DOI 10.15276/imms.v13.no3-4.228
UDC 004.62

Informatics and Mathematical Methods in Simulation
Vol.13 (2023), No. 3-4, pp. 228-235

RESEARCH OF PROGRESSIVE TOOLS OF PARALLEL COMPUTATIONS WITH THE USE OF SIMD ARCHITECTURE

O.O. Zhulkovskyi[1], I.I. Zhulkovska[2], H.Ya. Vokhmianin[1], O.D. Firsov[2], V.A. Riabovolenko[2]

[1]Dniprovsky State Technical University
Dniprobudivska str., 2, Kamianske city, 51918, Ukraine
e-mail: olalzh@ukr.net
[2]University of Customs and Finance,
Volodymyr Vernadskyi str., 2/4, Dnipro, 49000, Ukraine
e-mail: inivzh@gmail.com

The current stage of development of processes and technologies requires continuous improvement of computer hardware performance, efficient use of its resources, processing of large amounts of data and support of the growing requirements of modern information systems. When processing large amounts of data, it is often necessary to use additional effective solutions to speed up information processing in addition to parallel computing. One such approach is to use the SIMD mechanism. The concept of SIMD instructions is a progressive solution for speeding up computations in tasks with large amounts of data, due to the ability to perform one operation on several data simultaneously. The purpose of the study is to evaluate the effectiveness of using SIMD instructions to improve the performance of software code execution when processing large data sets compared to traditional software tools. The paper solves the following tasks: develop an algorithm for implementing the classical task of multiplying ultra-large (up to 36×10^6 bytes) square data matrices using the built-in Microsoft Visual Studio ISO/IEC C++20 <immintrin.h> library with SIMD technology to parallelise the program at the data level; study the performance of the developed algorithm when processing a significant amount of data compared to the traditional approach. By implementing a modified matrix multiplication algorithm using SIMD technology, it was possible to speed up the computation on a PC with an Intel Core i7-12700H processor by 4.8 times with a data volume of $\sim9\times10^6$ bytes. The obtained results will be taken into account in the development of application software, including for efficient computer models of technological processes and systems.

Keywords: SIMD, vector register, data-level parallelism, intrinsic function, computing acceleration, big data, computer modelling.

Introduction. A significant role in modern programming, especially in computer modeling, is the problem of computational efficiency and speed, which becomes notably prominent during large data processing. There is a need to apply, in addition to parallel computations, additional effective solutions to accelerate information processing. One such approach is the use of the SIMD mechanism (Single Instruction, Multiple Data).

The SIMD concept has long been present in the architecture of modern PCs and is used in processor technologies. It provides data-level parallelism, allowing one operation to be performed on multiple data simultaneously, significantly increasing program performance. Despite being an old concept, modern processors typically apply SIMD extensions to enhance parallel computation performance.

Literature review. The SIMD instruction is an element of classification according to M. Flynn’s taxonomy for parallel processors, proposed in 1966 and later expanded in 1972 [1]. Modern PCs use this architecture in the form of integrating special instruction sets or command extensions to accelerate specific types of computations.

SIMD extensions are considered one of the significant features of modern general-purpose processors (GPPs), aimed at improving software performance with minimal hardware modifications [2].

Different processor manufacturers, such as Intel, AMD, etc., have their own Instruction Set Architecture (ISA) and SIMD microarchitectures. However, Intel has significantly expanded SIMD technologies from both hardware and software perspectives. In the context of microprocessor development, there is an increase in register bit-width from 64 to 512 bits and an increase in the number of vector registers from 8 to 32, providing more parallelism paths and reducing excessive data movement to cache memory [2].

SIMD vector extensions have become an integral part of high-performance processors. Various architectures, such as x86, ARM, MIPS, and PowerPC, have specific instruction sets and microarchitectures for SIMD vector extensions. Applying SIMD vectorization can significantly improve algorithm performance with minimal overhead on equipment. This is especially important for optimizing computational performance [3].

The «single instruction – multiple data» type of parallel processing (Fig. 1) represents a parallel computing technology where one instruction is executed over multiple data simultaneously [1-4].

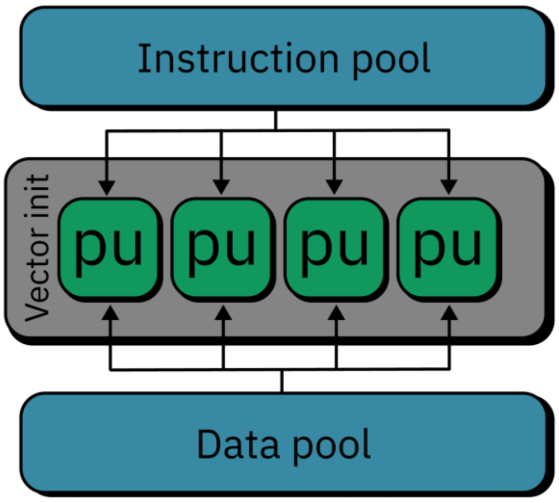

Fig. 1. SIMD architecture diagram

The Data pool is responsible for storing all processed data. The Instruction pool is responsible for storing executed instructions. The Vector unit is a key component of the SIMD architecture. It consists of several processing units (PU), each of which can process data in parallel. When a program runs on the SIMD architecture, the same instruction is sent from the instruction set to each processing device in the vector block. Each processing device then executes this instruction on different data from the set simultaneously. This allows SIMD to efficiently process large volumes of data, especially when one operation needs to be performed on each data element [1-4]. Accordingly, this technology finds its place in industries dominated by uniform operations, especially if they are applied to large volumes of data: graphics, signal processing, computer vision, computer modeling, etc.

A processor register used to store multiple data elements simultaneously in the form of a vector is called a vector register. Typically, vector elements represent a separate data value or component. Vector registers are part of SIMD hardware support, allowing the execution of a uniform operation over all vector elements simultaneously. They have a fixed width [5]. For example, a 128-bit vector register can store four float data type values. Thus, the size of the vector register determines the number of data elements that can be stored in each register. Depending on the specific processor architecture, support for different data types, including integers, floating-point numbers, etc., is possible, and the number of elements can reach 4, 8, 16, or more. Examples of vector registers from Intel include SSE (Streaming SIMD Extensions) and AVX (Advanced Vector Extensions) [2-4]. There are also NEON registers in some ARM processors [6].

SIMD technology is effective only in tasks where one operation can be applied to a large amount of data simultaneously. In other cases, its efficiency is either reduced or completely absent or negative. Among the disadvantages of SIMD, there is also a dependence on the architecture, as SIMD instructions may vary depending on the processor architecture, so a potential algorithm will work differently on different hardware systems [2, 7].

Recent studies have proposed significant improvements to multimedia applications aimed at increasing the performance, versatility, and programmability of computing cores. This involves the implementation of a massively parallel matrix SIMD core (CAMX) based on Content Addressable Memory, designed to work as an accelerator for processor cores. Notably, the study confirms the efficiency of CAMX with a detailed analysis of its operation during AES encryption [8].

Research in the field of graph computations using matrices also highlights the use of SIMD extensions on multi-core processors for efficient execution of graph algorithms. In particular, the graph algorithm compiler is adapted for the use of SIMD extensions on processors, leading to a significant acceleration of the naive multi-threaded implementation [9, 10].

To address the problem of sorting arrays containing a large amount of data, a parallel sorting implementation for MIPS processors is possible, based on concrete sorting networks and SIMD instructions [11].

Research Objective. The aim of this study is to evaluate the efficiency of using SIMD instructions to enhance the performance of programs during the processing of large data arrays compared to traditional software tools.

To achieve the set goal, the following tasks were formulated: Development of an algorithm for implementing the classic problem of multiplying ultra-large square data matrices using SIMD technology; Investigation of the performance of the developed algorithm with a significant amount of processed data compared to the traditional approach; Analysis of the obtained results and the development of a concept for the effective use of modern computing systems and tools to increase the productivity of computer applications.

Main Part. This work examines the evaluation of the efficiency of using SIMD instructions to accelerate computations in tasks of processing ultra-large data arrays. The research involves the development of a modified algorithm for solving the classic problem of processing large data arrays using SIMD instructions and analyzing the efficiency of applying this method compared to the traditional data processing approach.

One of the modern programming languages that provides and supports SIMD instructions is C++. Here, this technology is used thanks to compiler specifications and extensions, as well as through the use of specialized libraries.

For example, to use various sets of SIMD instructions from Intel, such as SSE, AVX, AVX2, AVX-512, etc., in C++, one can use special data types: __m128, __m256, __m512, etc. They represent vectors with 4, 8, or 16 elements of the corresponding type [7].

At the same time, the Intel Intrinsics library provides special functions for generating and using SIMD instructions [7]:

– The _mm_add_ps function is an intrinsic function (a special low-level function that provides access to processor instructions) for performing addition operations on several floating-point values in a vector register with a specified precision;

– The _mm_mul_pd function is also intrinsic, designed to perform multiplication operations on several double-precision floating-point values in a vector register.

In addition to the mentioned functions, there are many others, including subtraction, division, comparison, data loading and saving, bitwise arithmetic, element permutations, etc. Most of them can also define floating-point precision: single or double.

To use the listed instructions, the built-in library <immintrin.h> is used. It includes all Intel SIMD intrinsics, providing access to them. Also, for various SIMD extensions, different libraries are used. For example, for SSE, you should connect the header file <xmmintrin.h>, MMX is provided after connecting <mmintrin.h>, etc. But connecting one of the header files that provide access to the use of various extensions automatically connects all previous ones.

Thus, the variety of functions allows for the effective use of SIMD technology for various tasks and operations, leading to a significant increase in the productivity of computational tasks that support parallel data processing [7].

To maximize the power of processors with minimal development costs, it is advisable to use the NSIMD library, which abstracts SIMD programming and provides the following main paradigms [12]:

– Imperative programming provided by the NSIMD core and supports numerous CPU/SIMD extensions;

– Expression templates provided by a separate module that supports all architectures from the NSIMD core.

To achieve maximum performance, NSIMD uses optimized built-in compiler functions. Therefore, using any basic compiler can provide a SIMD abstraction library without significant costs. NSIMD supports work in all modern C++ programming language standards [12].

Another example of SIMD extension in C++ is the OpenMP standard for parallel programming, which supports the simd directive, which can be used to vectorize loops thanks to the #pragma omp simd construct. In this case, the compiler can ignore vector dependencies, considering the intention to execute several iterations simultaneously [13].

The mentioned ways of using SIMD technology allow for accelerating calculations by performing one operation on several data simultaneously.

For research purposes, an algorithm was developed to implement the classic problem of multiplying ultra-large square data matrices using the built-in Microsoft Visual Studio ISO/IEC C++20 <immintrin.h> library with SIMD technology for data-level program parallelization; an analysis of the performance of the developed algorithm was carried out with a significant amount of processed data compared to the traditional data processing approach.

For the experiments, the following PC infrastructure was used: Intel Core i7-12700H (14 cores, 2.3 GHz / 4.7 GHz); Goodram DDR4 (16 GB) × 2 = 32 GB; Microsoft Windows 10.

Program testing was conducted for matrices of size 100–3000, filled with random float type values (4 bytes) using a 128-bit register. Using wider registers may not be supported by some processors and requires more program resources. The obtained results are presented in Table 1.

Table 1

Computational Experiment Results

Matrix Dimension	Data Size, bytes	Execution Time, s		Acceleration a=s1/s2
		Traditional Algorithm (s1)	SIMD Algorithm (s2)	
100	4×10^4	$3{,}27\times10^{-4}$	$1{,}3\times10^{-4}$	2,53
500	10^6	$7{,}2\times10^{-2}$	$1{,}7\times10^{-2}$	4,24
1000	4×10^6	0,854	0,195	4,38
1500	9×10^6	2,89	0,604	4,78
2000	16×10^6	10,18	2,88	3,53
2250	$20{,}25\times10^6$	19,09	6,14	3,11
2500	25×10^6	30,96	8,97	3,45
2750	$30{,}25\times10^6$	48	14,55	3,30
3000	36×10^6	79,64	24,53	3,25

With the increase in matrix size, and therefore the volume of processed data, there is a natural increase in their processing time, regardless of the applied calculation algorithm. The use of SIMD technology has significantly accelerated the execution of calculations in all test cases and, especially, with a matrix dimension of 1500×1500 elements (data size 9×10^6 bytes).

Figure 2 shows that with an increase in matrix size in the range of 100–3000 float type elements, the computational data processing time increases significantly – from 33 ms to 80 s with the traditional (STD) algorithm and from 13 ms to 24.5 s with the SIMD algorithm.

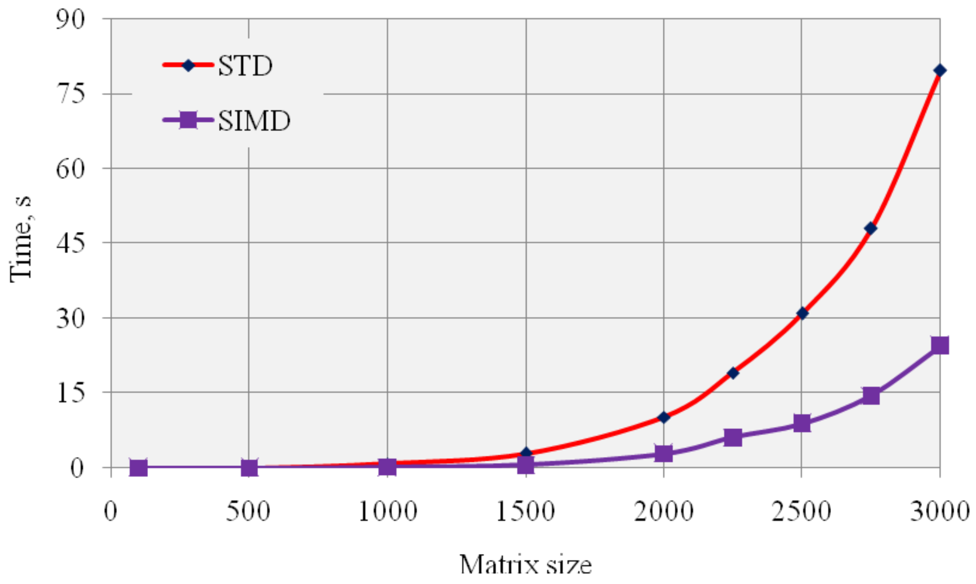

Fig. 2. Dependence of the implementation time of the compared algorithms on the dimensionality of the data matrices.

The acceleration of calculations when using SIMD technology compared to traditional data processing (Fig. 3) is between 2.53–4.78 and does not depend on the volume of processed data. The highest acceleration (~4.8), as already mentioned, is demonstrated by the modified algorithm with a matrix dimensionality of 1500 elements (~9×10^6 bytes), which should be taken into account during the development of application software.

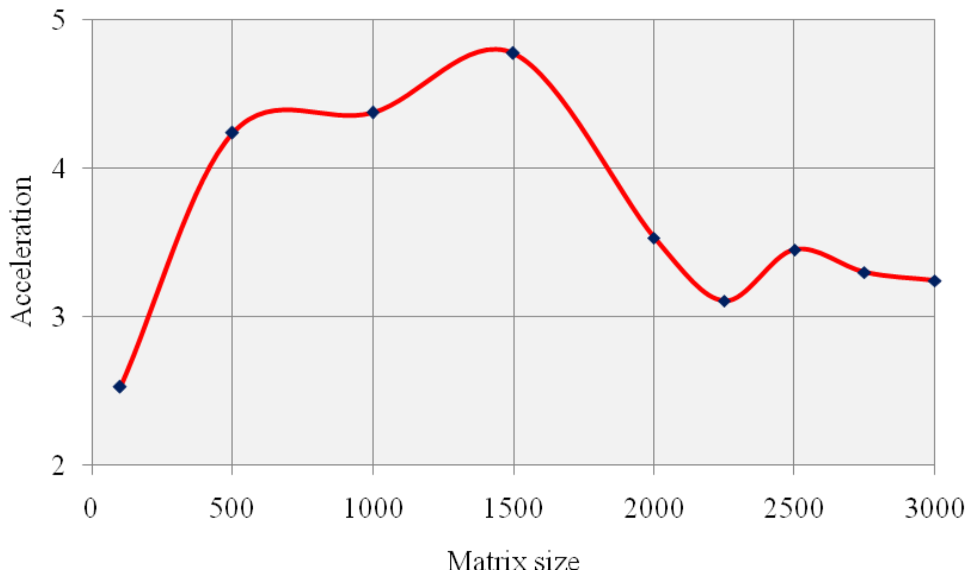

Fig. 3. Dependence of calculation acceleration due to the application of SIMD technology on the dimensionality of the data matrices.

Thus, the research has proven the effectiveness of using SIMD technology for solving tasks related to the processing of large volumes of data. The obtained data correspond with the results of known studies [3, 5] on the issue of enhancing the productivity of computer calculations using SIMD technology.

The use of the findings in conjunction with other alternative software tools to enhance computational productivity [14, 15] will contribute to the development of efficient computer models of technological processes and systems [16].

Conclusions. An algorithm has been developed to implement the classic problem of multiplying ultra-large square data matrices using SIMD technology. The performance of the developed algorithm was investigated with a significant amount of processed data (up to 36×10^6 bytes) compared to the traditional approach. The effectiveness of using advanced computing systems and SIMD-type tools to increase the productivity of application computer programs during the processing of large data volumes has been proven.

It has been established that the acceleration of calculations on a PC with an Intel Core i7-12700H processor, due to the application of SIMD technology compared to traditional data processing, is between 2.53–4.78 and does not depend on the volume of processed data. The highest acceleration (~4.8) is achieved with a matrix dimensionality of 1500 elements (~9×10^6 bytes), which should be taken into account during the development of application software, including for efficient computer models of technological processes and systems.

References

1. Flynn M. J. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*. 1972. Vol. C-21, № 9. P. 948-960. URL: https://doi.org/10.1109/TC.1972.5009071
2. Amiri H., Shahbahrami A. SIMD programming using Intel vector extensions. *Journal of Parallel and Distributed Computing*. 2020. Vol. 135. P. 83-100. URL: https://doi.org/10.1016/j.jpdc.2019.09.012
3. Xie C., Wu H. Zhou J. Vectorization Programming Based on HR DSP Using. *MDPI Journals Awarded Impact Factor*. 2023. Vol. 12, № 13. URL: https://doi.org/10.3390/electronics12132922
4. Lee S., Kye H. Efficient MIP volume rendering via fast SIMD interpolation and memory access reordering. *Multimedia Tools and Applications*. 2023. Vol. 82.

URL: https://doi.org/10.1007/s11042-022-13732-z

5. Chen Yi., Mendis C., Carbin M., Amarasinghe S. VeGen: a vectorizer generator for SIMD and beyond. *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2021. P. 902-914.
URL: https://doi.org/10.1145/3445814.3446692
6. Lindoso A., Garcia-Valderas M., Entrena L. Analysis of neutron sensitivity and data-flow error detection in ARM microprocessors using NEON SIMD extensions. *Microelectronics Reliability*. 2019. Vol. 100-101. URL: https://doi.org/10.1016/j.microrel.2019.06.038
7. Compiler intrinsics. URL: https://learn.microsoft.com/en-us/cpp/intrinsics
8. Kageyama K., Arai S., Hamano H., Kong X., Koide T., Kumaki T. Implementation of parallel AES Processing with CAM-based Massive-parallel SIMD Matrix Core. *2022 5th World Symposium on Communication Engineering (WSCE)*. 2022. Vol. 5.
URL: https://doi.org/10.1109/WSCE56210.2022.9916049
9. Zheng R., Pai S. Efficient Execution of Graph Algorithms on CPU with SIMD Extensions. *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2021. URL: https://doi.org/10.1109/CGO51591.2021.9370326
10. Mehrafsa A., Chester S., Thomo A. Vectorising k-Truss Decomposition for Simple Multi-Core and SIMD Acceleration. *2022 13th International Conference on Information, Intelligence, Systems & Applications (IISA)*. 2022. Vol. 13.
URL: https://doi.org/10.1109/IISA56318.2022.9904350
11. Brankovic S., Markovic A., Simic D., Rikalo A. Improving performance of sorting small arrays on MIPS CPUs using bitonic sort and SIMD instructions. *2019 27th Telecommunications Forum (TELFOR)*. 2020. Vol. 27.
URL: https://doi.org/10.1109/TELFOR48224.2019.8971325
12. NSIMD Documentation. URL: https://github.com/agenium-scale/nsimd
13. SIMD Extension. URL: https://learn.microsoft.com/en-us/cpp/parallel/openmp/openmp-simd
14. Zhulkovskyi O.O. 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, №4. P. 268-277. URL: https://doi.org/10.15276/imms.v11.no4.268
15. Zhulkovskyi O. O. Evaluation of the efficiency of the implementation of parallel computational algorithms using the <thread> library in C++. *Computer Systems and Information Technologies*. 2022. №3. P. 49-55.
URL: https://doi.org/10.31891/csit-2022-3-6
16. Zhulkovskii O. A., Panteikov S. P., Zhulkovskaya I. I. Information-Modeling Forecasting System for Thermal Mode of Top Converter Lance. *Steel in Translation*. 2022. Vol. 52, №5. P. 495-502. URL: https://doi.org/10.3103/s0967091222050138

ДОСЛІДЖЕННЯ ПРОГРЕСИВНИХ ЗАСОБІВ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ ІЗ ЗАСТОСУВАННЯМ SIMD АРХІТЕКТУРИ

О.О. Жульковський[1], І.І. Жульковська[2], Г.Я. Вохмянін[1], О.Д. Фірсов[2], В.А. Рябоволенко[2]

[1]Дніпровський державний технічний університет
2 Дніпробудівська вул., Кам'янське, 51918, Україна
e-mail: olalzh@ukr.net
[2]Університет митної справи та фінансів
2/4, Володимира Вернадського вул., Дніпро, 49000, Україна
e-mail: inivzh@gmail.com

Сучасний етап розвитку процесів та технологій потребує постійного підвищення продуктивності комп'ютерної техніки, ефективного використання її ресурсів, обробки великих обсягів даних та підтримки зростаючих вимог сучасних інформаційних систем. Під час обробки великих обсягів даних часто виникає необхідність застосування, окрім паралельних обчислень, додаткових ефективних рішень для прискорення обробки інформації. Одним з таких підходів є використання механізму SIMD. Концепція SIMD-інструкцій є прогресивним рішенням для пришвидшення обчислень у задачах з великим обсягом даних, завдяки можливості виконувати одну операцію над декількома даними одночасно. Метою дослідження є оцінка ефективності використання SIMD-інструкцій для підвищення продуктивності виконання програмного коду під час обробки великих масивів даних у порівнянні з традиційними програмними засобами. В роботі вирішені наступні задачі: розроблено алгоритм реалізації класичної задачі перемноження надвеликих (до 36×10^6 байт) квадратних матриць даних із використанням вбудованої бібліотеки Microsoft Visual Studio ISO/IEC C++20 <immintrin.h> з технологією SIMD для розпаралелювання програми на рівні даних; досліджено продуктивність виконання розробленого алгоритму при значній кількості оброблюваних даних у порівнянні з традиційним підходом. розроблено алгоритм реалізації класичної задачі перемноження надвеликих квадратних матриць даних із використанням; виконано аналіз продуктивності розробленого алгоритму при значній кількості оброблюваних даних у порівнянні з традиційним підходом до обробки даних Тестування програмного забезпечення проводилось для матриць розмірністю 100–3000, заповнених випадковими значеннями типу float (4 байти) із використанням 128-бітного регістру SIMD-архітектури. За рахунок впровадження модифікованого алгоритму перемноження матриць з використанням технології SIMD вдалося пришвидшити виконання обчислень на PC з процесором Intel Core i7-12700H у 4,8 рази при обсягах оброблюваних даних $\sim9\times10^6$ байт. Отримані результати будуть враховуватися під час розроблення прикладного програмного забезпечення, у тому числі для ефективних комп'ютерних моделей технологічних процесів та систем.

Ключові слова: SIMD, векторний регістр, паралелізм на рівні даних, інтринзична функція, пришвидшення обчислень, великі дані, комп'ютерне моделювання