Computing of 3D Bifurcation Diagrams With Nvidia CUDA Technology

ARTUR PALA AND MAREK MACHACZEK
Institute of Computer Science, Opole University of Technology, 45-758 Opole, Poland
Corresponding author: Artur Pala (a.pala@po.edu.pl)

ABSTRACT This paper presents three-dimensional bifurcation diagrams in the form of a point cloud. The diagrams were created by means of parallel calculations using the electric arc model. A simple dynamic DC electric model that can show complex bifurcations with periodic and chaotic responses is presented. The Nvidia CUDA massively parallel architecture was used to perform the calculations. The paper shows cross-sections of the bifurcation cloud obtained as a result of calculations. It also presents an interactive tool specially developed to visualize the cloud. The reader can access the tool by using the Web address provided in this paper. This tool enables to freely specify the plane and axis of the cut in order to obtain the desired cross-section of the cloud — it is possible to obtain exactly such diagrams as those presented in the paper, as well as many others.

INDEX TERMS Electric arc, 3D bifurcation diagrams, cloud of points, Nvidia CUDA, ODE, parallel computing, 3D visualization tool.

I. INTRODUCTION
This paper uses three-dimensional approach to bifurcation diagrams, while the bifurcation diagrams usually presented in the literature take a single-parameter or much less often two-parameter form [1]–[6]. The use of a three-dimensional approach to bifurcation involves the presentation of a point cloud in space. In the considered case, bifurcation diagrams are understood as changes in the oscillatory reactions of variable parameters of a given model of an electric arc. In the case of a three-dimensional approach to bifurcation, three parameters of such a system change simultaneously. It has been assumed in the paper that these parameters are slowly changing and their changes occur in the established intervals. Such an approach aims at obtaining high resolution graphs. 1000 × 1000 × 1000 points in space has been assumed as the target resolution of the diagrams. Thus, generating such diagrams requires multiple solving of the system of differential equations (ODE) (of the order of $10^9$). In each iteration, the first stage of the algorithm is solving the system of ODEs. The next stage requires the analysis of the solution to find local maxima and to identify the period on their basis. Once the identification is complete, graphic interpretation is possible in the form of a point in the three-dimensional space of arguments. Each point is assigned the appropriate colour from the previously adopted colour palette. This creates a cloud of different-coloured points in space that depicts bifurcations.

Parallel and massively parallel computing technologies such as Nvidia CUDA have recently gained a lot of popularity due to the increasing complexity of computing problems. The application areas include medicine [7], [8], physics [9], chemistry [10] and many other fields. However, the development of appropriate parallel algorithms to exploit the potential of this type of computing technology remains an important problem to be solved. Parallelization of the program is usually aimed at shortening the calculation time, by using more computing units, e.g. many cores/processors, GPUs. One should note, however, that in the case of complex existing programs, effective implementation on the GPU is not always possible or cost-effective [11]. Many computational problems have a sequential nature that forces the use of a sequential approach. For this reason, statements in the program must be executed one by one (the next statement depends on the previous one). However, the sequential nature of the problem is not the only obstacle to overcome when developing parallel algorithms. Very often the problem turns out to be a highly complicated hardware architecture hindering the development of a parallel algorithm. Nvidia CUDA is an example of such a specific computational technology with
high theoretical computational potential but difficult to use in practice. Another important problem with GPU and CUDA technology may be the limited amount of memory available on the graphics card. The small amount of graphics memory leads to frequent data exchange between the host and the CUDA computing device and significantly complicates the implementation of calculations. A further consequence of the presented difficulties is the decrease in the efficiency of calculations.

The bifurcation diagrams presented in this paper were based on typical electric arc models attached as Appendix A, and described in more detail in the literature [12]. The presented models are based on different arc voltage (V-I) characteristics and are used to estimate the energy in the event of an arc fault. These types of characteristics can be used to calculate bifurcation diagrams. Examples of calculating such diagrams based on a single selected parameter whose value changes slowly can be found in literature [13] and [14]. Any change in the value of such a parameter can lead to a change in the nature of the response of the system of ODE. As shown in the literature, a small change in the parameter value (for example, a resistor, inductor or capacitor) can lead to a significant change in the nature of the response of the system of ODE [15]. Combining changes to two parameters can result in very complicated two-parameter bifurcation diagrams [16], [17]. This is the case with the electric arc circuit considered in this paper.

One should note that the paper shows bifurcation diagrams which were calculated not for one or two, but for three parameters changing simultaneously.

Despite the existence of many other chaotic and oscillatory systems, it was decided to first perform tests on simpler arc circuits. They are well described in the literature, and due to their simpler structure, the implementation of an appropriate algorithm for the Nvidia CUDA technology seemed achievable and could bring good results.

II. THREE-DIMENSIONAL BIFURCATION DIAGRAMS

This paper shows three-dimensional bifurcation diagrams for a system of equations describing electric arc circuits (appendix A). We can observe changes in the nature of the system response depending on changes in all three parameters (R, L, C) simultaneously. When solving the system of equations, the time horizon was \( 0 \leq t \leq 2000 \). The assumed initial conditions for all presented cases were the same: \( x(0) = 0.5y(0) = 4.0z(0) = 1.0 \). The integration step for all presented cases was \( h = 0.005 \). The identification of the period was carried out on the basis of searching for local maxima of the solution for the \( 1700 \leq t \leq 2000 \) range. The period identification included local maxima from 1 to 16. If there were more than 16 maxima, 16 were accepted. A palette of 16 colours has been created and each of them has been assigned a different value from 1 to 16 (Fig. 3.). The more maxima were found for a given solution, the brighter colour was assigned to the point in the three-dimensional space of R, L, C arguments. White is assumed for the value of 16, while black is assumed for the value of 1. The error tolerance value with which individual maxima were compared in the period identification process was \( tol = 0.0005 \). This value was determined empirically on the basis of observation of noise behaviour occurring on the obtained bifurcation diagrams.

III. IMPLEMENTATION OF CALCULATIONS

To solve the example system of equations presented in Appendix A, the Runge-Kutta fourth-order method was used. It provides relatively good solution accuracy with a relatively high value of the integration step. The algorithm of the method is expressed by the formula (1) [18]–[20].

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

gdzie:

\[
k_1 = h \cdot f(x_i, t_i),
\]

\[
k_2 = h \cdot f(x_i + k_1/2, t_i + h/2),
\]

\[
k_3 = h \cdot f(x_i + k_2/2, t_i + h/2),
\]

\[
k_4 = h \cdot f(x_i + k_3, t_i + h).
\]

The calculations were carried out based on Nvidia CUDA massively parallel processing technology [21]–[25]. This technology is a massively parallel computing platform based on the use of graphics processors (GPUs). Thanks to the use of lightweight thread technology, it is possible to effectively utilize a very large number of execution cores.

The 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 with its separate resources. Performing calculations on a CUDA device always requires control from the host, and copying data between the host and the CUDA device. The data processing model within Nvidia CUDA technology has some limitations. The effective use of CUDA technology requires the implementation of very specific algorithms characterized by a high degree of parallelism, high computational complexity, and independence of individual threads from each other.

In order to perform the calculations, special software dedicated to the measuring platform was written, described in detail in the section titled: “Description of the measuring platform.” The main solver performing calculations was designed in the form of two separate CUDA kernels (kernel is a function performed on the CUDA device). The task of the first kernel is to solve the system of differential equations using the Runge-Kutta fourth-order method (1).

The first kernel also carries out the search for local maxima of \( Z \) variable functions in the equation system under consideration. This is done based on the \( K1 \) coefficients in formula 1. The maximum is found when:

\[
K1_i > 0 and K1_{i+1} < 0
\]

The next maxima found are stored in the CUDA memory. The second CUDA kernel performs period identification based on the previously stored maxima.
The computing software written enables to generate a point cloud in unlimited resolution. Only the computational performance of the equipment used can be a practical limitation here. The dimensions of the task, although not limited to the maximum value, are also not completely arbitrary. Two freely selected from the three dimensions of the calculation task are derived from formula 2:

\[
\begin{align*}
    \text{dimA} &= bpgA \times tpbA \times \text{partsA} \\
    \text{dimB} &= bpgB \times tpbB \times \text{partsB} \\
    \text{dimC} &= \text{fv} > 0
\end{align*}
\]

(2)

where:
- \(\text{dimA}\) and \(\text{dimB}\) – overall dimensions of the calculation task
- \(bpgA\) and \(bpgB\) – the number of CUDA calculation blocks
- \(tpbA\) and \(tpbB\) – the number of threads per block
- \(\text{partsA}\) and \(\text{partsB}\) – dimensions of fragments of the entire task

The third dimension (\(\text{dimC}\)) can take any integer value (\(\text{fv}\)) greater than zero.

IV. DESCRIPTION OF THE MEASURING PLATFORM

The computing cluster installed at the Institute of Computer Science of the Opole University of Technology has been used as the measurement platform for all developed algorithms shown in the paper. The hardware configuration of a computer running a massively parallel algorithm using Nvidia CUDA technology is as follows: two 14-core Intel(R) Xeon(R) CPU E5-2683 v3 2.0 GHz, 3 x CUDA Tesla K80 computing devices (each device has 2 GPUs). Each card had 4992 CUDA cores and 24 GB of GDDR5 memory. The server also consisted of 128 GB of RAM and an SSD with a write speed of 1000MB/s and a read speed of 2000MB/s. The installed operating system is Windows Server 20112 R2 64bit.

V. MEASUREMENT RESULTS

Measurements of algorithm operating time for 4 cases were made. All presented time values obtained from the measurements are given in hours.

Table 1 shows the exact times of the algorithm operation.

| CESE | PROBLEM DIMENSION        | CUDA / GPU’s [HOURS.] |
|------|--------------------------|-----------------------|
| 1    | 96 x 96 x 96             | 3.821                 |
| 2    | 384 x 384 x 384          | 22.657                |
| 3    | 768 x 768 x 768          | 199.037               |
| 4    | 1152 x 1152 x 1152       | 508.336               |

A logarithmic scale was used to increase the readability of the Y axis graph.

VI. VISUALIZATION OF DATA FROM CALCULATIONS

In order to present a point cloud showing bifurcations in the three-dimensional space of RLC parameters (Fig. 2.), special software has been written allowing to display sections of such a point cloud in real time.
The software was written using the Three.js library [26]–[30] for generating and displaying graphics in a web browser.

The application works on the client side, while the data from which the graphs are generated are located on the server.

In order to minimize the requirements on the user side, especially memory requirements, the necessary data is retrieved only when subsequent graphs are generated. The program enables to view three-dimensional graphs in the form of two-dimensional sections (graphs) for one pair of RL, RC or LC parameters along the chosen axis (R, L, C).

The application offers the capability to change the axis along which the graphs are viewed at any time (see Appendix B for an illustrated description of the operation of the interface).

The same number of two-dimensional graphs are generated for each axis. The program has been designed in such a way that it is possible to go directly to the selected cross-section with the implemented slider.

The tool discussed in this paper presenting the bifurcation cloud obtained by calculations can be found at the following Web address: https://m.machaczek.po.opole.pl/diagrams.html

Using the developed tool, six examples (Fig. 4, 5, 6, 7, 8, 9) of sections of the bifurcation cloud were made, determining the cutting plane in relation to each of the R, L and C axes, respectively. All diagrams (cloud sections) shown in this paper have a resolution of 1152 × 1152 and are the result of case 4 presented in Table 1. Note that the diagram shown in this paper in Figure 8 is a wider picture of the diagram presented in Figure 2(b) in the paper[16], which
FIGURE 6. Diagram for the RC cutting plane. \((R = [5.0, 25.0], \ L = 0.16, \ C = [3.14, 4.5])\).

FIGURE 7. Diagram for the RC cutting plane. \((R = [5.0, 25.0], \ L = 0.645, \ C = [3.14, 4.5])\).

FIGURE 8. Diagram for the RL cutting plane. \((R = [5.0, 25.0], \ L = [0.16, 1.45], \ C = 3.14)\).

FIGURE 9. Diagram for the RL cutting plane. \((R = [5.0, 25.0], \ L = [0.16, 1.45], \ C = 3.82)\).

may be considered the confirmation of the correctness of the calculations. The point cloud available on the Web at the indicated address has been limited in resolution. It is a result of case 3 from Table 1 and has a resolution of \(768 \times 768 \times 768\). This treatment is aimed at increasing the responsiveness of the developed web application, taking into account the limitations of the transfer of large amounts of data through the network.

Initially, the graph presentation tool was supposed to represent all points in three-dimensional space. This approach is presented in Figure 10 (the figure shows artificially generated points and colours for a three-dimensional \(10 \times 10 \times 10\) cube). The application enabled to perform the following operations on the cube: zoom in/out, rotate, move with respect to the axis and zoom in/out the point size. Figure 11 shows the result of the last operation.
VII. CONCLUSION

Calculations aimed at generating the data needed to obtain three-dimensional bifurcation diagrams are very time consuming. For this reason, it was decided not to carry out calculations using a sequential algorithm in this paper. We have decided to use a massive parallel algorithm for such time-consuming calculations and large dimensions of the problem. This method of implementing calculations using CUDA technology allowed obtaining the results in a satisfactory time.

Parallelization of the algorithm in CUDA technology for such large dimensions of the problem was a complicated task due to the amount of data and the limitations of using graphics cards to perform calculations, including a limited number of registers or memory size. The scale of the computational problem solved in this paper can be easily illustrated by referring to the number of solved systems of equations. It was 1,528,823,808 for the case 4 in Table 1. Proper organisation of calculations and division of tasks translated into a parallel algorithm that did not require a large number of massive and long transfers between the host and the CUDA device.

Nvidia CUDA technology shows great potential in parallel computing provided that the algorithm is properly implemented and that graphics card resources are used accordingly.

The use of the Three.js library allowed the implementation of a real-time program on the client side, making it generally available. A properly implemented program generates graphs in a satisfactory time. The time depends on the specification of the computer on which the graphs are viewed, as well as the speed of the network connection and the capabilities of the server.

It should be noted that the electric arc model has been used as an example only in this paper, while the presented algorithms, methods, and technology can be applied in many other fields of science where non-linear systems of differential equations are applicable.

APPENDIX A

Figure 12 shows the examples of electric arc circuits, described in the form of a system of equations (3).

However, such a presentation of data was not accepted because the browser’s responsiveness decreased and the demand for system resources increased with the increase of the size of the problem. In addition, it was difficult to view the graphs (in the form of sections) because some points obscured others. When implementing this data presentation method, the points were not loaded from the file (they were generated by the program), so it would also be necessary to take into account the time of loading the file by the application, which could be significant and affect the length of the page loading.

FIGURE 10. Initial presentation of the results in a three-dimensional space.

FIGURE 11. Initial presentation of the results in a three-dimensional space after increasing the size of points.

FIGURE 12. Two typical electric arc circuits.
The circuit shown in Fig. 12 (b) [13].

The system of equations (3) also describes the system on the left when $m = 1$ and $E = RI0 + U0$. Taking into account the appropriate conversion of variables, the system of equations (3) also describes the circuit shown in Fig. 12 (b) [13].

$$\frac{di}{d\tau} = \frac{1}{L} \left( u - \frac{U(i_0)}{i_0} \right), \quad \frac{dx}{dt} = \frac{1}{L} \left( y - x^m \right)$$

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

$$\frac{d^2 u}{d\tau^2} = \frac{1}{\theta} \left( \gamma^2 - \gamma \right), \quad \frac{dz}{dt} = x^2 - z$$

where:

$$x = \frac{i}{I_0}, \quad y = \frac{uU}{U_0}, \quad z = \frac{i}{I_0}$$

**APPENDIX B**

Figure 13 shows an example of the graph available on the page along with the entire interface. Sliders below the graph allow movement along the selected axis (there is information about the current position next to the sliders). Initially, the sliders are in the far left position — this corresponds to the first cross-section on the selected axis, while the far right position corresponds to the last graph.

You can position the slider by clicking on the desired position of the slider or by dragging the marker along the slider. The first slider loads the graphs while moving the marker, the second slider loads the graph only after the marker is placed at the selected point. You can change the axis along which the movement will take place by using buttons 1, 2 and 3. You can also move along the axis using the arrows — the up arrow displays the next graph, while the down arrow displays the previous graph (on the selected axis).

**REFERENCES**

[1] Y. Itoh, S. Uenohara, M. Adachi, T. Morie, and K. Aihara, “Reconstructing bifurcation diagrams only from time-series data generated by electronic circuits in discrete-time dynamical systems,” Chaos, An Interdiscip. J. Nonlinear Sci., vol. 30, no. 1, 2020, Art. no. 013128.

[2] A. Pala and M. Machaczek, “Parallel computing of two-parameter bifurcation diagrams of an electric arc model with chaotic dynamics using Nvidia CUDA and OpenMP technologies,” Przegląd Elektrotechniczny, vol. 95, no. 3, pp. 138–142, 2019.

[3] J. Awrejcewicz, J. Pyryev, G. Kudra, and P. Olejnik, “Mathematical and numerical methods of bifurcation and chaotic dynamics analysis of mechanical systems with friction and impacts,” (in Polish), Tech. Univ. Łodź Publisher, Łódź, Poland, 2006.

[4] L. N. Virgin and C. J. Begley, “Grazing bifurcations and basins of attraction in an impact-friction oscillator,” Phys. D, Nonlinear Phenomena vol. 130, nos. 1–2, pp. 43–57, 1999.

[5] J. Awrejcewicz, G. Kudra, and C.-H. Lamarque, “Investigation of triple pendulum with impacts using fundamental solution matrices,” Int. J. Bifurcation Chaos, vol. 14, no. 12, pp. 4191–4213, 2004.

[6] M. Mali, N. Goyal, and H. Trivedi, “Geometric methods in the theory of, Ordinary Differential equations: A review,” Int. J. Res. Sci. Innov., RTSS, vol. 1, no. 4, pp. 37–43, Sep. 2014.

[7] P. V. Asavei, V. Ionita, F. Moldoveanu, and A. Moldoveanu, “3D real-time parallel volume rendering using NVIDIA CUDA for medical imaging,” DAAAM Int., Vienna, Austria, Tech. Rep., Nov. 2009, vol. 1, pp. 1483–1485.

[8] L. Shi, W. Liu, H. Zhang, Y. Xie, and D. Wang, “A survey of GPU-based medical image computing techniques,” Quant. Imag. Med. Surg., vol. 2, no. 3, p. 188, 2012.

[9] A. Harju, T. Siro, F. Canova, S. Hakala, and T. Rantalaahlo, “Computational physics on graphics processing units,” in Applied Parallel and Scientific Computing, Helsinki, Finland: Springer, Jun. 2012.

[10] S. Sosutha and D. Mohana, “Heterogeneous parallel computing using cuda for chemical process,” Procedia Comput. Sci., vol. 47, pp. 237–246, Jan. 2015.

[11] O. Seiskari, J. Kommeri, and T. Niemi, GPU in Physics Computation: Case Geant4 Navigation, vol. 23, Geneva, Switzerland: CERN, CH-1211, Jan. 2011, [Online]. Available: arXiv:1209.5235

[12] R. F. Ammerman, T. Gammon, P. K. Sen, and J. P. Nelson, “DC-arc models and numerical methods of bifurcation and chaotic dynamics analysis of mechanical systems with friction and impacts,” IEEE Trans. Ind. Appl., vol. 46, no. 5, pp. 1810–1819, Sept./Oct. 2010.

[13] V. N. Sydorets and I. V. Pentegov, “Deterministic Chaos in Nonlinear Circuits with Arcs,” Kiev, Ukraine: Svarka, 2013.

[14] I. V. Pentegov and V. N. Sydorets, “Comparative analysis of models of dynamic welding arc,” Paton Welding J., vol. 12, pp. 45–48, Dec. 2015, doi: 10.15407/pwj2015.12.09.

[15] W. Marszalek and H. Podhaisky, “2D bifurcations and newtonian properties of memristive Chua’s circuits,” EPL (Europhys. Lett.), vol. 113, no. 1, 2016, Art. no. 10005.
[16] W. Marszałek and J. Sadecki, “2D bifurcations and chaos in nonlinear circuits: A parallel computational approach,” in Proc. 15th Int. Conf. Synth., Modeling, Anal. Simulation Methods Appl. Circuit Design (SMACD), Prague (Czech Republic), Jul. 2018, pp. 2–5.

[17] W. Marszałek, “Autonomous implicit models of pinched hysteresis with application to memristors,” Przegląd Elektrotechniczny, vol. 94, no. 4, pp. 13–16, 2018.

[18] S. C. Hapra, R. P. Canale, Numerical Methods for Engineers. New York, NY, USA: McGraw-Hill, 2011.

[19] S. D. Conte and C. De Boor, “Elementary numerical analysis,” in An Algorithmic Approach. New York, NY, USA: McGraw-Hill, 1980.

[20] E. Kącki, A. Malolepszy, and A. Romanowicz, (in Polish). Numerical Methods for Engineers, Łódź, Poland: Technical Univ. Łódź Publisher, 2000

[21] Nvidia Corporation: Nvidia CUDA C Programming Guide, Nvidia Corporation, Santa Clara, CA, USA, 2015.

[22] Nvidia Corporation: An Introduction to GPU Computing and CUDA Architecture, Nvidia Corporation, Santa Clara, CA, USA, 2011.

[23] D. B. Kirk, Wen-Mei, and W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach. San Mateo, CA, USA: Morgan Kaufmann, 2010.

[24] D. Storti and M. Yurtoglu, CUDA for Engineers: An Introduction to High-Performance Parallel Computing. Reading, MA, USA: Addison-Wesley Professional, 2015.

[25] J. Sanders and E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming. Reading, MA, USA: Addison-Wesley Professional, 2010.

[26] (Dec. 2019). Official Three.Js Library Documentation. [Online]. Available: https://threejs.org/docs/

[27] (Dec. 2019). Three.Js Fundamentals. [Online]. Available: https://threejsfundamentals.org/threejs/lessons/threejs-fundamentals.html

[28] (Dec. 2019). Documentation for the Three.Js library. [Online]. Available: https://discoverthreejs.com/

[29] J. Dirksen, Learn Three.Js, 3rd ed. Birmingham, U.K.: Packt Publishing, 2018.

[30] J. Dirksen, Three.Js Essentials. Birmingham, U.K.: Packt Publishing, 2014.

ARTUR PALA received the Diploma in Engineering degree from the Faculty of Applied Computer Science, University of Applied Sciences in Nysa, in 2008, and the master’s degree in computer science from the Faculty of Electrical Engineering, Automation, and Computer Science, Opole University of Technology, in 2010. He is currently an Employee of the Institute of Computer Science, Opole University of Technology. As a part of his doctoral thesis, he conducts research in the field of application of massive parallel computation technology (on the example of Nvidia CUDA) to solve systems of ordinary differential equations and dynamic optimization problems.

MAREK MACHACZEK received the M.Sc. degree in computer science from the Faculty of Automation, Electrical Engineering, and Computer Science, Opole University of Technology, in 2013. He is currently an Employee of the Institute of Computer Science, Opole University of Technology. As part of his doctoral thesis, he focuses on parallel calculations in application to static optimization methods.

* * *