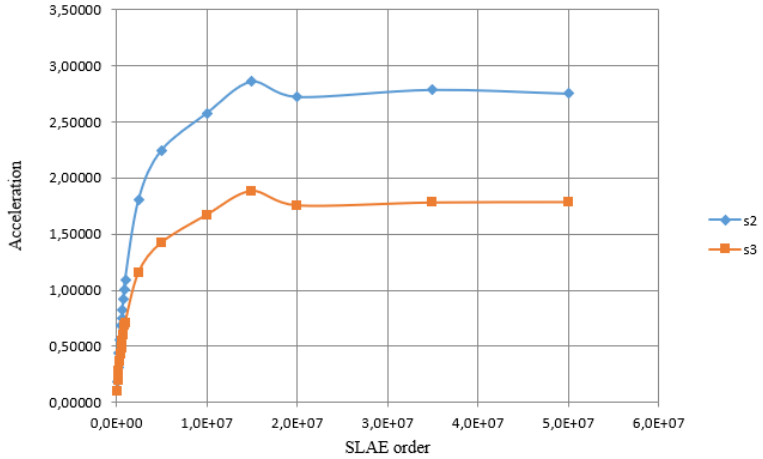

Fig. 3. Dependence of the speedup of parallel computation on the order of SLAE with respect to the sequential implementation of the right (s2) and counter (s3) sweep method.

The computation time slows down and acquires predictable values when the order of SLAE exceeds 2.5×10^5. The process begins to accelerate (s3 exceeds one and reaches the value 1.88).

Slowdown of calculations for SLAEs of the order less than 2.5×10^5 is caused by spending machine time for creation of computational threads, and the values are comparable or exceed the time of calculations themselves at a relative slowdown of the software implementation of the parallelized algorithm. Thus, the remark about inexpediency of using multicore architectures and parallel computing technologies up to a certain order of SLAE becomes relevant.

The results obtained (Fig. 3, graph s3) [19] correspond to the data obtained by the authors on computers with similar infrastructure in the course of research using a multiprocessor system with shared memory and OpenMP technology [16], which is a widely used methodology for organizing parallel computations. The results also correlate with the data of obtained on computers with similar infrastructure using the Intel MKL mathematical library.

It turned out that the computation time increases in direct proportion to the increase in the matrix size, regardless of the approach to the implementation of the sweep method algorithms under study. However, due to the use of parallel algorithms for a significant order of SLAEs (more than 2.5×10^5), the computation acceleration always exceeds unity (Fig. 3).

Obviously, the practical application of the mentioned algorithms in the processing of large amounts of data is appropriate.

Conclusions

As a result, the expediency of using progressive hardware-software tools based on modern multicore architectures to increase the efficiency of computational experiment has been proved. The use of the above programming technologies in the development of widespread methods of solving SLAEs provided an increase in the computational speed by 1.88-2.86 times.

For the first time estimates of time for software implementation of algorithms of sequential and parallelized by means of <thread> library of C++ computational algorithms of the sweep method for a significant order of SLAEs have been obtained. At the same time, the expediency of application of this paralleling for SLAE of the order more than 2.5×10^5 is demonstrated. The main significance of the work lies in the practical application of the obtained results for computer simulation of engineering problems, the most resource-intensive stage of which is the multiple solution of SLAE of significant order.

Further prospects of research assume in-depth paralleling of algorithms of numerical solution of SLAE by using scalable variations of applied methods, choosing the most productive software technologies, paralleling the program code to the maximum (by the number of processor cores) number of threads in order to increase the efficiency of computer modeling even more.

References

1. Trobec R., Slivnik B., Bulic P., Robic B. Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art Platforms: textbook. 2018. 268 p.
2. Hord R. Parallel Supercomputing in MIMD Architectures: textbook. 2018. 623 p.
3. Williams A. C++ Concurrency in Action, Second Edition: textbook. 2019. 592 p.
4. Keckler S., Olukotun K., Hofstee H. Multicore Processors and Systems: textbook. 2009. 319 p.
5. Cleary S. Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming: textbook. 2019. 254 p.
6. Robey R., Zamora Y. Parallel and High Performance Computing: textbook. 2021. 704 p.
7. Flynn M. J. Very High-speed Computing Systems. 1966. Pp. 1901-1909.
8. Flynn M. J. Some computer organizations and their effectiveness. 1972. Pp. 948-960.
9. Schimmel C. UNIX Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers: textbook. 1994. 432 p.
10. Levesque J., Vose A. Programming for Hybrid Multi/Manycore MPP Systems: textbook. 2017. 342 p.
11. Gropp W., Lusk E., Skjellum A. Using MPI, third edition: Portable Parallel Programming with the Message-Passing Interface: textbook. 2014. 336 p.
12. Kerrisk M. The Linux Programming Interface: textbook. 2010. 1552 p.
13. The Open Group UNIX. URL: https://unix.org
14. Technical Specification for C++ Extensions for Transactional Memory. URL: https://www.iso.org/standard/66343.html
15. Mattson T., He H., Koniges A. OpenMP Common Core: Making OpenMP Simple Again: textbook. 2019. 320 p.
16. Zhulkovskyi O. O., Zhulkovska I. I., Shevchenko V. V. Evaluating the effectiveness of the implementation of computational algorithms using the OpenMP standard for parallelizing programs. Informatics and Mathematical Methods in Simulation. 2021. Vol. 11. Pp. 268-277.
17. Posch M. Mastering C++ Multithreading: Write robust, concurrent, and parallel applications: textbook. 2017. 246 p.
18. Microsoft Documentation: thread-class. URL: https://docs.microsoft.com/ru-ru/cpp/standard-library/thread-class
19. Zhulkovskyi O. O., Zhulkovska I. I., Shevchenko V. V., Vokhmianin H. Ya. Vykorystannia zasobiv <thread> C++ dlia pidvyshchennia efektyvnosti kompiuternoho modeliuvannia: materialy Vseukr. naukovo-metod. konf. «Problemy matematychnoho modeliuvannia», m. Kamianske, 25-27 trav. 2022 r. Kamianske, 2022. Pp. 60-61.

Oleg Zhulkovskyi **Олег Жульковський**	Ph.D., Associate Professor of the Department of Software Systems, Dniprovsky State Technical University, Kamenskoe, Ukraine. e-mail: olalzh@ukr.net https://orcid.org/0000-0003-0910-1150	кандидат технічних наук, доцент кафедри програмного забезпечення систем, Дніпровський державний технічний університет, Україна.
Inna Zhulkovska **Інна Жульковська**	Ph.D., Associate Professor of the Department of Software Systems, Dniprovsky State Technical University, Kamenskoe, Ukraine. e-mail: inivzh@gmail.com https://orcid.org/0000-0002-6462-4299	кандидат технічних наук, доцент кафедри програмного забезпечення систем, Дніпровський державний технічний університет, Україна.
Volodymyr Shevchenko **Володимир Шевченко**	Student of the Department of Software Systems, Dniprovsky State Technical University, Kamenskoe, Ukraine. e-mail: volodshzk@gmail.com https://orcid.org/0000-0002-9542-7060	здобувач вищої освіти другого (магістерського) рівня кафедри програмного забезпечення систем, Дніпровський державний технічний університет, Україна.
Hlib Vokhmianin **Гліб Вохмянін**	Student of the Department of Software Systems, Dniprovsky State Technical University, Kamenskoe, Ukraine. e-mail: vohmyanin.yleb@gmail.com https://orcid.org/0000-0002-9582-5990	здобувач вищої освіти першого (бакалаврського) рівня кафедри програмного забезпечення систем, Дніпровський державний технічний університет, Україна.