OCA – Graphical System for Algorithm Structure Analysis and Processing

Anatoliy Melnyk¹ and Inna Iakovlieva²

¹ Department of Computer Engineering, Lviv Polytechnic National University / 12, S. Bandera Str., Lviv, 79013, Ukraine / amelnyk@intron-innovations.com
² Department of Computer Systems and Networks, Yuriy Fedkovych Chernivtsi National University / 2, Kotsubinsky Str., Chernivtsi, 58012, Ukraine / iakovlieva@yandex.ru

* Corresponding Author: Anatoliy Melnyk

Received January 25, 2012; Revised March 17, 2012; Accepted April 03, 2012; Published April 30, 2012

Abstract: We suggest a graphical system for analyzing and processing algorithm structure with a user-friendly interface and a set of necessary tools. The system is designed to make the process of visualization, modification, and analysis of algorithm graphs easier. The system provides algorithm flow graph building and analysis, the algorithm flow graph height and width adjustment, algorithm simplification due to the data output rule, and algorithm flow graph parameters output.

Keywords: graphic system, algorithm structure, algorithm graph, algorithm flow graph, algorithm analysis

Introduction

The time it takes for computer systems with parallel architecture to solve problems depends not only on the complexity of the algorithm performed, i.e., the number of operations, but also on the algorithm structure and the selected sequence of its operations performance [1-3]. Faster algorithm performance in such systems is achieved by a certain number of processing elements (PE) that can simultaneously perform independent operations of the algorithm. To realize what algorithm operations are independent, one should be informed about their place in the algorithm and the connections between operations of the algorithm. The need to identify independent algorithm operations leads one to naturally want to view a mathematical representation of its structure. The algorithm structure is presented by its graph. Thus, the algorithm graph (AG) is basic tool for obtaining information about its operations connection. To find the time and spatial dependence of the algorithm operations, the algorithm is presented as an algorithm flow graph (AFG) [1]. AFG building means distribution of all algorithm vertices, i.e. their functional operators (FO), in tiers so that in the i-th layer...
only FO depend on previous tiers’ FO and independent of where the following tiers’ FO would be located. Within one tier, there are no connections between the AFG vertices. Such a form of algorithm presentation enables one to estimate the potentiality of its paralleling (the number of vertices in one tier), as well as minimally possible computation time of the given algorithm (the number of tiers). The need to analyze and process the structure of an algorithm arises very often, particularly when it is time to estimate performance time and complexity. Algorithm structure modification is often required, for instance, in order to decrease the FO number in the tier if the computer system does not contain enough PE for parallel performance. The algorithm can be simplified by simplifying AFG, excluding operations that do not participate in forming the resulting range of an output data vector. Obtaining different AFG variants, computing their complexity characteristics, and comparing them in order to choose the best variant for further use are important problems. Therefore, the task is to create software with a user-friendly interface and a set of necessary tools, by means of which the process of displaying, modification, and analysis of algorithm graphs could be simplified. This task is performed by creating a graphical system OCA for analyzing and processing the algorithm structure, the functions and application of which are described in this paper.

State of the Art

Analysis of the Techniques and Means of Algorithm Presentation

By “algorithm” one usually means a direct instruction that determines the computing process used to solve a problem. It is well known that for an algorithm presentation verbal-deductive [4], analytical [5] and graphic [4] techniques can be used. A verbal-deductive technique [4] is not visual and is not suitable for performing certain formal transformations with algorithms. Logical schemes of algorithms can be included into the analytical group of algorithm presentation techniques [5]. The main advantage of analytical presentation consists of algorithm written compactness in the form of a system of formulae. However, sophisticated processing and low visualization make them unsuitable when evaluating the performance of algorithms and estimating the correct approach to implement them in hardware. The graphical technique of algorithm presentation is characterized by high visualization due to displaying the functional operators of the algorithm and connections between them. Algorithm presentation as a graph is one of the possible graphical forms of the graphical technique of algorithm presentation.

Figure 1. Algorithm graph for calculating the expression $y = (a + b) \times (c - d) + e \times f$:

- $a$: algorithm graph; $b$: flow algorithm graph; $F$: functional operator

As an example, Figure 1 (a) presents an algorithm graph (AG) of calculating the expression: $y = (a + b) \times (c - d) + e \times f$, every vertex of which displays FO, and every arc – the connection between FO. The vertex corresponding to functional operator $F_i$ is connected with that corresponding to functional operator $F_j$, only in the case when the result obtained after the $F_i$ operator performance is one of the arguments for the $F_j$ operator. Such an AG (Figure 1(a)) does not enable one to completely estimate the potentiality of algorithm paralleling. If the spatial algorithm performance is considered, it is possible to distribute the groups of algorithm operations into the graph tiers that possess the following features: 1) there are only independent operations in every tier; 2) there is such a sequential numbering of tiers that every operation of any tier uses either results of performing operations from the tiers with lower numbers, or input data of the algorithm as an argument. That is, all operations of the same group are independent and can be performed simultaneously, while the groups themselves are realized sequentially in time. Within one tier there are no connections
between operations. This representation is called a tier-parallel algorithm form [2], or an algorithm flow graph [1], and the parameters characterizing this representation – space-time ones. Figure 1 (b) presents AFG of calculating the expression: \( y = (a + b) \times (c - d) + e \times f \) as an example. If the performance of the algorithm suggested in AFG form is considered, it is supposed that all data are simultaneously moving in the AFG arcs. If a vertex appears on the way of the data moving in arcs, then the data participates in the operation, assigned by this AFG vertex. In other words – they are transferred to the next tier. As a result of data flowing between the FO of AFG and the performance of operations assigned by the algorithm, the end result data about performing the algorithm, presented in the AFG form, is obtained at AFG output. The number of operations in the i-th tier of AFG is the tier width \( w_i \), and the number of AFG tiers \( l \) is its height, where \( i = 1, \ldots, l \). Every algorithm can have several types of AFG for a fixed amount of input data. AFG, in which all the tiers have a width equal to unity, have maximum height equal to the general number of algorithm operations, and represent sequential calculations. However, AFG of minimal height are of the greatest interest, as they give information on the possibility of paralleling the algorithm and therefore taking the least amount of time to do so. An AFG of minimal height is called canonical [2]. For any algorithm with the given input data, there always exists only one canonical AFG; it is of minimal height. In canonical AFG all tiers have maximal width.

Algorithm Graph Visualization

It is obvious that to ensure the analysis and processing of algorithm structure it is necessary to provide an AFG feed in the computer in a form suitable for performing these procedures. Algorithm graph visualization is a labour-intensive process, as their initial models presented for visualization (these are mostly adjacency matrices, connectivity tables, etc.) do not contain information concerning the mutual location of the graph vertices on the plane. Such a situation enables one to obtain a practically unlimited amount of various graph images of the same algorithms. Techniques and approaches to graph visualizations are very different, and they are chosen depending on what graph is to be visualized [10-14]. With the appearance of a number of standards for working with structured documents coordinated by the World Wide Web Consortium (W3C) [15, 16] and called the W3C Document Object Model (W3C DOM), the techniques and means of graphs visualization further advanced. These standards do not depend on the platform or programming language and the documents are called XML documents. XML stands for Extensible Markup Language [15], and it outlines a set of basic lexical and syntactical rules for construction of a language of information description using simple tags. It is flexible enough for suitable application in different fields. The DOM standard [16] defines the classes, techniques, and attributes of these techniques for document structure analysis and presenting documents in the form of a tree. It enables computer programs to gain access and dynamic modification of the structure, content, and document format. This caused the appearance of such new languages and formats of graph description as GraphML – a file format for graph description, enabling to perform graph formatting, archiving and processing [17]; Graph eXchange Language (GXL) [18]; eXtensible Graph Markup and Modeling Language (XGML) [19], Scalable Vector Graphics (SVG) [20], based on standards [15,16] and others [21]. Numerous studies were carried out and various tools for graph visualization were created [10-21], though, at the same time, there is no specialized graphical system for algorithm processing. Thus, the authors created the OCA system – a graphical system for the analysis and processing of algorithmic structure [22]. The problem of creating software facilities for the analysis and processing of algorithm structure by displaying, modification, and analysis of their graphs is very important. Such a facility is considered in this paper, namely, the OCA system – a software product with a user-friendly interface and a set of required tools by means of which these procedures are performed. The OCA system ensures the analysis of algorithm graphs and building flow graphs, AFG width and height changing, algorithm simplifying according to the data output rule, and AFG parameters output.

Operation of the Graphic System for Analysis and Processing of Algorithms Structure

General Structure of OCA System

The OCA system consists of 14 modules that provide information input, information output, and specify the modes and options of the software in the process of the dialogue with the user (Figure 2).

Initial Data Input

The module of initial data input operates in two modes: that of creating the algorithm graph and that of downloading the AG. The user chooses the mode. The first mode is used at a relatively small amount of vertices of the algorithm graph. At that, the vertices are located on the working area of the window (Figure 3) and the connections between them are drawn.
The process of displaying AG consists in "dragging" the vertices from the window of standard components by means of a mouse and further connecting them with one another. The AG vertices are numbered automatically. The elements of the image can be moved, connected with one another, deleted, signed, etc. For user convenience it is possible to adjust the size and colour of the working window and the net's sizes.

The second mode presupposes AFG downloading from the text files, storing the algorithm structure either in matrix form or in the form of XML files, which had been obtained and saved in this software previously. As an example, Figure 4 presents the AFG sorting by a modified bubble sort method displayed in OCA systems [23], by the odd-even sorting method [24], and by the Batcher method [25] for 16 pieces of input data. The displayed AFG suggest a visual idea of the algorithm structure and its parameters. Thus, it can be seen from Figure 4 (a) and Figure 4(b) that sorting algorithms by bubble sort and the odd-even sorting method have an equal amount of "compare and swap" operations, which are distributed into a different number of tiers. AFG sorting by the Batcher method shows both the least amount of operations and tiers. This information enables us to choose the best algorithm for solving a certain problem.
Algorithm Structure Analysis

Algorithm structure analysis is carried out by means of modules ensuring:

- AFG building;
- AFG width changing;
- AFG height decreasing;
- Algorithm simplification due to the input rule;
- AFG parameters output.

Algorithm Flow Graph Building

As AG does not allow one to estimate space-time parameters of the algorithm and the possibility of it becoming completely parallel, the OCA system provides AG transformation into AFG (Figure 5).

The following can be automatically determined from AFG:

- the number of input vertices, from which data arrives;
- the number of AFG tiers, equal to the AFG critical path;
- the number of FO in every tier;
- AFG width;
- general amount of FO of AFG;

Figure 4. AFG of sorting 16 input data: a – by the modified bubble sorting; b – by the odd-even sorting; c – by the Batcher sorting; d, e – basic operation "to compare and swap"
- tiered density of the i-th tier equal to the ratio of i-th tier width to AFG width;
- the numbers of input and output arcs given for every vertex-FO of AFG.

**Figure 5.** Displaying AG: a – AG; b – its AFG

The time delay on the critical path in the number of performed operations presents integral characteristics of the algorithm, as it covers other time characteristics and, in many cases, it defines the reasonability of performing a certain algorithm. This theoretically calculated characteristic determines the performance speed of the algorithm and/or the time delay of the computing device that realized the given algorithm in hardware. Tiered density should tend to unity at best, and it enables one to analyze the efficiency of occupancy of PE, on which the given algorithm will be performed. The tier width reflects the amount of PE used at every moment of time. AFG characteristics are displayed on the screen as corresponding electronic tables that can be processed by other software for working with electronic tables. As an example, Figure 6 presents an AFG of binary multiplication with horizontal spreading of swapping on the graph of distribution of FO operations of dyadic single-digit addition into tiers of AFG for various capacities of factors.

**Figure 6.** Operations distribution of dyadic single-digit addition into tiers of AFG

**AFG Width Change**

As it had already been mentioned, for any algorithm at the given input data there is a canonical AFG having the minimal height; tiers have the maximally possible width and the number of independent FO operations in every tier can be different. Between AFGs with minimal height and that with maximal height there are still more AFGs with intermediate height and with certain amount of corresponding FO in every tier. The tier’s width indicates the possible number of PE used at every moment of time, while the height shows the time of the algorithm performance. Such graphs assign equivalent algorithms
[2], which differ in topology only. Such forms can be efficiently applied for optimal occupancy of the previously-given amount of parallel PE that is more than unity and lesser than the AFG width. Due to the fact that the amount of FO varies between AFG tiers, it is possible to move FO from one tier to another and to change the AFG width from its maximal value up to a complete absence of parallelism. Changing AFG width in the OCA system occurs in two modes. The mode is chosen by a user. The first mode is used for obtaining information about all AFGs with different widths. The second one presupposes displaying AFGs with the chosen width. In the first mode, a table of the following parameters for AFGs of different widths is obtained: AFG width, the number of AFG tiers (height), minimal and maximal amount of FO in a tier, and average amount of FO in AFG tiers. Figure 7 presents the table of parameters of fast Fourier transforms (FFT) of AFGs of various widths [26].

![Information](image)

**Figure 7.** Table of FO FFT distribution in tiers at AFG width changing

![Diagram](image)

**Figure 8.** Set of parallel AFG FFT forms: a – AFG 4 width; b – AFG 3 width; c – AFG 2 width

This information enables one to choose an AFG width under which the maximal occupancy of PE, used to perform the given algorithm, is achieved. The second mode displays AFG with the given width. Figure 8 presents AFG FFT for 8 input values with different widths. The AFG width change leads to the change in the amount of AFG tiers and shows algorithm performance time depending on the performed FO number at every moment of time. Thus, at AFG width equal to 4, the number of tiers is 3, while at AFG width is equal to 3, the tiers number is 5, and so on. The results obtained by means of an OCA system are presented as a graph of dependence of AFG FFT height changes on AFG width changes for different pieces of input data in Figure 9.
Using the results obtained by the OCA system for different sorting algorithms, the graphs of AFG height dependence on AFG width for 16 input values were built (Figure 10).

The results obtained (Figure 10) show that sorting by the Batcher method at the sequential performance of operations (AFG width=1) for 16 pieces of input data is performed two times faster than by other methods. Paralleling the algorithm decreases the performance time threefold if compared with the modified "bubble" method, and if it is compared with the odd-even sorting method, by 1.6 times. If compared with sequential sorting, parallel sorting of 16 pieces of input data is performed faster: for the modified "bubble" method – by 4.1 times; for the odd-even sorting method – by 7.5 times; for the Batcher method – by 6.3 times. It hyperbolically increases for increasing amounts of input data. Analysis of the AFG FFT width change (Figure 8) proved that with the change of AFG width, the number of basic "butterfly" operations that can be performed at every tier changes non-linearly. This, in its turn, causes incomplete occupancy of PE, on which the algorithm is performed. An AFG width change leads to a change in tiered density. Tiered density of the i-th tier is equal to the relation of the i-th tier width to that of AFG. The information about the value of tiered density enables one to determine the occupancy of PE, on which the algorithm is performed. If the tiered density tends to unity, the PE occupancy tends to 100%. Figure 11 shows the dependence of AFG height and PE occupancy on the width of the FFT algorithm for 32, 512, 1024, and 2048 bytes of data. If the width of the parallel form of the FFT algorithm for 512 bytes of data decreases (Figure 11, a), the PE occupancy decreases from 100% to 64% at $w = 222$, and then again approaches 100% at $w = 128$. A further decrease of the width of the parallel form of the algorithm, the PE occupancy decreases to 89%. At $w \leq 10$, its occupancy approaches 100%, though the number of tiers constantly increases. That is, with the decrease of the width of the parallel form of the FFT algorithm, the PE occupancy abruptly decreases from 100% in points $2^n$ to 60% at a distance from such points, while the AFG height increases monotonously. The obtained results enable one to choose the optimal width of the parallel form of the algorithm for maximal occupancy of PE, and for performing the algorithm during the possibly minimal time.
Algorithm Simplification due to the output rule

The performance time for any algorithm depends on the number of operations performed. There are tasks for which it is necessary to obtain not the whole vector of output data that is determined by the given algorithm, but only a range of the result vectors, assigned by the output rule [27]. As only the range of the output data vector is needed as a result, it is obvious that not all FO participate in its formation. Thus, it is necessary to perform only those operations that directly account for obtaining the chosen range of output data or on which it depends.

Figure 11. Dependence of PE occupancy on AFG iFFT width: a – for N=32; b – for N=512; c – for N=1024; d – for N=2048

So the performance time of algorithm calculation can be decreased by means of AFG simplification due to the output rule, by excluding from the operations that do not participate in forming the resulting range of the output data vector from the algorithm. Algorithm simplification due to output rule [27] is performed in the OCA system in two modes. The first mode is used for obtaining information about simplified algorithms due to the output rule of the chosen AFG. In this mode the user sets the AFG and obtains a table decreasing the number of operations of the algorithm for the value of output vector, from the maximal value to unity. The second mode suggests obtaining the image of a simplified algorithm depending on the output data vector. For that, the user sets the output data vector for AFG and obtains a modified AFG with information about the number of FO excluded from AFG, if they do not participate in forming this output data vector. Figure 12 presents a window of the OCA system in the second mode of algorithm simplification due to the output rule. The FFT N=2 algorithm simplification due to the output rule for the range [1, 2] led to the change in the order of analysis of input data, and only 10 operations are required instead of 12.

AFG FFT for N=32 reference points is presented in Figure 13, and AFG FFT for the range of output vertices [1, 10] truncated at the output by this method is shown in Figure 14. At truncations up to 10, the basic “butterfly” operations number decreased from 80 to 43. Calculation complexity of the algorithm was simplified to about 50%. The analysis of truncation of FFT N = 128 + 4096 reference points for the output data vector from the maximal value to unity enabled one to obtain data and build graphs simplifying the calculation complexity of algorithms. The results obtained showed the simplification of calculation complexity of the algorithm from 70 to 80% at the best, if compared with the total number of
operations (Figure 15), and at $N = 65563$ to 90%.

**Figure 12.** Window of the second mode of the algorithm simplification due to the output rule

**Figure 13.** AFG of FFT algorithm for $N=32$

### AFG Height Decrease

One of the purposes of algorithm parallelization is building AFG graphs of minimal height. While analyzing a certain already-built AFG, one can see that the following situation often occurs: the k-th operation that is to be performed in the i-th tier, according to the algorithm, can be performed earlier, as it does not depend on the results of performing the previous (i-1)-th tier. At that, there occurs the change of the order of performing operations or blocks of operations, set by the algorithm or obtained due to AFG modification and AFG height decrease, and, correspondingly, the performance time of
the algorithm decreases. This mode of the algorithm AFG height decreasing in the OSDA system is activated automatically at any AFG modification and can be chosen by the user manually. Figure 16 illustrates the application of this mode of decreasing the height of AFG sorting by the Batcher method for five pieces of input data.

![Truncated AFG FFT at the output for the range of the output vertices](image)

**Figure 14.** Truncated AFG FFT at the output for the range of the output vertices [1, 10]

![Decrease of operations for N reference points of FFT at truncation of output data vector from unity to the maximal value](image)

**Figure 15.** Decrease of operations for N reference points of FFT at truncation of output data vector from unity to the maximal value

Without changing the total FO number, having just changed their number in the tiers, the decrease of AFG height per tier is obtained. Having used the given mode for a modified mode of the Batcher sorting with the value of input vector of data from 1 to 2048 bytes (Table 1), the results of AFG height decrease are obtained averagely by 16.3% with further moderate extinction (Figure 17) with the increase of input vectors of data.
Here the AFG height decrease occurs not because of operations excluding, but due to their optimal location in AFG tiers that enables the decrease of the total number of AFG tiers and, thus, the performance time of the parallel algorithm.

**Results Analysis**

The results of algorithm graphs analysis are displayed on the screen as corresponding AFGs. The AFG obtained can be saved in the formats JPEG, BMP, PNG or in the format DOM. The results of AFG analysis are displayed on the screen in the form of corresponding electronic tables that can be processed by other software for electronic tables, e.g., Microsoft Excel.

**Hardware and Software Requirements**

Software for the OCA System is developed in the C++ language in the medium QtCreator IDE. The developed software product is oriented to the work in two operating environments: Windows 98/ME/NT/2000/XP, and Linux 2.6.27 with the KDE window environment. Minimal requirements of PC hardware are: a 6th generation CPU (AMD K6-2 300 MGz or better, Intel Pentium Pro/II/Celeron 300 MGz or better); 64 MB of RAM; 4 MB graphic adapter; 2 GB of hard disk space; a
CD drive; and a mouse.

**Table 1. The Decrease of the Number of Tiers of Modified Batcher Algorithm**

<table>
<thead>
<tr>
<th>The input data vector value</th>
<th>The number of tiers of the parallel form</th>
<th>The decreased number of tiers</th>
<th>Percentage of the decrease of the number of tiers %</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^2 + 1$</td>
<td>6</td>
<td>5</td>
<td>16,7</td>
</tr>
<tr>
<td>$2^3 + 1$</td>
<td>10</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>$2^4 + 1$</td>
<td>15</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>$2^5 + 1$</td>
<td>21</td>
<td>17</td>
<td>19,05</td>
</tr>
<tr>
<td>$2^6 + 1$</td>
<td>28</td>
<td>23</td>
<td>17,8</td>
</tr>
<tr>
<td>$2^7 + 1$</td>
<td>36</td>
<td>30</td>
<td>16,7</td>
</tr>
<tr>
<td>$2^8 + 1$</td>
<td>45</td>
<td>39</td>
<td>13,33</td>
</tr>
<tr>
<td>$2^9 + 1$</td>
<td>55</td>
<td>47</td>
<td>14,55</td>
</tr>
<tr>
<td>$2^{10} + 1$</td>
<td>66</td>
<td>58</td>
<td>12,12</td>
</tr>
<tr>
<td>$2^{11} + 1$</td>
<td>78</td>
<td>68</td>
<td>12,82</td>
</tr>
</tbody>
</table>

**Conclusion**

The suggested and realized graphic system for analysis and processing of the algorithm structure ensures: AG analysis and its AFG building; width changes; AFG simplification due to the output rule; AFG height decreases; and AFG parameter output. The OCA system enables one to obtain different variants of flow graphs of the same algorithm, to estimate their characteristics and to choose the best variant for further applications.

**References**

Anatoliy Melnyk is Professor and Head of the Department of Computer Engineering at Lviv Polytechnic National University. He graduated from Lviv Polytechnic Institute with a degree in computer engineering in 1978. In 1985, he received a Ph.D. in Computer Systems from Moscow Power Engineering Institute. In 1992 he received a D.Sc. from the Institute of Modelling Problems in Power Engineering of the National Academy of Science of Ukraine. He was recognized as a Fellow Scientific Researcher in 1988 and Professor in 1996. Since 1997, he has been the Scientific Director of the Institute of Measurement and Computer Technique NDKI ELVIT of Lviv Polytechnic National University. Since 1999, he has been the Dean of the Department of Computer and Information Technologies of the Institute of Business and Perspective Technologies, Lviv, Ukraine. Since 2000, he has been the Director of Research and Production Enterprise at Intron. He is a member of the guidance commission on computer engineering of the Ministry of Science and Education of the Ukraine, and a member of the expert commission of the state accreditation commission of the Ukraine; a member of IEEE, ACM, IACSS, AESU, and main editor of the proceedings “Computer Systems and Networks.” He has taken part as a project leader in performing a large number of scientific projects in the field of computer systems. He has written over 350 scientific papers and patents.

Inna Iakovleva graduated from the Computer Department of Chernivtzi State University in the area of electronic computers, complexes, systems, and networks. She received an engineer degree in 1993. In 2010 she received a Ph.D. thesis in Computer Systems and Components at the Lviv Polytechnic National University. She has been working in Chernivtzi National University since 2003 and as an Assistant Professor of the Computer Systems and Networks Department since 2011. Her research interests include specialized computers and computer systems, theoretical foundations of their construction and methods of design, synthesis and implementation of computer solutions for solving data-flow studies and processing algorithms, parallelization of algorithms, and design of specialized processors from graphical level.