ENERGETIKA. 2025
T. 71. Nr. 1. P. 23–33

ISSN 0235-7208 eISSN 1822-8836
DOI: https://doi.org/10.6001/energetika.2025.71.1.2

Using asynchronous programming to improve computer simulation performance in energy systems

Oleg Zhulkovskyi[1],

Inna Zhulkovska[2],

Petro Kurliak[3],

Oleksandr Sadovoi[4],

Yuliia Ulianovska[5],

Hlib Vokhmianin[6]

[1] *Department of Software Systems,*
Dniprovsky State Technical University,
Kamianske, Ukraine
Email: olalzh@ukr.net

[2] *Department of Cybersecurity*
and Information Technologies,
University of Customs and Finance,
Dnipro, Ukraine
Email: inivzh@gmail.com

[3] *Department of Electrical Power Engineering,*
Ivano-Frankivsk National Technical University
of Oil and Gas,
Ivano-Frankivsk, Ukraine
Email: petro.kurliak@nung.edu.ua

[4] *Department of Electrical Engineering*
and Electromechanics,
Dniprovsky State Technical University,
Kamianske, Ukraine
Email: sadovoyav@ukr.net

[5] *Department of Computer Science*
and Software Engineering,
University of Customs and Finance,
Dnipro, Ukraine
Email: yuliyauyv@gmail.com

[6] *Department of Software Systems,*
Dniprovsky State Technical University,
Kamianske, Ukraine
Email: vohmyanin.yleb@gmail.com

Due to the progressing complexity of modern energy systems, the need to forecast energy consumption and generation, optimise processes and develop new technologies in the energy sector, analyse scenarios for the development of energy systems and elaborate a strategy for their development, modelling and simulation is of particular relevance in this industry. The growing need to improve the productivity of computer simulation in the energy industry is effectively addressed by utilising modern computer architectures and advanced software tools that provide acceleration for computationally intensive tasks. Research presented in this paper focuses on enhancing the performance of computationally intensive algorithms using the Thomas algorithm by employing modern asynchronous programming techniques. The work implements classical and develops and implements asynchronous computational algorithms of the sweep method with subsequent assessment of the time and efficiency of their execution for the order of systems of linear equations (SLAEs) up to 5×10^7. The program code was developed using Microsoft Visual Studio C++ and the standard template for asynchronous programming. The numerical experiments showed the possibility of increasing of the implementation speed of the asynchronous algorithm by 1.87–2.91 times. Research results correspond with the literature data and the results previously obtained by the authors in similar studies using alternative parallel programming software. In general, the results of this study determine the potential for further improvement and development of methods and technologies for parallel implementation of computational tasks using the Tridiagonal Matrix Algorithm. These approaches can be extended to developing various computer models of energy processes and systems based on the solution of SLAEs with tridiagonal matrices on computers with multiprocessor or multi-core architectures.

Keywords: computer model, computational acceleration, asynchronous programming, computational algorithm, numerical solution of SLAE

INTRODUCTION

Due to the progressive complexity of modern energy systems, the need to forecast energy consumption and generation, optimise processes and develop new technologies in the energy sector, analyse scenarios of energy systems development, and elaborate a strategy for their development, modelling and simulation are of particular relevance in energy industry [1, 2].

Computer modelling becomes especially relevant given the rapid increase in computing power, which means storing and processing large amounts of data in real time. Software is also being developed and improved to facilitate the effective use of the latest computer architectures and the realisation of modern virtual models of complex energy processes and systems.

The main approach to improving the efficiency of computer modelling in the energy industry is improving the speed of calculations in the software implementation of numerical methods underlying the mathematical models of the processes and phenomena under study. When developing models related to actual problems of power engineering, such as numerical modelling of physical processes (heat transfer and deformations in elements of power equipment, fluid dynamics in turbines, compressors, pipelines, electromagnetic fields in electric machines, transformers, power lines), search for optimal modes of operation of power plants, the modified Gauss method (TDMA – Tridiagonal Matrix Algorithm) or Thomas algorithm, also known as the sweep method, is most frequently used. This method's computational complexity of solving systems of linear equations (SLAEs) is linear concerning the number of equations, making it more suitable for large systems compared to general methods for solving SLAEs [3, 4]. In addition, the widespread use of unconditionally stable, more accurate, reliable, and efficient implicit difference schemes in the practice of computer modelling instead of low-performance explicit schemes makes this method preferable [5].

An important problem in the numerical solution of SLAEs is the performance of the TDMA, which demands an increase in computational resources as the system dimensionality grows. Therefore, there is a need to optimise the algorithm of this method and explore ways for a more efficient use of modern hardware and software computing resources.

One of the approaches to increasing the efficiency of computational algorithms and improving the performance and scalability of modelling is asynchronous computing – a method of parallel task execution in which operations can start and run independently in different threads without blocking one another. Moreover, modern multi-core computing architectures and programming languages provide powerful tools for implementing asynchronous programming.

The priority goal of this research is to use modern asynchronous programming technologies for parallelising computations to improve the efficiency of implementing computer simulation tasks of energy processes and systems using TDMA.

To achieve this goal, the following objectives were set for this study:

- development and implementation of a program of sequential and parallel algorithms for the TDMA;
- comparative analysis of the performance of serial and parallel algorithms for the TDMA using asynchronous programming tools for large-order SLAEs;
- elaboration of a strategy for developing approaches to enhance the performance of computer modelling using parallel computing algorithms of the sweep method, their scaling to the development of computer models based on the solution of SLAEs with tridiagonal matrix on computers with multiprocessor architecture.

LITERATURE REVIEW

Modern computers with multiprocessor architectures provide high performance through parallel computing. However, the complexity of using multiprocessor architectures, the necessity of redesigning existing algorithms, and the insufficient study of appropriate software tools that enable efficient parallel computations often lead to the underutilisation of computing power in computer modelling. In other words, numerical methods and approaches to computer modelling do not always keep pace with the new capabilities of computing hardware and software [6].

The practical application of parallel computing requires the use of specialised technologies and tools, including the following key ones:

• parallelism at the level of instructions and data [7, 8]: SIMD (Single Instruction, Multiple Data), MIMD (Multiple Instruction, Multiple Data), ILP (Instruction-Level Parallelism), MLP (Memory-Level Parallelism);

• platforms and libraries for parallel computing [9, 10]: OpenMP (Open Multi-Processing), MPI (Message Passing Interface), CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language);

• parallelism at the level of threads and tasks [11, 12]: asynchronous programming, the library for parallel programming TBB (Intel Threading Building Blocks), OpenMP.

Each of the aforementioned technologies and tools used for parallel computing have advantages and preferred practical application areas. To varying degrees, all of them can effectively optimise the computer modelling of complex processes and systems, which is an area of scientific interest [5, 13, 14] for the authors of this paper.

Thus, [15] optimised a background extraction method based on GMM (Gaussian Mixture Model), which is widely used for detecting moving objects. Since GMM applies arithmetic operations to each image pixel, vectorisation and implementation on modern SIMD architectures are feasible. An efficient vectorisation of GMM data for two different Intel architectures using SSE2 instructions was proposed. Performance evaluation showed that a speedup of 1.8 times was achieved using even a single processor core.

In [16], the use of multithreading and SIMD-vectorisation techniques is discussed for performing computationally intensive parallel computations of power flows in power systems. Two levels of parallelism are proposed: inter-model (parallel processing of different models) and intra-model (parallel processing of computations within a single model). The results of simulations of a network with 70,000 nodes demonstrate a tenfold acceleration of calculations.

Performance improvements of the Finite Difference Time Domain (FDTD) method using OpenMP and MPI, as well as GPU and SIMD vectorisation, are discussed in [17]. Here, FDTD is applied to simulate computational electrodynamics on a fine computational grid, which often leads to significant computational slowdowns. The approach considered allows for the accurate computation of both the shortest electromagnetic waves and the fine geometric details of the model.

The application of OpenMP for parallel modelling of geoengineering tasks using the Finite-Discrete Element Method (FDEM) is considered in [18]. The research investigates computational performance on multi-core processors, specifically the AMD Ryzen Threadripper PRO 5995WX (64/128 cores/threads) and AMD EPYC 7T83 (128/256 cores/threads). The computational domain was divided into 33–3304 k elements. An increase in computational performance of up to 43 times is demonstrated depending on the number of elements in the model due to modern multi-core architectures and advanced parallel programming technologies.

The study [19] presents parallel implementations of a dynamic explicit CDM (Central Difference Method) algorithm for efficient modelling of the coin minting process with complex relief patterns using three approaches: OpenMP, MPI and hybrid MPI/OpenMP. The challenge of accurately capturing small relief patterns approximately 50 microns in size is addressed by partitioning the computational domain into at least seven million tetrahedral elements. Such computational volumes make the use of traditional sequential programs impractical. The model implementation using MPI showed a maximum speedup of 9.5 on a single compute node (12 cores), while the hybrid MPI/OpenMP implementation showed a speedup factor of 136 in a cluster (6 compute nodes and 12 cores per compute node).

Of particular interest to the authors is [20], which considers the application of parallel computational algorithms based on the Finite Difference Method (FDM) to solve steady-state heat conduction equations in a three-dimensional setting. The advantages of the developed approach are demonstrated in comparison with traditional tools for sequential computing and the results of CFD modelling (Computational Fluid Dynamics). The paper describes program code development for parallel calculating three-dimensional temperature fields using MPI, OpenMP and CUDA technologies.

In [21], TBB is described for parallel processing of video streams in the TIS (Tactical Integration System) for maritime patrol aircraft. C++ software using TBB for multicore computers was developed, which allowed for a speedup of video stream processing by more than 1.3 times in parallel mode compared to serial mode at different quality factors. This enhancement enables the TIS system to operate more efficiently under real conditions.

A new approach to the parallel implementation of the Doolittle Algorithm using TBB, increasing the efficiency of multiprocessor systems is presented in [22]. The C++ implementation of the Parallel Doolittle Algorithm (PDA) using TBB showed a significant speedup compared to the sequential version for matrices of different orders. The results indicated that PDA utilises all processor cores and provides faster processing of systems of linear equations.

The trend of increasing computational performance by utilising modern computer architectures and improving parallel computing algorithms is discussed in [23]. It considers the parallel improvement of the Mean Shift algorithm using TBB on multiprocessor systems and proposes parallel improvement based on TBB clustering of Mean Shift, the main image segmentation stage, enabling significant acceleration of the process.

The literature analysis shows that asynchronous programming and the study of parallelism at the level of threads and tasks are insufficiently used to optimise computer modelling of complex processes and systems. This observation was the impetus for the present research, the results of which are described below.

RESEARCH METHODOLOGY

As stated earlier, when developing simulations of many physical processes, the TDMA is most often used as the most optimal method for solving SLAEs with a tridiagonal matrix [3–5].

To solve SLAEs with a tridiagonal matrix by the TDMA, a system of n equations is written in the following canonical form

$$a_i x_{i-1} - c_i x_i + b_i x_{i+1} + f_i = 0,$$

$$a_i,\ b_i \neq 0,\ a_1 = 0,\ b_n = 0,\ i = \overline{1,n} \qquad (1)$$

The tridiagonal form of the matrix allows organising the computations by the Gauss method to avoid operations with zero elements, thus significantly reducing the volume of computations.

The algorithm for the numerical implementation of the above method in the case of a right sweep includes the following steps:

forward pass of the sweep method:

$$\begin{aligned} \alpha_2 &= \frac{b_1}{c_1},\ \alpha_{i+1} = \frac{b_i}{c_i - a_i\alpha_i},\ i = \overline{2, n-1}, \\ \beta_2 &= \frac{f_1}{c_1},\ \beta_{i+1} = \frac{a_i\beta_i + f_i}{c_i - a_i\alpha_i},\ i = \overline{2, n-1}; \end{aligned} \qquad (2)$$

backward pass of the sweep method:

$$\begin{aligned} x_n &= \frac{a_n\beta_n + f_n}{c_n - a_n\alpha_n}, \\ x_i &= \alpha_{i+1}x_{i+1} + \beta_{i+1},\ i = \overline{n-1, 1} \end{aligned} \qquad (3)$$

For computational stability of the TDMA, it is necessary to fulfil the condition of diagonal dominance

$$|c_1| \geq |b_1|,\ |c_n| \geq |a_n|,\ |c_i| > |a_i| + |b_i|,\ i = \overline{2, n-1}$$

The formulas for the left sweep are written out in the same way:

forward pass of the sweep method:

$$\begin{aligned} \xi_n &= \frac{a_n}{c_n},\ \xi_i = \frac{a_i}{c_i - b_i\xi_{i+1}},\ i = \overline{n-1, 2}, \\ \eta_n &= \frac{f_n}{c_n},\ \eta_i = \frac{b_i\eta_i + f_i}{c_i - b_i\xi_{i+1}},\ i = \overline{n-1, 2}; \end{aligned} \qquad (4)$$

backward pass of the sweep method:

$$\begin{aligned} x_1 &= \frac{b_1\eta_2 + f_1}{c_1 - b_1\xi_2}, \\ x_{i+1} &= \xi_{i+1}x_i + \eta_{i+1},\ i = \overline{1, n-1} \end{aligned} \qquad (5)$$

By combining left and right sweeps, we obtain a counter-sweep method that allows parallel implementation in two different threads, including the use of asynchronous programming tools.

To do this, we divide the system between two threads, each of which will operate on only its half of the system equations numbered $i = \overline{1,p}$ and $i = \overline{p,n}$, respectively, where $p = n/2$.

The parallel execution involves:

finding the TDMA coefficients by (2) for $i=\overline{1,p}$ in the main thread;

finding the TDMA coefficients by (4) for $i=\overline{p,n},$ in the parallel thread, which, for example, is involved in an asynchronous call;

conjugation of solutions in the form (3) and (5) for $i = p$ with finding x_p from the system

$$\begin{cases} x_p = \alpha_{p+1} x_{p+1} + \beta_{p+1} \\ x_{p+1} = \xi_{p+1} x_p + \eta_{p+1} \end{cases}$$

finding all x_i for $i=\overline{1,p-1}$ by (3) and all x_i for $i=\overline{p+1,n}$ by (5).

The program code developed for numerical experiments implements asynchronous programming using the std::async template and std::launch::async function from the standard <future> library in Microsoft Visual Studio C++ IDE. Thanks to the launch policy, two asynchronous tasks are explicitly created for parallel execution of calculations, allowing the work to be divided between the main thread and two additional threads. Thus, running in a separate thread, the first asynchronous task performed the right sweep, calculating the TDMA coefficients according to (2). In parallel with the first asynchronous task, the main thread executed the left sweep, calculating the TDMA coefficients according to (4). After the completion of both passes, the main thread, which is blocked, waited for the first asynchronous task to finish. Asynchronously, in the new thread, the second task was started, which performed the reverse right sweep, calculating the roots of the equation according to (3). In parallel with the second asynchronous task, the main thread performed the reverse left sweep, also calculating the roots of the equation according to (5).

Timing of the computational experiment was performed using the steady_clock class from the special standard library <chrono>, which provides access to a stable clock and is optimised for measuring time intervals.

The size of the SLAE was varied in the range of 1×10^5–5×10^7, and the values of the equation coefficients were generated randomly (subject to the diagonal dominance condition of the matrix) into variables of the standard hardware-supported double type.

The developed code effectively used parallelism to speed up computations related to solving SLAEs. In a multithreaded environment, especially on multicore processors, asynchronous execution reduced the total execution time due to the simultaneous processing of different parts of the task, according to Amdahl's law.

RESULTS AND DISCUSSION

A laptop with the following characteristics was used for the research (Table 1).

Table 1. **Test environment**

CPU	Intel Core i5-8400 (6 cores, 2.8 GHz), cache 9 MB
RAM	Goodram DDR4 (4 GB, 2666 MHz, 21300 MB/s) × 4
OS	Microsoft Windows 10
IDE	Microsoft Visual Studio C++
Programming technology	std::async template, <future> library

Tables 2 and 3 present the results of computational experiments showing the comparative temporal characteristics of the computational efficiency of classical algorithms and the parallel variant of the counter sweep.

Figures 1, 2, and 3 present the most significant results, emphasising the practical significance of the research.

The experimental data obtained show (Figs 1, 2) that the use of the counter sweep method without asynchronous programming increases the performance of the computational algorithm compared to the traditional right sweep by 1.47–1.84 times in the considered range of SLAE order variation. Compared to the traditional right sweep, the increase in computation speed due to asynchronous calculation for the counter sweep method was 1.03–2.91 times.

The regression equations describing the linear approximation of the numerical results obtained (Tables 1, 2; Figs. 1, 2) are presented next:

$$t1 = 0.125 \times 10^{-3} + 1.88 \times 10^{-8}\, n;$$

$$t2 = 0.102 \times 10^{-3} + 1.22 \times 10^{-8}\, n;$$

$$t3 = 0.445 \times 10^{-4} + 6.97 \times 10^{-9}\, n$$

Table 2. **Results of the computational experiment for sequential algorithms**

SLAE order	Sequential algorithms		
	right sweep (*t*1, *s*)	counter sweep (*t*2, *s*)	*s*1 = t1/t2
1×10^5	0.001969	0.001070	1.84019
2×10^5	0.003778	0.002325	1.62495
3×10^5	0.005328	0.003411	1.56201
4×10^5	0.007292	0.004712	1.54754
5×10^5	0.009424	0.005836	1.61480
6×10^5	0.010809	0.007042	1.53493
7×10^5	0.012564	0.008276	1.51812
8×10^5	0.014579	0.009442	1.54406
9×10^5	0.016272	0.011038	1.47418
1×10^6	0.018275	0.011839	1.54363
2.5×10^6	0.047037	0.030118	1.56176
5×10^6	0.092932	0.058827	1.57975
1×10^7	0.184777	0.119720	1.54341
1.5×10^7	0.301776	0.198740	1.51845
2×10^7	0.370866	0.238921	1.55225
3.5×10^7	0.654413	0.418034	1.56545
5×10^7	0.942063	0.610584	1.54289

Table 3. **Results of the computational experiment for the parallel algorithm**

SLAE order	Sequential algorithms		
	counter sweep (*t*3, *s*)	*s*2 = t1/t3	*s*3 = t2/t3
1×10^5	0.001911	1.030351	0.559916
2×10^5	0.002502	1.509992	0.929257
3×10^5	0.003361	1.585242	1.014877
4×10^5	0.003660	1.992350	1.287432
5×10^5	0.004265	2.209613	1.368347
6×10^5	0.005109	2.115678	1.378352
7×10^5	0.005508	2.281046	1.502542
8×10^5	0.006095	2.391961	1.549139
9×10^5	0.007178	2.266927	1.537754
1×10^6	0.007337	2.490800	1.613602
2.5×10^6	0.016169	2.909085	1.862700
5×10^6	0.032315	2.875816	1.820424
1×10^7	0.064159	2.879986	1.865989
1.5×10^7	0.115512	2.612508	1.720514
2×10^7	0.130305	2.846138	1.833552
3.5×10^7	0.236075	2.772055	1.770768
5×10^7	0.355541	2.649661	1.717338

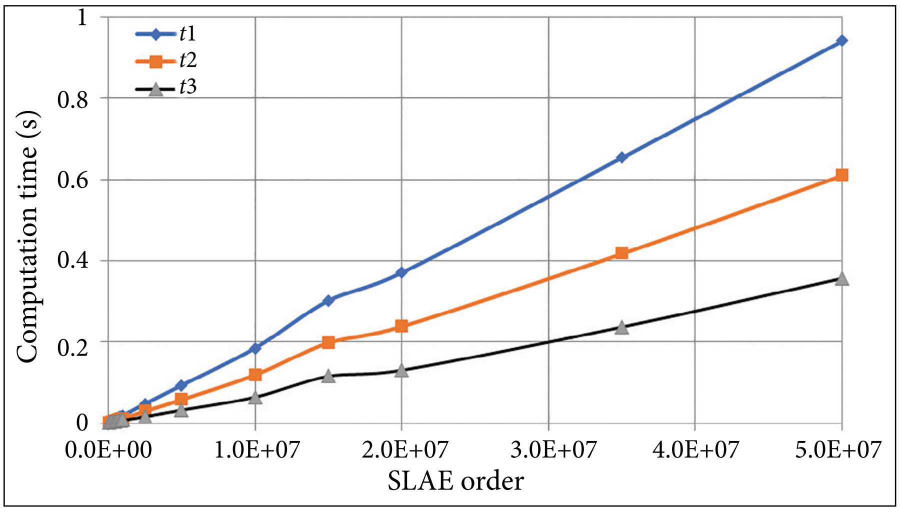

Fig. 1. Time of realisation of the sweep method: *t*1 – right, *t*2 – counter (sequential calculation); *t*3 – counter (parallel calculation) for SLAE order 1×10^5–5×10^7

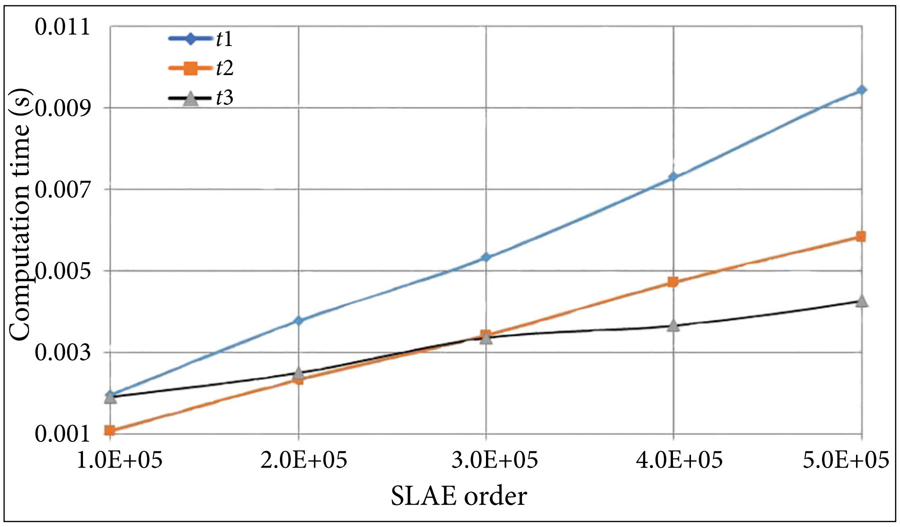

Fig. 2. Time of realisation of the sweep method: *t*1 – right, *t*2 – counter (sequential calculation); *t*3 – counter (parallel calculation) for the SLAE order 1×10^5–5×10^5

None of the considered TDMA implementations under the conditions of the used infrastructure of computational experiments and the investigated range of SLAE order showed a computation time of more than 1 s (Fig. 1).

As expected and also confirmed by similar experiments of the authors using alternative software tools for parallel implementation of TDMA, the time of asynchronous implementation of the algorithm for relatively small (up to 3×10^5) order of SLAE was less than for its synchronous implementations. Under such experimental conditions, *t*3 > *t*2 (Fig. 2) and *s*3 < 1 (Fig. 3). For higher SLAE order, the computational process started to accelerate (*s*3 > 1), and in the range of SLAE order 2.5×10^6–1.0×10^7, the acceleration reached its maximum value at ~1.9.

The slowdown of computations for SLAEs of order less than 3×10^5 was caused by the irrational use of machine time for creating computational threads and subsequent synchronisation of computations, i.e., the resulting time costs exceeded the time of direct computations. Thus,

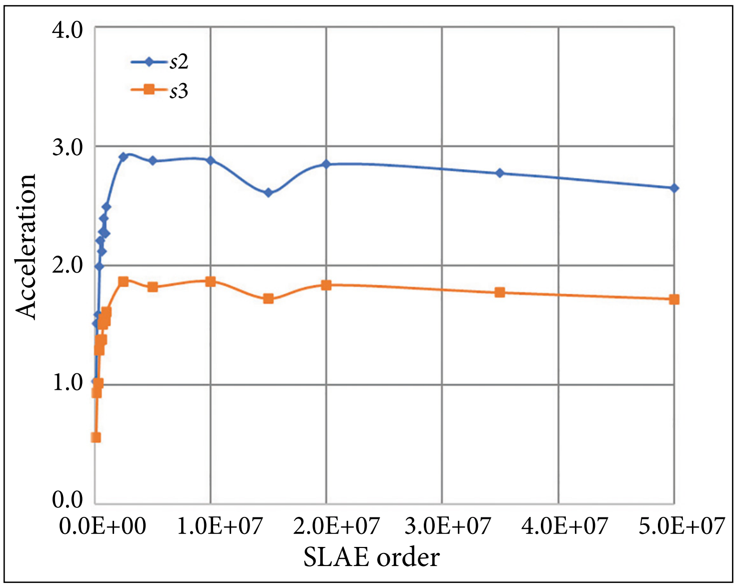

Fig. 3. Acceleration of the parallel implementation of the sweep method with respect to the sequential implementation of the right (*s*2) and counter (*s*3) sweep methods

observing the inexpediency of using asynchronous computing techniques up to a certain order of SLAE becomes relevant. In this case, the impractical order of SLAE for using asynchronous programming tools was less than 3×10^5.

Asynchronous computing for parallel implementation of TDMA increased the algorithm performance by 1.3–1.9 times in the range of SLAE order of 4×10^5–5×10^7. In the range of SLAE order 2.5×10^6–1.0×10^7, the increase in the speed of asynchronous calculations compared to the implementation of the classical right sweep method amounted to 291%.

Thus, the effectiveness of using asynchronous programming tools to improve the performance of the TDMA computational algorithms across a wide range of SLAE orders was demonstrated. The results align with the data from available sources of information and with results previously obtained by the authors in similar studies using alternative software tools.

CONCLUSIONS

As a result, computational algorithms of sequential and parallel (using asynchronous programming tools) sweep methods were developed and implemented, followed by the evaluation and comparative performance characterisation of their implementation for significant (up to 5×10^7) SLAE orders. In this case, the threshold of the SLAE order at which the use of asynchronous programming tools seems to be an effective solution for modelling energy processes and systems is 3×10^5. Approximations of the calculated data with their linear approximations and corresponding regression functions are obtained.

The conducted research proved the feasibility of using progressive multiprocessor architectures and specialised software to improve the efficiency of computer modelling in general. The use of asynchronous programming technologies for implementing popular SLAE solution methods resulted in an increase in calculation speed by 1.87–2.91 times in the range of SLAE orders 2.5×10^6–1.0×10^7 due to the creation of asynchronous tasks for parallel execution of calculations.

The results of calculations correspond with similar data from available literature sources and with data previously obtained by the authors in similar studies using alternative software tools.

The research results, in general, demonstrate the potential for further improvement and development of methods and technologies for parallel realisation of computational tasks based on TDMA. These approaches can be extended to developing various computer models of energy processes and systems based on the solution of SLAEs with tridiagonal matrices on computers with multiprocessor or multi-core architectures. When implementing the developed simulations on multiprocessor computers, parallel computing allows for dividing the computational load among several processors or cores, leading to faster execution of considered modelling tasks [24].

Received 28 December 2024
Accepted 30 December 2024

References

1. Subramanian A. S. R., Gundersen T., Adams T. A. II. Modeling and simulation of energy systems: a review. *Processes*. 2018. Vol. 6(12). P. 238. https://doi.org/10.3390/pr6120238
2. Bendigiri P., Rao P. Energy system models: a review of concepts and recent advances using bibliometrics. *International Journal of Sustainable Energy*. 2023. Vol. 42(1). P. 975–1007. https://doi.org/10.1080/14786451.2023.2246082
3. Ford W. Algorithms. In: *Numerical Linear Algebra with Applications*. Amsterdam. 2015. P. 163–179. https://doi.org/10.1016/B978-0-12-394435-1.00009-0
4. Rapp B. E. Numerical methods for linear systems of equations. In: *Microfluidics: Modelling, Mechanics and Mathematics*. Amsterdam. 2017. P. 497–535. https://doi.org/10.1016/b978-1-4557-3141-1.50025-3
5. Zhulkovskyi O., Savchenko I., Zhulkovska I., Petrenko I., Davitaia O., Titiov A. Features of mathematical simulation of the processes of combined heat transfer in waveguides. *Proceedings 2022 IEEE 4th International Conference on Modern Electrical and Energy System (MEES 2022)*. 2022. P. 452–456. https://doi.org/10.1109/MEES58014.2022.10005676
6. Bouras M., Idrissi A. A survey of parallel computing: challenges, methods and directions. In: *Modern Artificial Intelligence and Data Science*. Cham. 2023. Vol. 1102. P. 67–81. https://doi.org/10.1007/978-3-031-33309-5_6
7. Mustafa D., Alkhasawneh R., Obeidat F., Shatnawi A. S. MIMD programs execution support on SIMD machines: a holistic survey. *IEEE Access*. 2024. Vol. 12. P. 34354–34377. https://doi.org/10.1109/ACCESS.2024.3372990
8. Kiriansky V., Xu H., Rinard M., Amarasinghe S. Cimple: instruction and memory level parallelism. *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques (PACT '18), November 1–4, 2018, New York, USA*. P. 1–16. https://doi.org/10.1145/3243176.3243185
9. Mohamed K. S. Parallel computing: OpenMP, MPI, and CUDA. In: *Neuromorphic Computing and Beyond*. Cham: Springer, 2020. P. 63–93. https://doi.org/10.1007/978-3-030-37224-8_3
10. Khairy M., Wassal A. G., Zahran M. A survey of architectural approaches for improving GPGPU performance, programmability and heterogeneity. *Journal of Parallel and Distributed Computing*. 2019. Vol. 127. P. 65–88. https://doi.org/10.1016/j.jpdc.2018.11.012
11. Belson B., Holdsworth J., Xiang W., Philippa B. A survey of asynchronous programming using coroutines in the Internet of Things and embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*. 2019. Vol. 18. No. 3. P. 1–21. https://doi.org/10.1145/3319618
12. Balaji P. Intel Threading Building Blocks. In: *Programming models for parallel computing*. Cambridge: MIT Press, 2015. P. 353–372. https://ieeexplore.ieee.org/document/7352784
13. Zhulkovskii O. A., Panteikov S. P., Zhulkovskaya I. I. Information-modeling forecasting system for thermal mode of top converter lance. *Steel Translation*. 2022. Vol. 52. No. 5. P. 495–502. https://doi.org/10.3103/S0967091222050138
14. Zhulkovskyi O., Panteikov S., Zhulkovska I., Kashcheev M., Leshchenko E. Heat Transfer Calculation for Numerical Simulation of Thermal Mode of Slag-Splashing Lance in the Forecasting System. In: *Mathematical Modeling and Simulation of Systems. Lecture Notes in Networks and Systems* Vol. 1091. Cham: Springer, 2024. P. 70–81. https://doi.org/10.1007/978-3-031-67348-1_6
15. Mabrouk L., Houzet D., Huet S., Belkouch S., Hamzaoui A., Zennayi Y. Single core SIMD parallelization of GMM background subtraction algorithm for vehicles detection. *2018 IEEE 5th International Congress on Information Science and Technology (CiSt), October 21–27, 2018, Marrakech, Morocco*. P. 308–312. https://doi.org/10.1109/CIST.2018.8596385
16. Cui H., Li F., Fang X. Effective parallelism for equation and Jacobian evaluation in large-scale

power flow calculation. *IEEE Transactions on Power Systems*. 2021. Vol. 36. No. 5. P. 4872–4875. https://doi.org/10.1109/TPWRS.2021.3073591

17. Liu Y., Wang J. Simulation of finite difference time domain (FDTD) with GPU and SIMD. *2023 International Applied Computational Electromagnetics Society Symposium (ACES), March 26–30, 2023, Monterey/Seaside, CA, USA*. P. 1–2. https://doi.org/10.23919/ACES57841.2023.10114776
18. Wang Z., Li F., Mei G. OpenMP parallel finite-discrete element method for modeling excavation support with rockbolt and grouting. *Rock Mechanics and Rock Engineering*. 2024. Vol. 57. P. 3635–3657. https://doi.org/10.1007/s00603-023-03746-w
19. Li Y., Xu J., Liu Y., Zhong W., Wang F. MPI/OpenMP-based parallel solver for imprint forming simulation. *Computer Modeling in Engineering & Sciences*. 2024. Vol. 140. No. 1. P. 461–483 https://doi.org/10.32604/cmes.2024.046467
20. Sivanandan V., Kumar V., Meher S. Designing a parallel algorithm for heat conduction using MPI, OpenMP and CUDA. *2015 National Conference on Parallel Computing Technologies (PARCOMPTECH), February 19–20, 2015, Bengaluru, India*. P. 1–7. https://doi.org/10.1109/PARCOMPTECH.2015.7084516
21. Fathoni M. F., Sridadi B. Multicore computation of tactical integration system in the Maritime Patrol Aircraft using Intel Threading Building Block. *2014 International Conference on Advanced Computer Science and Information System (ICACSIS), October 18–19, 2014, Jakarta, Indonesia*. P. 1–6. https://doi.org/10.1109/ICACSIS.2014.7065821
22. Sah S. K., Naik D. Parallelizing doolittle algorithm using TBB. *2014 International Conference on Parallel, Distributed and Grid Computing (PDGC), December 11–13, 2014, Solan, India*. P. 13–15. https://doi.org/10.1109/PDGC.2014.7030707
23. Ding L., Li H. Parallel processing for accelerated Mean Shift algorithm based on TBB. *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), July 10–15, 2016, Beijing, China*. P. 6348–6351. https://doi.org/10.1109/IGARSS.2016.7730659
24. Alsuwaiyel M. H. *Parallel algorithms*. New Jersey: World Scientific, 2022. 400 p. https://doi.org/10.1142/12744

Oleg Zhulkovskyi, Inna Zhulkovska, Petro Kurliak, Oleksandr Sadovoi, Yuliia Ulianovska, Hlib Vokhmianin

ASINCHRONINIO PROGRAMAVIMO NAUDOJIMAS SIEKIANT PAGERINTI ENERGETIKOS SISTEMOSE NAUDOJAMO KOMPIUTERINIO MODELIAVIMO EFEKTYVUMĄ

Santrauka

Dėl vis sudėtingėjančių šiuolaikinių energetikos sistemų, auga poreikis prognozuoti energijos suvartojimą ir gamybą, optimizuoti procesus ir kurti naujas technologijas energetikos sektoriuje, analizuoti energetikos sistemų plėtros scenarijus ir kurti jų plėtros strategiją. Poreikis didinti kompiuterinio modeliavimo efektyvumą energetikos sektoriuje veiksmingai sprendžiamas naudojant šiuolaikines kompiuterių architektūras ir pažangias programinės įrangos priemones, kurios pagreitina skaičiavimams imlių užduočių vykdymą. Šiame straipsnyje daugiausia dėmesio skiriama skaičiavimams imlių algoritmų našumui didinti naudojant Thomaso algoritmą, taikant šiuolaikinius asinchroninio programavimo metodus. Straipsnyje pateikiami klasikiniai ir vystomi asinchroniniai iššlavimo metodo algoritmai, įvertinant tiesinių lygčių sistemų sprendimo laiką ir iššlavimo metodo iki 5×10^7 eilės efektyvumą. Programos kodas sukurtas naudojant *Microsoft Visual Studio C++* ir standartinį asinchroninio programavimo šabloną. Skaitiniai eksperimentai parodė, kad asinchroninio algoritmo įgyvendinimo greitį galima padidinti 1,87–2,91 karto. Darbo rezultatai atitinka mokslinės literatūros duomenis ir autorių anksčiau gautus rezultatus panašiuose tyrimuose naudojant alternatyvią lygiagrečiojo programavimo programinę įrangą. Apibendrinant galima teigti, kad darbo rezultatai lemia tolesnio skaičiavimo uždavinių lygiagretaus realizavimo metodų ir technologijų vystymo galimybes naudojant triįstrižainės matricos algoritmą. Šie metodai gali būti taikomi kuriant įvairius energetinių procesų ir sistemų kompiuterinius modelius, pagrįstus tiesinių lygčių sistemų su triįstrižainėmis matricomis sprendimu kompiuteriuose, kurių architektūra yra daugiaprocesorinė arba daugiabranduolinė.

Raktažodžiai: kompiuterinis modelis, skaičiavimo pagreitis, asinchroninis programavimas, skaičiavimo algoritmas, tiesinių lygčių sistemų sprendimas