https://doi.org/10.31891/csit-2024-4-15
UDC 004.62

Oleg ZHULKOVSKYI
Dniprovsky State Technical University
Inna ZHULKOVSKA
University of Customs and Finance
Hlib VOKHMIANIN
Dniprovsky State Technical University
Alexander FIRSOV, Illia TYKHONENKO
University of Customs and Finance

APPLICATION OF SIMD-INSTRUCTIONS TO INCREASE THE EFFICIENCY OF NUMERICAL METHODS FOR SOLVING SLAE

Computational efficiency has become a key factor in progress across many fields of science and technology. However, traditional methods for improving the performance of computational systems have reached their limits, necessitating the search for new approaches to algorithm optimization. This paper explores the application of SIMD instructions to enhance the efficiency of numerical methods for solving systems of linear algebraic equations, particularly the Gauss method and the conjugate gradient method. The proposed approach enables the vectorization of computations, significantly reducing the number of iterative steps and accelerating algorithm execution. An optimization mechanism is presented, based on an analysis of the capabilities of SIMD instructions and their integration into existing SLAE-solving algorithms. The research includes an examination of the impact of vectorization on the performance and stability of numerical algorithms for problems of varying size, as well as a theoretical justification of the proposed approach's effectiveness. The outcome of this work is the development of optimized versions of the Gauss and conjugate gradient methods, which demonstrate significant performance gains without loss of calculation accuracy. The proposed approach opens new perspectives for further development and improvement of numerical methods within the context of modern computing architectures, with broad applicability in engineering calculations, computer graphics, machine learning, and other fields where computational efficiency is of high priority. The presented approach opens new prospects for further development and improvement of numerical methods in the context of modern computational architectures, which may have wide applications in engineering calculations, computer graphics, machine learning, and other fields where computational efficiency is of high priority.

Keywords: SLAE, Gauss method, conjugate gradient method, SIMD instructions, computational optimization, data-level parallelism.

Олег ЖУЛЬКОВСЬКИЙ
Дніпровський державний технічний університет
Інна ЖУЛЬКОВСЬКА
Університет митної справи та фінансів
Гліб ВОХМЯНІН
Дніпровський державний технічний університет
Олександр ФІРСОВ, Ілля ТИХОНЕНКО
Університет митної справи та фінансів

ЗАСТОСУВАННЯ SIMD-ІНСТРУКЦІЙ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ЧИСЕЛЬНИХ МЕТОДІВ РОЗВ'ЯЗАННЯ СЛАР

Обчислювальна ефективність стає ключовим фактором прогресу в багатьох галузях науки і техніки. Однак традиційні методи підвищення продуктивності обчислювальних систем досягли своїх меж, що зумовлює необхідність пошуку нових підходів до оптимізації алгоритмів. У роботі розглядається застосування SIMD-інструкцій для підвищення ефективності чисельних методів розв'язання систем лінійних алгебраїчних рівнянь, зокрема методу Гауса та методу спряжених градієнтів. Запропонований підхід дозволяє векторизувати обчислення, що суттєво зменшує кількість ітераційних кроків та прискорює виконання алгоритмів. Представлено механізм оптимізації, який базується на аналізі можливостей SIMD-інструкцій та їх інтеграції в існуючі алгоритми розв'язання СЛАР. Дослідження включає вивчення впливу векторизації на швидкодію та стабільність чисельних алгоритмів при різній розмірності задач, а також теоретичне обґрунтування ефективності запропонованого підходу. Результатом роботи є розробка оптимізованих версій методів Гауса та спряжених градієнтів, які демонструють значне підвищення продуктивності без втрати точності обчислень. Представлений підхід відкриває нові перспективи для подальшого розвитку та вдосконалення чисельних методів у контексті сучасних обчислювальних архітектур, що може мати широке застосування в інженерних розрахунках, комп'ютерній графіці, машинному навчанні та інших галузях, де ефективність обчислень має високу пріоритетність. Представлений підхід відкриває нові перспективи для подальшого розвитку та вдосконалення чисельних методів у контексті сучасних обчислювальних архітектур, які можуть мати широке застосування в інженерних розрахунках, комп'ютерній графіці, машинному навчанні та інших галузях, де ефективність обчислень має високий пріоритет.

Ключові слова: СЛАР, метод Гауса, метод спряжених градієнтів, SIMD-інструкції, оптимізація обчислень, паралелізм на рівні даних.

Introduction

In the era of digital transformation and rapid development of information technologies, computational efficiency has become a key factor in progress across many fields of science and technology. Traditionally, the increase in computational system performance has been achieved by increasing processor clock speeds. However, this approach has now reached its limit, shifting the focus to the optimization of computational algorithms and more efficient use of available hardware resources [1].

Numerical methods for solving systems of linear algebraic equations (SLAE), such as the Gauss method and the conjugate gradient method, are widely used in engineering and scientific calculations, computer graphics, modeling, and more. The Gauss method is effective for solving small dense systems of equations, while the conjugate gradient method is used for large sparse matrices, which arise, for instance, in problems of mathematical physics, hydrodynamics, aerodynamics, and seismology. Both methods are key in optimization problems, fluid and gas flow modeling, as well as in machine learning. In this context, improving the efficiency of numerical methods for solving SLAE and further optimizing them for implementation on modern computational architectures is a relevant task.

SIMD instructions (Single Instruction, Multiple Data) allow performing a single operation simultaneously on multiple data elements, offering a modern and promising approach to data-level parallelism and contributing to increased computational performance [1, 2]. SIMD instructions enable the vectorization of computations, reducing the number of iterative steps and accelerating algorithm execution. This approach is effective in linear algebra tasks, including solving large SLAE.

This study aims to explore the potential of SIMD instructions for optimizing the Gauss and conjugate gradient methods. The goal of the work is to enhance the performance of these SLAE-solving methods without sacrificing result accuracy.

The scientific novelty of the work lies in identifying new aspects of optimizing popular numerical methods for solving SLAE through the use of SIMD instructions, examining the impact of computation vectorization on the performance and stability of numerical algorithms for problems of varying size, and obtaining new theoretical results on the potential of SIMD instructions to improve the efficiency of iterative methods for solving large SLAE.

Related works

A priority research direction when using SIMD technology is the optimization of operations with large numbers. Using Intel Advanced Vector Extensions 512 (AVX-512), it is appropriate to implement a method for dividing large integers. In [1], a modification of the standard division algorithm, better adapted for SIMD, is proposed. The AVX-512IFMA (Integer Fused Multiply-Add AVX-512) instruction set works effectively with large integer multiplication to compute division through reverse-multiplication-based functionality. This approach resulted in a 25-35% average performance increase compared to the GNU Multiple Precision Arithmetic Library (GMP) when working with numbers of varying sizes. The division process uses a basic algorithm and a «divide-and-conquer» approach, combining methods for efficient computation with SIMD instructions to perform division on values of various sizes.

Matrix computation optimization can be achieved through recursion and combining AVX instructions with OpenMP technology [2]. A software method for data prefetching is used for block matrix multiplication in shared memory. Together, this combination reduces memory latency by preloading memory pages before they are accessed. The results show that for matrices of size 8192×8192, the execution time decreased by about 22%, and performance improved by 17% when prefetching was applied on an Intel Core i7 processor.

512-bit SIMD instructions are frequently used in data processing and compression. For instance, [3] presents an enhancement to Huffman coding efficiency using SIMD technology, which consists of four sequential stages: Huffman table creation, data initialization, lookup table, shifting, and merging of data. The code table is set according to SIMD instruction characteristics and is divided into eight parts, with two in each group. The code table does not store the length of the codewords, as it uses flag bits in each storage element to distinguish codewords from non-codewords. The shifting and merging algorithm is necessary for handling data and removing gaps between them, as the size of each entry differs after SIMD-accelerated table lookups. Experiments conducted on three different datasets (Calgary, Silesia, Canterbury) showed a throughput increase of 12.01% on average compared to the existing Huff0 library. In the context of data encoding and decoding, [4] proposes an improved decoding algorithm for finite-length BATS codes using belief propagation (BP)-based decoding. This iterative approach combines BP decoding and incremental Gaussian elimination (IGE) to enhance decoding efficiency. Instead of directly applying Gaussian elimination after BP decoding, IGE is iteratively used to recover BP decoding, significantly reducing the computational complexity of the process. Simulation results demonstrate a substantial reduction in the number of operations in the finite field during decoding, while the proposed algorithm achieved similar efficiency to traditional Gaussian elimination (GE).

The addition to the C++ programming language standard and ongoing standardization efforts are aimed, among other things, at incorporating new data-parallel types into the C++ standard library [5]. This trend enables the use of vectorization methods in existing C++ code without relying on the compiler's capabilities for automatic code vectorization. The integration of existing parallel algorithms with new data-parallel types opens up a new way to

accelerate existing code with minimal effort. Performance analysis of two new data-parallel execution policies, simd and par_simd, proposed by the latest versions of GCC and Clang C++ standard libraries, demonstrates significant results, especially when combined with the HPX parallel algorithms module. Compared to sequential execution, a speedup of over three orders of magnitude was recorded [5] when using the par_simd data-parallel execution policy with HPX parallel algorithms. The implementation is performance-portable across various computational architectures (Intel, AMD, Arm), utilizing different vectorization extensions (AVX2, AVX512, NEON128).

In the context of specific computational tasks, such as one-dimensional Fast Fourier Transforms (1D FFT), SIMD optimization also shows significant potential. A new approach has been presented [6] for implementing 1D FFT on computational units composed of multiple cores with SIMD extensions and shared multi-banked local memory. The proposed method combines SIMD lane-slicing and sample partitioning techniques to create multicore FFT implementations that do not require matrix transposition and include only a single bit-reversal unscrambling stage. This approach was successfully demonstrated on the Kalray MPPA3 processor, where it outperformed the classic six-step multicore FFT implementation.

Complex control tasks, such as Model Predictive Control (MPC), can be optimized using SIMD and Graphics Processing Units (GPU) [7]. The MPC problem is reformulated by eliminating the state variables and applying the interior-point method in the condensed space to remove inequalities from the KKT system. The resulting condensed matrix is positive definite and can be efficiently factorized in parallel on GPU and SIMD architectures. The size of the condensed matrix depends only on the number of control elements in the problem. The method becomes more efficient when the problem has many states, few inputs, and a moderate horizon length. Numerical results for problems with constraints in the form of partial differential equations show that the described approach is an order of magnitude faster than the standard CPU implementation.

Modified methods of classical Gaussian elimination are often used in modern research and computational algorithms. Structured Gaussian Elimination (SGE) is an important class of methods for efficiently solving sparse linear systems [8]. A new approach addresses the triangulation process, which is a key step in SGE. Instead of using the traditional exhaustive search to find the maximum component (MC), the proposed method utilizes the Disjoint Set Data Structure (DSDS), allowing for efficient management of system components. The additional storage and time costs associated with DSDS grow linearly with the number of unknowns and constraints in the linear system. Simulation results show that using DSDS is several times faster than exhaustive search when determining the components.

In the context of signal recovery measured at rates below the Nyquist frequency, an efficient architecture for the Orthogonal Matching Pursuit (OMP) algorithm has been proposed [9]. The implementation is validated on a Field-Programmable Gate Array (FPGA) for performance testing. Instead of calculating the pseudo-inverse matrix based on matrix factorization, Gaussian elimination is used to estimate the signal. A new algorithm for incremental Gaussian elimination is employed in OMP. The design is implemented on a Virtex6 FPGA and compared with other published works under the same conditions. A recovery signal-to-noise ratio (RSNR) of 23.98 dB was achieved. Validation was performed using compressed measurements from an analog information converter (AIC) on an Artix7 FPGA. The proposed architecture is hardware-efficient, faster, and consumes less dynamic power compared to other existing designs, while maintaining effectiveness even with increased values.

The conjugate gradient method is also commonly used in modern research. In the fields of neural network optimization and adaptive filtering, methods based on stochastic conjugate gradients are actively being developed. Specifically, an adaptive strategy for stochastic conjugate gradient (ASCG) optimization of backpropagation neural networks has been proposed [10]. ASCG combines the benefits of stochastic optimization and conjugate gradient methods to enhance training efficiency and convergence speed. The algorithm adaptively calculates the learning rate and search direction at each iteration. Experimental results on benchmark datasets show that ASCG optimization outperforms standard optimization methods in terms of convergence time and model performance. For instance, the ASCG algorithm achieves 21% higher accuracy on the HMT dataset and demonstrates comparable results on other datasets (DIR-Lab).

Adaptive filtering employs a new online conjugate gradient algorithm with random Fourier features (RFFCG), based on the minimum mean square error (MMSE) criterion [11]. RFFCG approximates the kernel using random Fourier features, reducing computational complexity and memory requirements in kernel adaptive filters (KAF) without the need for sparsification. At the same time, RFFCG effectively uses only one error in the loss function, approaching the filtering accuracy of KAF with sparsification based on all errors in the loss function. Monte Carlo simulations for short-term chaotic time series forecasting confirm the advantages of the proposed RFFCG algorithm.

Reliability-Based Design Optimization (RBDO) employs a modified conjugate gradient approach (MCGA) for RBDO with a nonlinear performance function [12]. The MCGA enhances solution efficiency by modifying the relevant parameters of the conjugate gradient approach (CGA) and the direction of the conjugate gradient algorithm to find the optimal design point. Results from three numerical examples featuring highly nonlinear performance functions and an optimization example for a speed reducer design demonstrate that the MCGA method exhibits better efficiency and reliability in structural reliability analyses and RBDO compared to other methods.

Classical Gaussian and conjugate gradient methods for solving systems of linear algebraic equations are frequently used in contemporary algorithmic tasks and research. At the same time, the effectiveness of SIMD optimization depends on the specifics of the problem and the architecture of the processor, but in many cases, it allows for significant performance improvements through the implementation of data-level parallelism.

Based on the analysis of the current state of the researched problem, the potential for accelerating computations often remains underutilized, especially in classical algorithms for mathematical modeling. Given the growing demand for efficient computations in complex simulations in physics, engineering, and other scientific fields, the following tasks have been set for this work:

– to analyze the performance of classical methods for solving systems of linear algebraic equations, implemented using modern programming tools, to establish a baseline level of efficiency;

– to develop modified implementations of classical SLAE-solving algorithms using SIMD instructions aimed at maximizing the capabilities of modern processors;

– to conduct a comparative performance analysis of classical and modified approaches for different SLAE dimensions;

– to determine the efficiency limits and assess the feasibility of applying SIMD optimization for different classes of SLAE-related problems in the context of modern computational challenges.

Materials and Methods

SIMD instructions, which are included in many modern processors, provide the capability for parallel data processing. Unlike traditional instructions, SIMD allows for operations to be performed simultaneously on multiple data elements by utilizing specialized vector registers. These registers can store several values and execute vector operations with specialized commands [13]. Furthermore, this approach ensures more efficient use of the processor's computational resources, especially when its computational power is limited, and also contributes to reducing the overall energy consumption of the system [14].

Among the key features of SIMD technology in the context of solving SLAE, it is important to highlight vector addition, subtraction, and multiplication, blending operations, shuffle operations, and vector comparison operations. Blending operations combine the results of vector operations, while shuffle operations allow data reorganization within vector registers [15].

Modern processors that support AVX-256 instructions offer 16 vector registers, each 256 bits wide, which enables parallel computation on 4 double-precision floating-point numbers (double) or 8 single-precision numbers (float) (Fig. 1, a). Processors with AVX-512 support, in turn, provide twice the processing capacity for such numbers (Fig. 1, b) [16, 17].

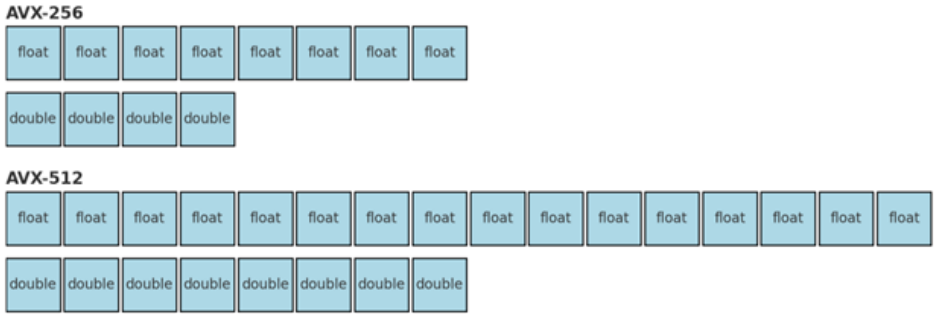

Fig. 1. Distribution of data types in AVX instructions

The built-in library <immintrin.h> used in this work within the modern IDE Visual Studio C++ provides appropriate interfaces for utilizing SIMD instructions, particularly AVX. It enables the use of low-level optimizations for executing vector operations, which significantly accelerates computations.

The main components of the library are [13, 14]:

- a set of new data types for different bit lengths [18]:
  – 128- bit vectors for single-precision floating-point operations (__m128), double precision floating-point operations (__m128d) and integer operations (__m128i);
  – 256- bit vectors: __m256, __m256d, __m256i respectively;
  – 512- bit vectors: __m512, __m512d, __m512i respectively;
- A set of SSE (Streaming SIMD Extensions) instructions for working with 128-bit vectors: SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2;
- AVX instructions to extend the capabilities of SSE using 256-bit and 512-bit registers: AVX, AVX2, AVX-256, AVX-512;
- FMA (Fused Multiply-Add) instructions to perform multiplication and addition operations in a single clock cycle.

The key functions include data loading and storage, arithmetic and logical operations, comparison operations,

and shuffling. The CPUID instruction, prior to using AVX instructions, ensures that the target processor supports them. To achieve maximum performance, data must be aligned according to the vector size (e.g., 32 bytes for AVX) [19]. Additionally, residual calculations should be considered for processing elements that are not multiples of the vector size.

The performance of computations depends on the specific architecture, and the use of SIMD may be less efficient for small matrices due to overhead costs associated with vectorization. Modern compilers can automatically vectorize some loops, but manual optimization using intrinsics is necessary to achieve maximum performance [18]. SIMD optimizations are not applicable to algorithms with a sequential nature of computations. For instance, SIMD is inappropriate when processing sequential data, where the next array element is calculated only after modifying the previous one.

When optimizing the classical Gaussian elimination method for solving SLAE using SIMD technology, the _mm256_set1_pd instruction creates a vector of four double values, all equal to a single scalar (the factor). The _mm256_loadu_pd instruction loads four consecutive double values from memory into a 256-bit vector. The instructions _mm256_mul_pd and _mm256_sub_pd perform multiplication and subtraction between two vectors of four double values, respectively. The _mm256_storeu_pd instruction stores four double values from the vector back into memory [18]. Thus, the optimization principle is based on vectorizing operations, which allows performing a single instruction on multiple data elements simultaneously. Addition, subtraction, and multiplication operations are performed in parallel for several matrix elements within each iteration. During the forward elimination stage, variable processing is done in blocks that match the width of the SIMD registers. The optimization of the back substitution phase is limited by the sequential nature of calculations, where the next value depends on the previous one.

In a similar implementation of the modified approach using the conjugate gradient method, besides the instructions _mm256_loadu_pd and _mm256_storeu_pd, the _mm256_setzero_pd instruction is used to create a vector of four double values initialized to zero, and _mm256_fmadd_pd is used to perform fused multiply-add operations on vectors of four double values [18]. The iterative nature of the method also limits the potential for full parallelism at a high level, but within each step, SIMD is effectively used to execute the main operations of addition, multiplication, and scalar multiplication.

Experiments

For the purpose of conducting experiments, implementations of the classical Gauss and Conjugate Gradient methods were developed for solving SLAE, followed by modifications using the <immintrin.h> library in Visual Studio C++, which provides the necessary SIMD instructions.

The experiments were conducted using the following test environment: Intel Core i7-12700H processor (14 cores, 2.3 GHz / 4.7 GHz), Goodram DDR4 working memory (16 GB) × 2 = 32 GB, and the Microsoft Windows 10 operating system.

The experiments involve gradually increasing the matrix size as an input parameter. The coefficient matrix and the free term vector for the SLAE are generated and filled randomly with real numbers in the range from -100 to 100, excluding 0.

The modified approaches use the instructions_mm256_set1_pd, _mm256_loadu_pd, _mm256_sub_pd, _mm256_storeu_pd, _mm256_setzero_pd, _mm256_fmadd_pd to implement functionality for vector creation, data loading and storing, as well as performing arithmetic operations. The methodology for evaluating the efficiency of SIMD optimization involves measuring the execution time of both the classical and SIMD-optimized versions of the algorithms, calculating the speed-up factor for different SLAE sizes, and investigating the relationship between SIMD optimization efficiency and the given matrix size. To maximize computational performance, data were intentionally aligned according to vector size (32 bytes for AVX-256). For measuring execution time, the appropriate classes and functions from the <chrono> standard library were used.

Results

Table 1 presents the comparative results of computational experiments for solving SLAE using the classical Gauss method and its modified approach.

Table 1

Results of computational experiments for solving SLAE by the Gaussian method

№	SLAE order	NO-AVX, s	AVX-256, s	Acceleration a_1
1	40	0,0003	0,0002	1,50
2	100	0,0044	0,0022	2,00
3	200	0,0325	0,0142	2,29
4	300	0,1111	0,0450	2,47
5	400	0,2543	0,1019	2,50
6	500	0,4903	0,1960	2,50
7	740	1,5994	0,6143	2,60
8	1 000	3,9960	1,4987	2,67
9	1 500	13,4331	5,0253	2,67
10	2 000	31,8316	11,9038	2,67

Both solution approaches predictably show an increase in computation time as the system size grows, but the optimized approach demonstrates a slower growth, providing consistent speed-up compared to the classical method. The acceleration factor a_1 ranges from 1,5 до 2,67. As the order of the SLAE increases a_1 rises and stabilizes at around 2.67 for large systems (1×10^3–2×10^3).

Figure 2 shows the dependency of the SLAE solution time using the Gauss method on the system order.

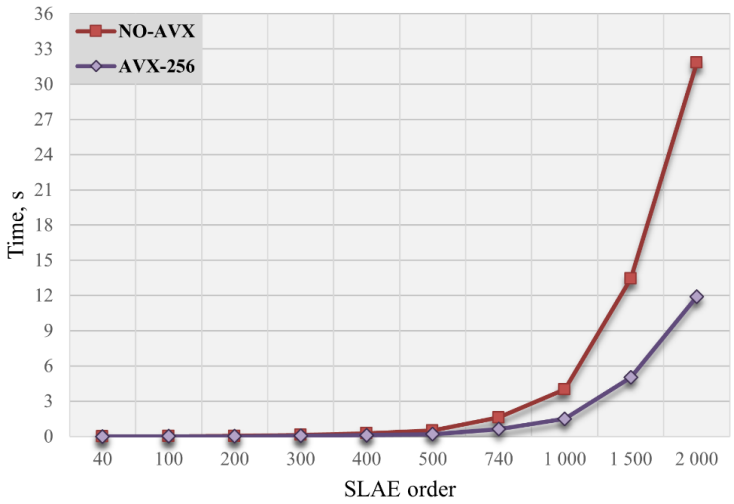

Fig. 2. The dependency of the SLAE solution time using the Gauss method on the system order is as follows: NO-AVX refers to the sequential method, and AVX-256 represents the modified approach using SIMD instructions

Table 2 presents the comparative results of computational experiments for solving SLAE using the classical method and its modified approach.

The optimized approach demonstrates significant acceleration compared to the classical one. The acceleration factor a_2 ranges from 2 до 5,9. As the order of the SLAE increases, a_2 rises and stabilizes around 5,7–5,8 for large systems (1×10^3–2×10^3).

Figure 3 shows the dependency of SLAE solution time using the conjugate gradient method on the system order.

Figure 4 demonstrates the achieved acceleration of both optimized methods in various experiments.

Table 2

The results of computational experiments for SLAE using the conjugate gradient method

№	SLAE order	NO-AVX, s	AVX-256, s	Acceleration a_2
1	40	0,0006	0,0003	2,00
2	100	0,0028	0,0008	3,50
3	200	0,0083	0,0019	4,37
4	500	0,0430	0,0078	5,51
5	1 000	0,1586	0,0274	5,79
6	2 000	0,5497	0,0951	5,78
7	5 000	3,2679	0,5542	5,90
8	10 000	11,4265	1,9982	5,72
9	15 000	26,2513	4,5469	5,77
10	20 000	45,1938	7,8757	5,74

From the obtained results of implementing both investigated methods, it is evident that the acceleration is insignificant for small orders of linear systems of equations, gradually increasing with the order of the system up to a certain point, after which it stabilizes at approximately the same level. For the optimized Gaussian method, this is around 2.67, while for the conjugate gradient method, it is approximately 5.75. This behavior can be explained by the overhead costs associated with vectorization for small system sizes.

Conclusions

As a result of the research, implementations of classical Gaussian and conjugate gradient methods for solving linear systems of equations have been developed, followed by the modification of computational approaches using SIMD technology and AVX-256 instructions. The modified approaches were implemented using the modern tools of the <immintrin.h> library in Visual Studio C++ with access to SIMD instructions. The experimental results confirm the feasibility of optimizing computational algorithms using SIMD. Thus, the implementation of the classical Gaussian method was accelerated by a factor of 2.67, while the conjugate gradient method achieved a speedup of 5.9 times for a system size of 5×10^3. To achieve maximum computational performance, data alignment was performed considering the vector size multiple.

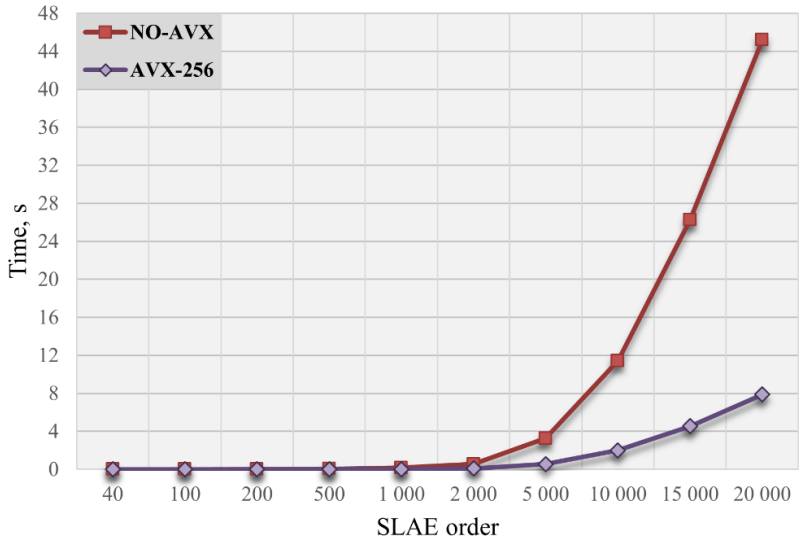

Fig. 3. The dependency of the solution time for SLAE using the conjugate gradient method on the order of the system: NO-AVX – sequential method; AVX-256 – modified approach using SIMD instructions

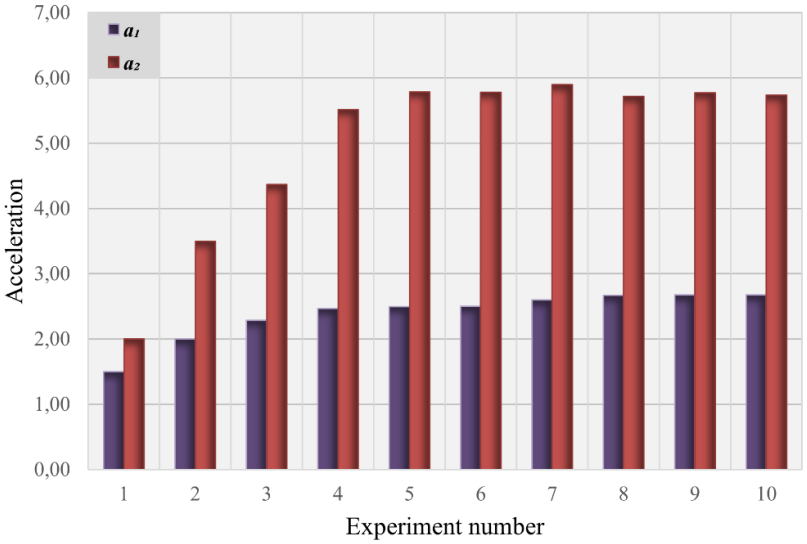

Fig. 4. Achieved acceleration of methods for solving SLAE in various experiments

The study confirmed that the effectiveness of SIMD optimization is insignificant for small orders of linear systems of equations and increases with the order up to a certain limit, after which it stabilizes. The impracticality of using SIMD optimization for small orders of linear systems can be explained by the overhead costs associated with vectorization.

The obtained results correlate with previously acquired data from studies on optimizing computational tasks using SIMD [20].

The presented approach opens new prospects for further development and improvement of numerical methods in the context of modern computational architectures, which may have wide applications in engineering calculations, computer graphics, machine learning, and other fields where computational efficiency is of high priority.

A direction for further research is the SIMD optimization of computational tasks in computer modeling, where the mathematical description of systems is based on numerical methods for solving SLAE.

References

1. Edamatsu T., Takahashi D. Fast Multiple-Precision Integer Division Using Intel AVX-512. *IEEE Transactions on Emerging Topics in Computing*. 2022. Vol. 11. P. 1–13. https://doi.org/10.1109/tetc.2022.3196147

2. Oo N. Z., Chaikan P. Fast Blockwise Matrix-Matrix Multiplication Using AVX and Prefetching on Shared Memory. *2022 13th Asian Control Conference (ASCC)*, Jeju, Korea, Republic of, 4–7 May 2022. 2022. https://doi.org/10.23919/ascc56756.2022.9828222

3. Accelerating Huffman Encoding Using 512-bit SIMD Instructions / Y. Yu et al. *IEEE Transactions on Consumer Electronics*. 2023. P. 554–563. https://doi.org/10.1109/tce.2023.3347229

4. An Improved BP Decoding of BATS Codes With Iterated Incremental Gaussian Elimination / J. Yang et al. *IEEE Communications Letters*. 2020. Vol. 24, no. 2. P. 321–324. https://doi.org/10.1109/lcomm.2019.2953699

5. Parallel SIMD - A Policy Based Solution for Free Speed-Up using C++ Data-Parallel Types / S. Yadav et al. *2021 IEEE/ACM 6th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, St. Louis, MO, USA, 15 November 2021. 2021. https://doi.org/10.1109/espm254806.2021.00008

6. de Dinechin B. D., Hascoët J., Desrentes O. In-Place Multicore SIMD Fast Fourier Transforms. *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, Boston, MA, USA, 25–29 September 2023. 2023. https://doi.org/10.1109/hpec58863.2023.10363536

7. Exploiting GPU/SIMD Architectures for Solving Linear-Quadratic MPC Problems* / D. Cole et al. *2023 American Control Conference (ACC)*, San Diego, CA, USA, 31 May – 2 June 2023. 2023. https://doi.org/10.23919/acc55779.2023.10155791

8. He X., Cai K. Disjoint-Set Data Structure-Aided Structured Gaussian Elimination for Solving Sparse Linear Systems. *IEEE Communications Letters*. 2020. Vol. 24, no. 11. P. 2445–2449. https://doi.org/10.1109/lcomm.2020.3012434

9. Roy S., Acharya D. P., Sahoo A. K. Incremental Gaussian Elimination Approach to Implement OMP for Sparse Signal Measurement. *IEEE Transactions on Instrumentation and Measurement*. 2020. Vol. 69, no. 7. P. 4067–4075. https://doi.org/10.1109/tim.2019.2947118

10. Adaptive Stochastic Conjugate Gradient Optimization for Backpropagation Neural Networks / I. A. Hashem et al. *IEEE Access*. 2024. P. 33757–33768. https://doi.org/10.1109/access.2024.3370859

11. Xiong K., Wang S. The Online Random Fourier Features Conjugate Gradient Algorithm. *IEEE Signal Processing Letters*. 2019. Vol. 26, no. 5. P. 740–744. https://doi.org/10.1109/lsp.2019.2907480

12. Wang Z., Zhang Y., Song Y. A Modified Conjugate Gradient Approach for Reliability-Based Design Optimization. *IEEE Access*. 2020. Vol. 8. P. 16742–16749. https://doi.org/10.1109/access.2020.2966661

13. Intel® Architecture Instruction Set Extensions Programming Reference. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/319433-024-697869.pdf (date of access: 10.10.2024).

14. Intel® 64 and IA-32 Architectures Optimization Reference Manual Volume 1. URL: https://www.intel.com/content/www/us/en/content-details/671488/intel-64-and-ia-32-architectures-optimization-reference-manual-volume-1.html (date of access: 10.10.2024).

15. Fast Longest Prefix Matching by Exploiting SIMD Instructions / Y. Ueno et al. *IEEE Access*. 2020. Vol. 8. P. 167027–167041. https://doi.org/10.1109/access.2020.3023156

16. Sovyn Y., Khoma V., Podpora M. Bitsliced Implementation of Non-Algebraic 8x8 Cryptographic S-Boxes Using x86-64 Processor SIMD Instructions. *IEEE Transactions on Information Forensics and Security*. 2022. P. 491–500. https://doi.org/10.1109/tifs.2022.3223782

17. SIMD-Constrained Lookup Table for Accelerating Variable-Weighted Convolution on x86/64 CPUs / Y. Naganawa et al. *IEEE Access*. 2024. P. 15800–15819. https://doi.org/10.1109/access.2024.3354720

18. Intel® Intrinsics Guide. URL: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html (date of access: 10.10.2024).

19. Osorio R. R., Rodriguez G. Truncated SIMD Multiplier Architecture for Approximate Computing in Low-Power Programmable Processors. *IEEE Access*. 2019. Vol. 7. P. 56353–56366. https://doi.org/10.1109/access.2019.2913743

20. Research of progressive tools of parallel computations with the use of simd architecture / O. O. Zhulkovskyi et al. *Informatics and mathematical methods in simulation*. 2023. Vol. 13, no. 3-4. P. 228–235. https://doi.org/10.15276/imms.v13.no3-4.228

Oleg Zhulkovskyi **Олег Жульковський**	PhD, Associate Professor of the Department of Software Systems, Dniprovsky State Technical University https://orcid.org/0000-0003-0910-1150 e-mail: olalzh@ukr.net	Кандидат технічних наук, доцент кафедри програмного забезпечення систем, Дніпровський державний технічний університет
Inna Zhulkovska **Інна Жульковська**	PhD, Associate Professor of Department of Cybersecurity and Information Technologies, University of Customs and Finance https://orcid.org/0000-0002-6462-4299 e-mail: inivzh@gmail.com	Кандидат технічних наук, доцент кафедри кібербезпеки та інформаційних технологій, Університет митної справи та фінансів
Hlib Vokhmianin **Гліб Вохмянін**	Master Student of the Department of Software Systems, Dniprovsky State Technical University https://orcid.org/0000-0002-9582-5990 e-mail: vohmyanin.yleb@gmail.com	Здобувач вищої освіти другого (магістерського) рівня, кафедра програмного забезпечення систем, Дніпровський державний технічний університет
Alexander Firsov **Олександр Фірсов**	PhD, Associate Professor of Department of Computer Technologies and Software Engineering, University of Customs and Finance https://orcid.org/0000-0002-6528-6447 e-mail: alexander.d.firsov@gmail.com	Кандидат технічних наук, доцент кафедри комп'ютерних технологій та інженерії програмного забезпечення, Університет митної справи та фінансів
Illia Tykhonenko **Ілля Тихоненко**	Student of Department of Cybersecurity and Information Technologies, University of Customs and Finance https://orcid.org/0009-0006-4158-0971 e-mail: tihilya1@gmail.com	Здобувач вищої освіти першого (бакалаврського) рівня, кафедра кібербезпеки та інформаційних технологій, Університет митної справи та фінансів