Artur PALA[1], Marek MACHACZEK[1]

(1) Opole University of Technology, Institute of Computer Science

doi:10.15199/48.2019.03.31

Parallel computing of two-parameter bifurcation diagrams of an electric arc model with chaotic dynamics using Nvidia CUDA and OpenMP technologies

Abstract. *This paper presents parallel and massively parallel calculations of two-parameter bifurcation diagrams of an electric arc model. A simple dynamical model of electric arc is used. Such a model can show complex two-parameter bifurcations with periodic and chaotic responses. Two different parallel computing technologies were used to implement the calculations. Parallel computations are implemented using the OpenMP library and CPU processors. Massively parallel computations are implemented using the Nvidia CUDA technology and GPU processors.*

Streszczenie. *W artykule przedstawiono równoległe i masowo równoległe obliczenia dwuparametrycznych diagramów bifurkacyjnych dla modelu łuku elektrycznego. Do analizy wykorzystano dynamiczny model łuku elektrycznego z okresowymi i chaotycznymi odpowiedziami. Do realizacji obliczeń wykorzystano dwie różne technologie. Obliczenia równoległe zaimplementowano przy użyciu biblioteki OpenMP i procesorów CPU. Obliczenia masowo równoległe zostały zaimplementowane przy użyciu technologii Nvidia CUDA i procesorów GPU. (Równoległe obliczanie dwuparametrycznych diagramów bifurkacyjnych dla modelu łuku elektrycznego z wykorzystaniem technologii Nvidia CUDA i OpenMP).* ***Równoległe i masowo równoległe obliczenia dwuparametrycznych diagramów bifurkacyjnych dla modelu łuku elektrycznego***

Keywords: Electric Arcs, Bifurcations, OpenMP, Nvidia CUDA, Multi GPU, Dynamical Models, Parallel Computing.
Słowa kluczowe: Łuk Elektryczny, Bifurkacje, OpenMP, Nvidia CUDA, Wiele GPU, Modele Dynamiczne, Obliczenia Równoległe

Introduction

This paper presents two-parameter bifurcation diagrams for a simple electric arc model. The bifurcation diagrams are understood as changes in the oscillatory solutions when two parameters of the analyzed electric arc model vary simultaneously. The goal is to obtain high resolution color diagrams showing various oscillatory and chaotic responses. However, obtaining such two-parameter diagrams requires solving the underlying system of ordinary differential equations (ODEs) many hundreds of thousands (or millions) of times. In each solution, in addition to solving the system of ODEs, it is necessary to find the local maxima, identify period of oscillation, or determine that the solution is chaotic or unstable. The final graphical representation of the identified solutions is the bifurcation diagram. Such a process of solving ODEs, identifying the type of response (i.e. period-*n*, chaotic or unstable) would be very time and memory consuming for large size of the two-parameter matrix values and, for those reasons, its sequential execution would not be obtainable in practice. The answer to this problem is the use of parallel programming.

Parallel programming is an increasingly popular way of solving complicated problems. Parallel designed and implemented program code is characterized by a better use of resources and shorter computation time. Parallelization usually aims to shorten computation time by using more computing units, e.g. multiple cores and graphical processing units (GPUs). Many calculation problems have a sequential nature, that is, instructions are to be executed one by one and the next instruction depends on the previous one. Such difficulties are a serious limitation for the possibility of parallelizing the code. The code parallelization process can be further complicated by the specific hardware architecture, for example the GPU. Another limiting factor, when using a GPU, may be the amount of available memory.

In [1-3] typical models of electric arcs are presented. The models are based on various arc voltage-current (V-I) characteristics to be used to estimate energy in the case of an arc fault [1] and one-parameter bifurcation diagrams in [2,3]. Various V-I characteristics results in different nonlinear systems of ODEs serving as models of electric arcs. As shown in the literature, a small change in the value of a parameter (for example, resistor, inductor or capacitor) may lead to a significant change in the nature of the ODE system's response [4]. Combining changes of two parameters may result in very complicated two-parameter bifurcation diagrams [5,6]. This is in fact the case for the electric arc circuit considered in this paper.

In this paper, we present bifurcation diagrams calculated for typical electric arc circuits shown in Fig. 1. Our diagrams are calculated for any pair of two parameters (from the three ones, R, L and C present in the arc circuits) changing simultaneously.

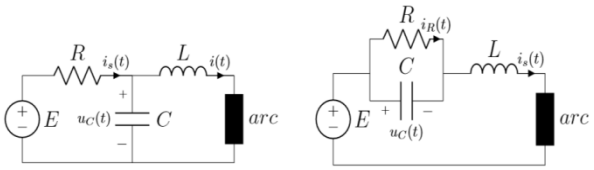

Fig. 1. Two typical electric arc circuits.

The following is a system of ODEs for the two electric arc circuits shown in Fig. 1. The system on the left side of (1) describes the circuit A. On the right side of (1) a dimensionless form of the ODE system is shown [2].

$$\frac{di}{d\tau}=\frac{1}{L}\left(u-\frac{U(i_\theta)}{i_\theta}i\right) \qquad \frac{dx}{dt}=\frac{1}{L}\left(y-xz^m\right)$$

$$\frac{du}{d\tau}=\frac{1}{RC}\left(E-u-Ri\right) \rightarrow \frac{dy}{dt}=\frac{1}{RC}\left(R+1-y-Rx\right) \tag{1}$$

$$\frac{di_\theta^2}{d\tau}=\frac{1}{\theta}\left(i^2-i_\theta^2\right) \qquad \frac{dz}{dt}=x^2-z$$

where: $x=\frac{i}{I_0}, y=\frac{u_c}{U_0}, z=i_\theta^2/I_0^2$, i_θ, is the arc current, U_0, I_0 are two constants from the static arc voltage-current

V-I characteristic $U(i) = U_0(\frac{i}{Io})^n$ and the constant $n<0$. Additionally, $\theta=\tau/t$ is a time constant, R, L, C are the: resistance, inductance and capacitance, and the m constant results from the fact that $-1 < m = \frac{n-1}{2} < 0$. Typically, $m=-2/3$, as it follows from $U(i_\theta) = i_\theta^n$ for $n=-1/3$. One can also consider the system of equations on the right side of (1), as the system on the left side with the unit values $\boldsymbol{U_0 = I_0 = \theta = 1}$ and $\boldsymbol{E = RI_0 + U_0}$. By making an appropriate change of variables, the system (1) also describes the circuit B in Fig. 1 [2].

Calculation of bifurcation diagrams

The bifurcation diagrams presented in this paper show the changes in the nature of the response of (1) when the pairs of parameters *(R,L), (R,C) or (L,C)* vary in certain intervals, as follows: $R_{min} \le R \le R_{max}$, $L_{min} \le L \le L_{max}$, $C_{min} \le C \le C_{max}$. When solving the system (1), two different time horizons were used, namely: $0 \le t \le 500$ and $0 \le t \le 2000$. The initial conditions for all presented cases were the same and equal $x(0)=0.5$, $y(0)=4.0$, $z(0)=1.0$. The integration step in the Runge-Kutta method of order IV (R-K IV) was also the same in all cases and equal $h = 0.005$.

The search for the local maxima of the ODE oscillatory solution of (1) and period identification were done in the interval $200 \le t \le 500$ for the overall time horizon $0 \le t \le 500$ and in $1700 \le t \le 2000$ for the time horizon $0 \le t \le 2000$. The period of oscillations was identified in the range from *1* to *16* maximum values. That is, *period-n* oscillation means that there are n distinguished maximum values in one period, with $n = 1, ..., 16$. Any oscillatory response with the number of maximum values greater than *16* in one period was classified as *period-16* type of response. If a chaotic response was identified, then such a response was also classified as *period-16* type of response. Limiting of the identification of the type of response as *period-1* to *period-16* does not seem to be a significant restriction, and one can easily consider an extension of the identification process to *period-32* or even *period-48* type of response. Each *period-n* type of response has been assigned a specific color on a bifurcation diagram. The color in the diagram was the lightest when the largest number of maxima was found in one period (that is if $n=16$) and the darkest when n was equal to *1*. The absolute error tolerance value for comparing subsequent maxima values in the period identification process was $tol = 0.0005$. This value was determined empirically based on the observation of noise levels appearing in the obtained bifurcation diagrams.

Implementation of parallel calculations

The R-K IV method was used to solve system (1). This is the well-known and widely used method from a wide class of the ODE numerical algorithms. The method provides a good accuracy of computations with reasonably large values of integration steps. The algorithm of this method is expressed by the following formula [7]

$$x_{i+1} = x_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2}$$

where: i – iteration counter, $i = 1,2,...N$; N – assumed number of integration steps;
$k_1 = h \cdot f(x_i, t_i)$ $k_2 = h \cdot f(x_i + k_1/2,\ t_i + h/2)$
$k_3 = h \cdot f(x_i + k_2/2,\ t_i + h/2)$ $k_4 = h \cdot f(x_i + k_3,\ t_i + h)$
$f(x_i, t_i)$ – function of the right hand side of ODE;
h – value of the integration step; t – time variable

Parallel computations were implemented on graphics cards using Nvidia CUDA technology and on standard CPU processors using the OpenMP library [8-12].

Nvidia CUDA technology [8-10] is a massively parallel computing platform based on the use of GPUs. Thanks to the use of light thread technology, it is possible to effectively use a very large number of execution cores. Calculations on the CUDA platform are based on a heterogeneous processing model. This means that the CUDA device is installed in the host system as a separate computing subsystem having its separate resources. Performing calculations on the CUDA device always requires controlling done on the host side and copying data between the host and the CUDA device. The specific nature of the calculation process within the Nvidia CUDA technology makes it suitable for the implementation of very specific algorithms characterized by a high degree of parallelization, high computational complexity and operation on relatively large data sets.

OpenMP [11,12] is an API for writing parallel programs on multi-core processors with shared memory. This library can be used, among others, for C and C++ programming on various platforms (e.g. Windows, Linux). By using special instructions programmer decides which parts of the code will be executed in parallel. The code execution is parallelized by creating new threads (fork) that perform the tasks assigned to them. After completing their individual tasks, the threads merge (join) and the program is executed sequentially by the main thread. Master thread can be divided again at another place and the program will be executed in parallel. Threads can work with different loads, and their number can be assigned arbitrarily. Number of threads can be larger than the number of physical processor cores. The programmer's task is to ensure that the threads access data correctly (using the shared memory). Otherwise the results may be incorrect, for example, because of the data race condition. OpenMP is portable and scalable. Thanks to those features, programs using this standard can be run on traditional desktop or cluster computers.

Special software has been written for the implementation of massively parallel calculations in Nvidia CUDA technology. The software measures the exact time of running the algorithms. This is a dedicated software for the measurement platform, which is described in detail in the following section of this paper. Analysis of early version of the algorithm written for CUDA showed that the number of available Tesla K80 GPU work registers is insufficient to fully load the GPU with computational tasks. To enable the full use of all available CUDA cores, it was decided to design the algorithm in a different way. The algorithm has been divided into two separate CUDA kernels, launched one after the other. This approach has significantly reduced the consumption of the GPU work registers.

The main task of the first kernel is to solve the ODE system (1) using the R-K IV method given by (2). The first kernel also performs a search for the local maxima of the variable z in (1). The choice of variable z is not significant and any of the remaining two variables x or y can also be used. Those maxima are identified based on the value of the k_1 coefficient in (2). The maximum is found if

$$k_{1,i} > 0 \text{ and } k_{1,i+1} < 0.$$

The found maxima are stored in the global memory of the CUDA device. They remain there until the second kernel is launched. The task of the second kernel is to identify the period based on the maxima previously stored.

The software allows generation of two-parameter bifurcation diagrams with theoretically unlimited resolution and with any sizes (proportions of horizontal to vertical grid points). However, the values defining the dimensions of the computational task and, consequently, the vertical and horizontal resolution values are not accidental. The total dimensions of the computational task are calculated from the following formula

$$\dim A = bpgA \cdot tpbA \cdot partsA$$
$$\dim B = bpgB \cdot tpbB \cdot partsB \quad (3)$$

where: $dimA$ and $dimB$ define the total dimensions of the computational task, $bpgA$ and $bpgB$ are the numbers of CUDA calculation blocks inside the grid, $tpbA$ and $tpbB$ are the numbers of threads in block, $partsA$ and $partsB$ denote the dimensions of parts of the whole task.

Although the grid point sizes of the task can be arbitrary, for aesthetic reasons it was assumed that $dimA = dimB$ for all cases (see Table 1). In order to compare the computational times of the parallel algorithms on the CPU and the mass-parallel algorithm on CUDA devices, the dimensions of the task for the CPU were adjusted to be the same as the dimensions adopted for CUDA.

Parallelization of calculations on CPUs using OpenMP has been achieved for each pair of varying parameters. Each thread solves its own system of equations for the given parameters, using the R-K IV method (2). In the next step the maxima and period are determined. Another method, not tested in this paper, that could also work well for smaller problems (with dimension less than the number of threads), is the parallelization of the method at the level of solving the system of equations or determining the maximum points. However, the smallest dimension of the problem was 384x384, which significantly exceeds the available number of physical cores. This does not require parallelization at such a low level.

Description of the measuring platform

The measuring platform for all algorithms used in this paper was a computational cluster installed at the Institute of Computer Science, Opole University of Technology.

The massively parallel algorithm implemented by using Nvidia CUDA technology was run on a computer with the following hardware configuration: two fourteen-core processors Intel(R) Xeon(R) CPU E5-2683 v3 2.0 GHz, 3 x computing devices CUDA Tesla K80. Each device consists of 2 GPUs. Each device had 4992 CUDA cores and 24 GB of GDDR5 memory. The server also consists of 128 GB of RAM and an SSD with a 1000MB/s write speed and 2000MB/s read speed. It is operating on the Windows Server 2012 R2 64 bit operating system.

Calculations based on the OpenMP library were carried out on the second server which consists of: two ten-core processors Intel Xeon (2.3 GHz), 128 GB of RAM and SSD (1000MB/s write speed and 2000MB/s read speed). It operates under the Linux (Xubuntu 14.04) operating system.

Selected measurement results

All time values obtained from our measurements are given in seconds. A summary of the time values of all the measurements is shown in Table 1. Three different parameter matrix sizes have been used for computations: 384x384, 768x768 and 1152x1152.

TABLE 1: The time measurements

Case	Dimension of the computational task	N	Computational times of the algorithms (in sec)	
			OpenMP / CPUs [s]	CUDA / GPUs [s]
1	384 x 384	100 000	377.58	53.09
2	768 x 768	100 000	1470.04	226.64
3	1152 x 1152	100 000	3279.87	386.97
4	384 x 384	400 000	1519.08	195.78
5	768 x 768	400 000	5871.89	904.74
6	1152 x 1152	400 000	13159.54	1516.07

For each of the matrix sizes, calculations were done for all three combinations of parameters (for each pair of variable parameters *RL, RC* and *LC*) with two values of the integration steps. First, the number of integration steps was $N = 100{,}000$, which corresponds to the time interval $0 \le t \le 500$ seconds. Then, the calculations were repeated for the number of integration steps $N = 400{,}000$, which corresponds to the time interval $0 \le t \le 2000$ seconds. The number of all calculation cases was 18.

Factors having a significant impact on the algorithms computational times were the dimension of the problem and the number of integration steps. Therefore, the number of cases in Table 1 was limited to 6. For all cases, the CUDA based massively parallel algorithm was faster than a competitive CPU based OpenMP algorithm.

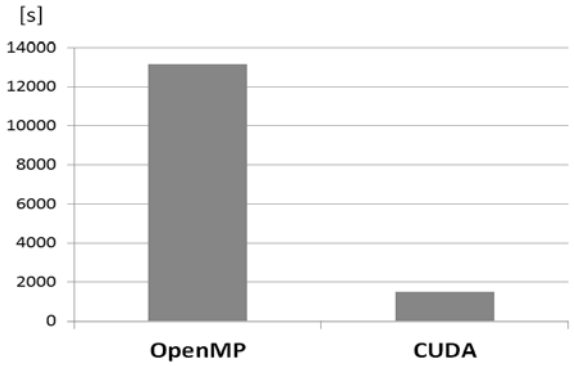

Fig. 2. Comparison of CUDA and OpenMP times for case 6.

The biggest advantage of the CUDA algorithm over the OpenMP algorithm was observed for case 6 from Table 1, as shown in Fig. 2, where the CUDA algorithm was about 9 times faster.

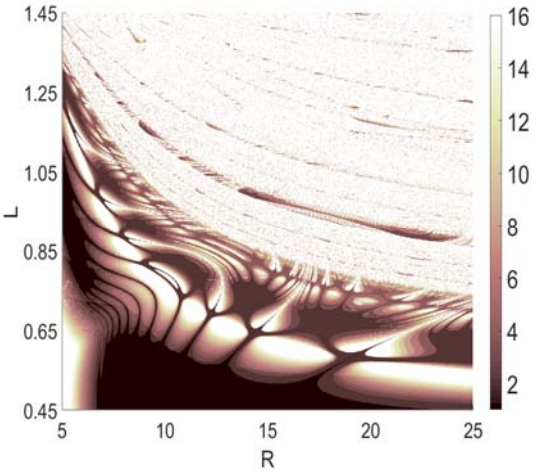

Fig. 3. Diagram for variables RL and constant C=3.14 (N=100,000)

Two-parameter bifurcation diagrams

Two bifurcation diagrams were made for each pair of parameters: RL, RC and LC. All presented diagrams have a

resolution of 1152 x 1152. The diagrams for the third case from Table 1 (Figs. 3,5,7 below) present an unsteady solution of the system (1) for the instant $t=500$ seconds. The number of integration steps $N=100,000$ was not large enough for the system (1) to achieve steady state. The application of a larger number of integration steps ($N = 400,000$) in the sixth case from Table 1 (Figs. 4,6,8 below) allowed to obtain steady state solution, as these three figures show the solution for the instant $t=2000$ seconds. Any further increasing of the number N do not bring any change of the solution.

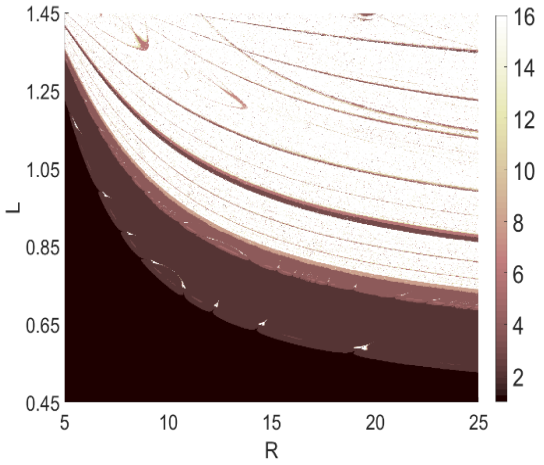

Fig. 4. Diagram for variables RL and constant C=3.14 (N=400,000)

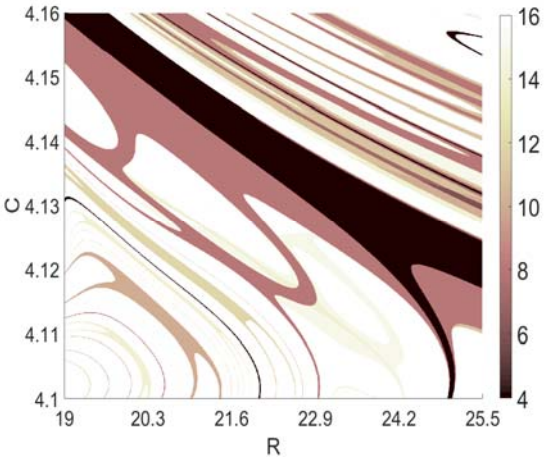

Fig. 5. Diagram for variables RC and constant L= 0.2 (N=100,000)

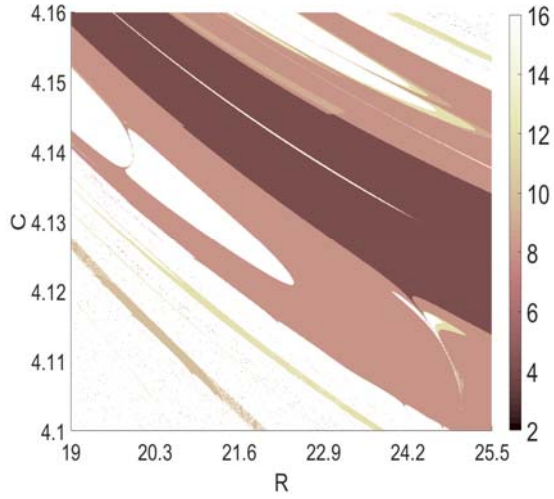

Fig. 6. Diagram for variables RC and constant L= 0.2 (N=400,000)

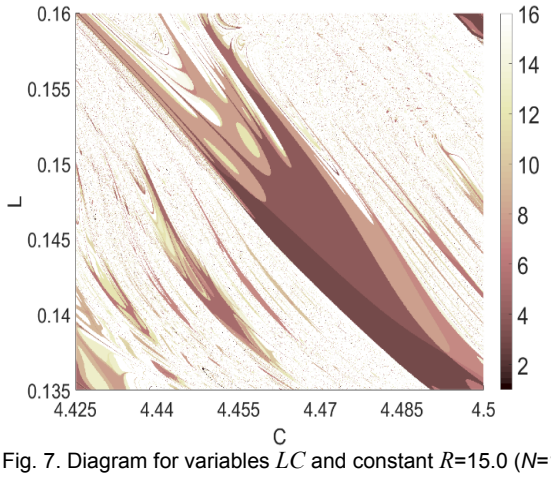

Fig. 7. Diagram for variables LC and constant R=15.0 (N=100,000)

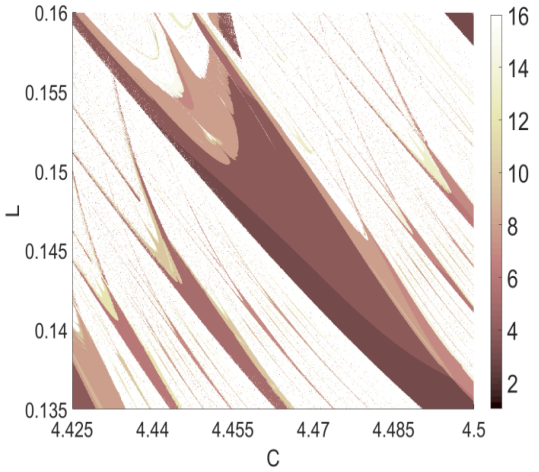

Fig. 8. Diagram for variables LC and constant R=15.0 (N=400,000)

Conclusions

Obtaining two-dimensional bifurcation diagrams is a very costly computation task. The use of a sequential algorithm is practically impossible, so it is necessary to implement calculations in parallel.

The presented results of the computational times of algorithms analyzed in this article indicate the high potential of Nvidia CUDA technology in this type of computations. The fact that each of the points of the bifurcation diagram is calculated independently of the others is particularly important. Thanks to this, there is no need for synchronization or communication between threads working on the GPU. In addition, due to a proper organization of calculations, the solved task did not require large number of massive and time expensive transfers between the host and the CUDA device.

The use of the OpenMP library shows acceptable execution times. Independent calculation of each solution (each point on the bifurcation diagram) and the lack of the need to synchronize threads made it easy to implement a parallel program.

Parallel algorithm based on CUDA technology in each tested case was faster than parallel algorithms with the OpenMP library. For the case with largest dimension of the problem, it was about 9 times faster. However, an important feature of the OpenMP algorithm is its simplicity and ease of implementation. In contrast of this, the algorithm based on CUDA was much more complicated. This is due to the limited number of resources on the graphics card and their appropriate use. Both parallel versions of the algorithm allowed to obtain results for the chosen dimensions of the tested algorithm in acceptable times.

It should be noted that the parallel computing approach, as well as the parallel computing technologies used in this paper, may also be used in other chaos research.

The issue of creating dedicated expert systems is an important element in most areas of electrical engineering, computer science and electronics. An example of such system is laid out in [13]. The research presented in this paper can be further developed until it will be possible to create expert system for designing and simulating chaotic circuits.

***Authors**: mgr inż. Artur Pala, Politechnika Opolska, Instytut Informatyki, Katedra Systemów Równoległych i Sztucznej Inteligencji, ul. Prószkowska 76, 45-758 Opole, E-mail: a.pala@po.opole.pl; mgr inż. Marek Machaczek, Politechnika Opolska, Instytut Informatyki, Katedra Systemów Równoległych i Sztucznej Inteligencji, ul. Prószkowska 76, 45-758 Opole, E-mail: m.machaczek@po.opole.pl.*

REFERENCES

[1] R. F. Ammerman, T. Gammon, P. K. Sen and J. P. Nelson, DC-Arc models and incident energy calculations, IEEE Trans. Industry Appls., vol. 46, no. 5, pp. 1810–1819, Sept./Oct. 2010.

[2] V. N. Sydorets and I. V. Pentegov, Deterministic Chaos in Nonlinear Circuits with Arcs, Kiev, Ukraine: Svarka, 2013 (in Russian).

[3] I. V. Pentegov and V. N. Sydorets, Comparative analysis of models of dynamic welding arc, The Paton Welding Journal, vol. 12, pp. 45–48, 2015.

[4] W. Marszalek and H. Podhaisky, 2D bifurcations and Newtonian properties of memristive Chua's circuits, EPL (Europhysics Letters), vol.113, no. 1, 10005, 2016.

[5] W. Marszalek and J. Sadecki, 2D bifurcations and chaos in nonlinear circuits: a parallel computational approach, 15th Int. Conf. Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Prague (Czech Republic), 2-5 July 2018.

[6] W. Marszalek, Autonomous implicit models of pinched hystereses with application to memristors, Przegląd Elektrotechniczny, vol. 94, no. 4, pp. 13-16, 2018.

[7] S. C. Hapra and R. P. Canale, Numerical Methods for Engineers, McGraw-Hill, 2011.

[8] NVIDIA CORPORATION: Nvidia CUDA C Programming Guide, Nvidia Corporation 2015

[9] NVIDIA CORPORATION: An Introduction to GPU Computing and CUDA Architecture, Nvidia Corporation 2011

[10] D. B. Kirk, Wen-mei W. Hwu Programming Massively Parallel Processors: A Hands-on Approach, Morgan Kaufmann, 2010.

[11] OpenMP, https://www.openmp.org/specifications/, 2018.

[12] OpenMP, https://computing.llnl.gov/tutorials/openMP/, 2018.

[13] A. Handkiewicz, P. Katarzyński, S. Szczęsny, M. Naumowicz, M. Melosik, P. Śniatała, M. Kropidłowski, Design automation of a lossless multiport network and its application to image filtering, Expert Systems with Applications, vol. 41, no. 5, 2014.