ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2020, Vol. 41, No. 8, pp. 1427–1434. © Pleiades Publishing, Ltd., 2020.

Information Graph Visualization Using AlgoView Software Tool

A. S. Antonov[1]* and N. I. Volkov[1,2]**

(Submitted by E. E. Tyrtyshnikov)

[1] *Research Computing Center, Moscow State University, Moscow, 119991 Russia*

[2] *"TESIS" Company, Moscow, 127083 Russia*

Received March 31, 2020; revised April 4, 2020; accepted April 15, 2020

Abstract—One of the approaches to the study of the information structure of algorithms is the analysis of information graphs. These are complex objects, the understanding of the structure of which can significantly facilitate their visualization. However, it is difficult to visualize large graphs, while maintaining clarity of the information structure of the algorithm is generally very difficult. Therefore, we are developing a tool for three-dimensional interactive visualization of information graphs called AlgoView, designed to facilitate this task. At the moment, AlgoView is embedded as a visualizer on separate pages of the AlgoWiki Open encyclopedia of parallel algorithmic features, and in the near future it is planned to obtain a stand-alone version of this system.

DOI: 10.1134/S199508022008003X

Keywords and phrases: *information graph, parallel structure, visualization, level parallel form, AlgoView, AlgoWiki.*

1. INTRODUCTION

1.1. Information Graphs

Algorithmic parallelism has been an essential feature of high performance computing for decades. Thus, the search for potential parallelism on all possible levels and the choice of an optimal way to parallelize the algorithm are the key milestones one faces when writing an application for high-performance computing system. The answer to these questions depends on, firstly, the algorithm structure and, secondly, on the high-performance system architecture. Even though nowadays we don't write applications for particular systems, cases of applications that make use of specific hardware architecture families are still common. Meanwhile hardware architecture specifications are normally publically available and thus well-known to developers, specifications for newly developed applications simply do not exist and have to be built from scratch. Algorithm analysis provides a basic framework for coming up with these specifications including key features such as an algorithm's source of parallelism. A number of classical studies are devoted to methods for analyzing cyclic constructions and iteration spaces, for example, in [1–4]. Currently, several research groups are moving along the path of studying typical algorithmic structures (for example, [5]). But most of these methods have only limited applicability and do not describe the full potential of parallelism of the studied fragment of the program. To get rid of these shortcomings, approaches are used that focus on the use of a concept of an algorithm's information graph [6].

Information graph of an algorithm is a Directional Acyclic Graph (DAG) [7, 8] that represents the internal structure of the given algorithm. Its vertices correspond to operations within the algorithm and edges correspond to data dependencies between those operations. Information graphs possess a solid potential as means of an algorithm's visual representation as they allow both an expert express evaluation of the algorithm's structure and detailed analysis of its properties through the properties of the given graph. Information graphs are particularly useful for analyzing the resource of algorithm's parallelism and thus its theoretical speedup potential when launched on various computational platforms. A detailed theory of algorithm information graphs is given in [6].

*E-mail: asa@parallel.ru

**E-mail: volkovnikita94@gmail.com

1.2. Visual Representation of Information Graph

Algorithm information graphs are analyzable in many ways. To start with, basic graph metrics and features, such as vertex average and median vertex degree, amount of connectivity components and so on can be applied to information graphs. Secondly, some of these common metrics play a critical role in algorithm information graph analysis, e.g. graph critical route. Thirdly, some metrics such as features of a graph's level parallel form are specific to information graphs and do not apply anywhere else in graph theory. One of these "metric" is a regular graph structure—a thin graph characteristic that is described by the graph vertices relationship pattern. While visual representation is still not necessary in order to analyze graph regular structure, it has proven itself useful for information graph express analysis and as a supporting feature for mathematical information graph description. Creation of information graphs' visual representations was implemented in the V-Ray system [9] and hand-drawn visual representations were used in the AlgoWiki [10, 11] project way before we've developed a corresponding system. Moreover, we've even created a cookbook on how to properly manually implement information graph representations. An example of such a representation is given below.

However, currently we have a functional graph visualization system and examples of interactive visual representations for information graphs built by it can be found online at the AlgoWiki project domain. These are provided by AlgoView visualization system we have developed, which transitions initial algorithm descriptions through a pipeline of transformations resulting in a set of visual 3d-models which are finally presented online to an end user. The detailed AlgoView pipeline description can be found in the future sections of this paper. These representations are, as it was mentioned before, interactive, which means that built-in analysis tools such as information graph vertex typization, projections and level parallel form are provided by the system. The web part of the system is, however, only capable of display the pre-generated information graphs and thus we have developed two toolsets used for creating those. The first is a desktop application that takes algorithm description written in special markup language as an input, asks end user to specify some algorithm parameters and features and then provides a set of information graph visual models that can be further uploaded online and used within the web part of the system. The second toolset is still under development and is an online service that provides pretty much the same pipeline for the end user, only available online.

2. ALGOVIEW SOFTWARE TOOL

2.1. Program Basis

The desktop client core written in C++ and Qt framework used for GUI design. The core part responsible for actual graph representation generation also makes use of a list of third-party libraries, including RapidXML [12] for initial input analysis (as the algorithm descriptions written in markup language have .xml format) and DOM-tree construction, Exrtk for regular expression analysis, as such expressions play a role of great importance within the algorithm description, serving as decisive rules for whether the resulting graph should contain certain vertices and edges.

The web client core utilizes the same set of libraries and frameworks as the desktop one, yet the user interface is completely different. The entire service runs on a nginx web-server and is written in Python with a Flask framework widely used. We also had to introduce mandatory user registration as well as collection of user history and statistics, as the system is supposed to be used as a platform for our students to solve their tasks throughout the year. Thus, the service also utilizes numerous Python libraries for managing databases (SQL Alchemy, Alembic), authorization and so on.

Finally, the visual representation display service itself is written in JavaScript and utilizes WebGL [13] technology. This part of the system was earlier described in detail in our previous papers, such as [14] and [15]. Means and principles of web visualization integration didn't change as well—AlgoWiki based on the MediaWiki engine [16] still uses IFrame [17] MediaWiki extension in order to provide interactive visualizations and our newly developed web client also allows the end user to see the results in a form of interactive visualization run in a separate site frame.

Overall, our system provides two possible pipelines used to transform input data into an interactive visual representation. The first one relies on desktop client a user downloads and works with locally. The scheme of that pipeline is given in Fig. 1. The second pipeline relies on a web client and is schematically given in Fig. 2.

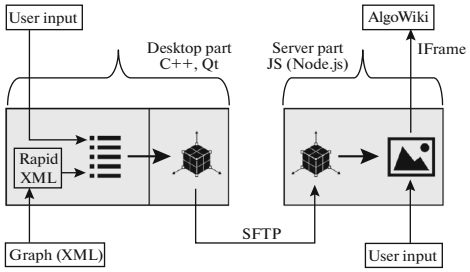

Fig. 1. Desktop client pipeline.

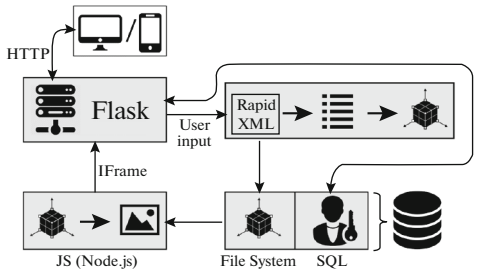

Fig. 2. Web client pipeline.

2.2. Input Data Description

The most computationally intense and deep part of the system is the pipeline used to provide information graphs based on algorithm descriptions. The desired pipeline is as follows: user-oriented markup algorithm description $\rightarrow$ machine-oriented markup $\rightarrow$ information graph $G(V, E)$ $\rightarrow$ set of JSON [18] documents containing visual representation $\rightarrow$ interactive visualization.

Both user- and machine-oriented markups basically describe the algorithm itself. Both of these are supposed to be supplied in a form of XML files of given structure, only that machine-oriented markup already exists and user-oriented is still under development, as specifying means no less than creating a totally new, convenient algorithm description language, which is the current primary purpose of our information graph-related research. Thus, the currently existing ideas and basics of the user-oriented markup will be given in the end of this paper and currently we'll focus on machine-oriented markup description and further pipeline.

Machine-oriented markup exists in two forms, the first, simplified, being just a list of resulting $G(V, E)$ vertices and edges. It has to be mentioned that the system has an internal 3-dimensional regular grid entity, so that each vertex of a resulting graph is given its coordinates within this grid, while graph edges are specified and a pair of vertices, just like in any graph description. The simplified form already has all grid coordinates defined for each vertex of the information graph. The second, generic, markup is more complicated, as it only parametrically defines loops and general data flows within the algorithm, which can be parsed given exact parameter values resulting in an actual $G(V, E)$ of an algorithm. This machine-oriented markup is supposed to be generated from the source code of a reference algorithm implementation. In Figs. 3 and 4 we provide an example of implementation source code with resulting algorithm graph visualization, and Fig. 5 demonstrates corresponding machine-oriented markup.

2.3. Visualization System Features

Our web visualizations are called "interactive" for a reason, as the end user has a solid real-time control over how exactly visualization is handed to him. In this section, we describe the key features of the AlgoView visualization system.

2.3.1. Choosing the best point of view. Adjustable point of view is the first thing that comes into one's mind when something is described as "interactive". Thus, user-adjusted camera view is the basic feature of our web visualization system.

```
int accumSum(int** _src, int _nCol, int _nRow)
{
   for (int rowId = 0; rowId < _nRow; ++rowId)
   {
      for (int colId = 0; colId < _nCol; ++colId)
      {
         _src[rowId][colId] += _src[rowId][colId == 0 ? colId : colId - 1];
         _src[rowId][colId] += _src[rowId == 0 ? rowId : rowId - 1][colId];
      }
   }
   return _src[_nRow - 1][_nCol - 1];
}
```

Fig. 3. Example source code.

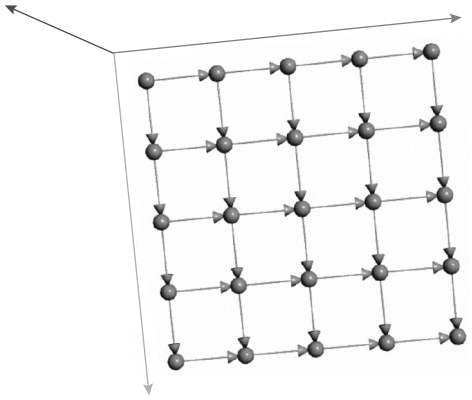

Fig. 4. Example interactive visualization.

2.3.2. Projections onto coordinate hyperplanes. Each information graph vertex has its place in 3-dimensional space when it comes to visual representation. In some cases, the entire 3-dimensional graph structure becomes rather complex and difficult to analyze. If the information graph is a four or more dimensional object, then the only reasonable way to vizualize it is to consider a set of its projections. In many cases 2-dimensional graph projections on a coordinate plane may benefit to clarifying the entire graph structure. Projections allow you to highlight the regular components of the information graph and at the same time hide insignificant details. A typical case of a regular structure that is only clear from a hyperplane projection point of view is given in Figures 6–9.

2.3.3. Using different types of vertices. In the canonical information graph definition, graph vertices are defined as correspondent to algorithm operations. Operation itself is, nevertheless, a very general definition and the exact meaning of it may vary from a 1-tact register command to a complex subroutine within the algorithm. To address that, the end user may either distinct algorithm operations into multiple types based on his own operation complexity analysis and reflect this distinction is algorithm description or rely on the source code parser doing so. Regardless of end user's choice, the resulting information graph representation will contain vertices of multiple types, where vertices of non-default operation types may represent subroutines within the algorithm. An example of such representation is given below in Fig. 10.

2.3.4. Level parallel form. Level parallel form of an information graph is the key point of information graph analysis, as it shows the algorithm's actual resource of parallelism. In level parallel form, all information graph vertices are assigned a level, with level 1 meaning the vertex doesn't have any data dependencies and level X means the vertex has data dependencies from vertices of level $X - 1$ and not more. Cycling through levels shows how the algorithms would've been executed within the concept of unlimited parallelism. Our web visualization system supports the level parallel form representation of an information graph, an example of cycling through the information graph levels is given in Fig. 11. Different color shades in the image correspond to different groups of parallel levels.

```
<?xml version="1.0" encoding="UTF-8"?>
<algograph version="0.0.1">
   <algorithm num_ext_params="1" num_groups="2">
      <ext_param name="n"/>
      <group id="0" num_or_parts="1" num_oper="1">
         <or_part id="0" num_and_parts="1">
            <and_part id="0"><![CDATA[]]></and_part>
            <statement id="0" num_inputs="1">
               <input id="0" num_solutions="0"/>
            </statement>
         </or_part>
      </group>
      <loop>
         <loop>
            <group id="1" num_or_parts="1" num_oper="1">
               <or_part id="0" num_and_parts="4">
                  <and_part id="0"><![CDATA[-i1<=-1]]></and_part>
                  <and_part id="1"><![CDATA[i1<=n]]></and_part>
                  <and_part id="2"><![CDATA[-i2<=-1]]></and_part>
                  <and_part id="3"><![CDATA[i2<=n]]></and_part>
                  <statement id="0" num_inputs="3">
                     <input id="0" num_solutions="0"/>
                     <input id="1" num_solutions="3">
                        <solution num_altareas="1" group_id="1" or_part_id="0" statement_id="0" dimsolut="2">
                           <altarea num_conditions="5">
                              <condition><![CDATA[i2-n<=0]]></condition>
                              <condition><![CDATA[-n+2<=0]]></condition>
                              <condition><![CDATA[i1-n<=0]]></condition>
                              <condition><![CDATA[-i1+1<=0]]></condition>
                              <condition><![CDATA[-i2+2<=0]]></condition>
                           </altarea>
                           <dependency>
                              <coordinate><![CDATA[i1]]></coordinate>
                              <coordinate><![CDATA[i2-1]]></coordinate>
                           </dependency>
                        </solution>
                        <solution num_altareas="1" group_id="1" or_part_id="0" statement_id="0" dimsolut="2">
                           <altarea num_conditions="5">
                              <condition><![CDATA[i2-n<=0]]></condition>
                              <condition><![CDATA[-n+2<=0]]></condition>
                              <condition><![CDATA[i1-n<=0]]></condition>
                              <condition><![CDATA[-i1+2<=0]]></condition>
                              <condition><![CDATA[-i2+1<=0]]></condition>
                           </altarea>
                           <dependency>
                              <coordinate><![CDATA[i1-1]]></coordinate>
                              <coordinate><![CDATA[i2]]></coordinate>
                           </dependency>
                        </solution>
                        <solution num_altareas="1" group_id="0" or_part_id="0" statement_id="0" dimsolut="0">
                           <altarea num_conditions="5">
                              <condition><![CDATA[-n+1<=0]]></condition>
                              <condition><![CDATA[-i2+1<=0]]></condition>
                              <condition><![CDATA[-i1+1<=0]]></condition>
                              <condition><![CDATA[i2-1<=0]]></condition>
                              <condition><![CDATA[i1-1<=0]]></condition>
                           </altarea>
                        </solution>
                     </input>
                     <input id="2" num_solutions="0"/>
                  </statement>
               </or_part>
            </group>
         </loop>
      </loop>
   </algorithm>
</algograph>
```

Fig. 5. Machine-oriented markup.

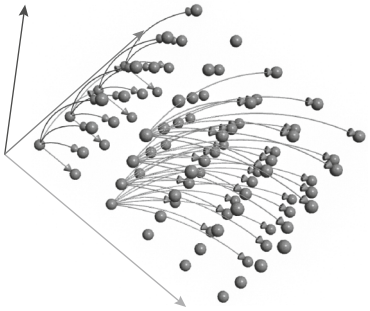

Fig. 6. Base visualization.

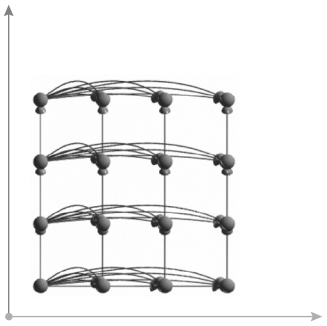

Fig. 7. oXY hyperplane projection.

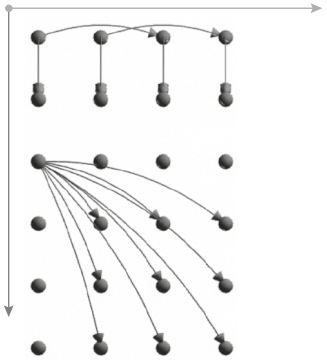

Fig. 8. oXZ hyperplane projection.

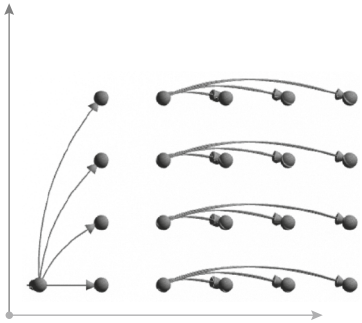

Fig. 9. oYZ hyperplane projection.

3. USING ALGOVIEW

3.1. Information Graphs Visualization on the Pages of the AlgoWiki Encyclopedia

The most critical application for AlgoView we've found is its integration in the AlgoWiki project. Each AlgoWiki page may refer to a structure, detailed description of a specified algorithm. An important part of such a description is an algorithm's information graph and we prefer to provide an interactive visualization instead of a static and schematic one. Since the MediaWiki engine used in the AlgoWiki project does not directly support WebGL technology, we used the IFrame MediaWiki widget, which provided us with such an opportunity. An example of different types of information graph visualizations, including the use of the AlgoViev interactive visualization system, can be found, for example, on a page with a description of the Givens method [19].

3.2. Using AlgoView in a Student Workshop

Another application of AlgoView is its use as a toolset for our students who're listeners of our course "Supercomputing Simulation and Technologies" [20] that has been implemented at the Faculty

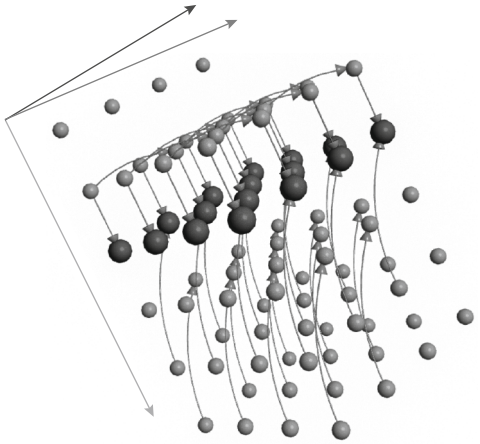

Fig. 10. An example of graph visualization with different types of vertices.

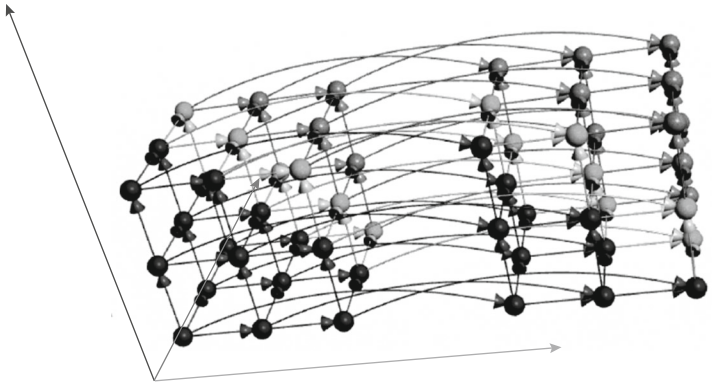

Fig. 11. An example of a graph in level parallel form.

of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. The total number of students studying this course is around 250 yearly. In 2019, this course included one "theoretical" and two "practical" assignments, the theoretical one being an analysis of a reference algorithm described by its implementation in a programming language. Building an information graph for the given algorithm was a part of the assignment and our plans (currently in development) are supplying our students with a functional web instance of the AlgoView system instead of asking them to manually create a visual representation. This is one of the main reasons we've developed a web client mentioned earlier.

4. CONCLUSIONS

We have developed the information graph web visualization system for a few years, integrated it into the AlgoWiki project and used it for stand-alone purposes. We still have plans to further develop the minor features of the system, but what we currently stall at is a convenient process of feeding algorithm markups to the system. The automated source code parser only works with a limited number of algorithm implementations and machine-oriented algorithm markups are tedious to handwrite. Thus, our key goal is developing a user-friendly algorithm description language that can be later integrated into our visualization system. This language will definitely use a dataflow-oriented approach and will be somewhat based on already existing machine-oriented markup language. We have already made a few algorithm descriptions in a way similar to what we would like to see as a final result of our work, as well as defined some language concepts and formal grammar rules. We're looking forward to completing this research in order to finally get a fully-functional and complete information graph visualization system.

FUNDING

The results were obtained in Lomonosov Moscow State University with the financial support of the Russian Foundation for Basic Research (grant no. 19-07-01030). The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

REFERENCES

1. L. Lamport, "The parallel execution of do loops," Commun. ACM **2** (17), 83–93 (1974).
2. J. R. Allen and K. Kennedy, "Automatic loop interchange," SIGPLAN Notices **6** (19), 233–246 (1984).
3. D. E. Maydan, J. L. Hennessy, and M. S. Lam, "Efficient and exact data dependence analysis," in *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation, 1991,* pp. 1–14. https://doi.org/doi10.1145/113445.113447
4. W. A. Pugh, "Practical algorithm for exact array dependence analysis," Commun. ACM **8** (35), 102–114 (1992).
5. K. Asanovic, R. Bodik, B. C. Catanzaro, et al., "The landscape of parallel computing research: A view from Berkeley," Report UCB/EECS-2006-183 (Univ. California at Berkeley, Berkeley, 2006). https://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf. Accessed 2020.
6. V. V. Voevodin and Vl. V. Voevodin, *Parallel Computing* (BHV-Petersburg, St. Petersburg, 2002) [in Russian].
7. J. A. Sharp, *Data Flow Computing: Theory and Practice* (Ablex Publ., 1992).
8. G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "DAGuE: A generic distributed DAG engine for High Performance Computing," Parallel Comput. **38**, 37–51 (2012). https://doi.org/10.1109/IPDPS.2011.281
9. V. V. Voevodin and Vl. V. Voevodin, "The V-ray technology of optimizing programs to parallel computers," in *Proceedings of the 1st Workshop on Numerical Analysis and Applications, Russe, Bulgary, 1996,* pp. 24–27.
10. Open Encyclopedia of Parallel Algorithmic Features. https://algowiki-project.org. Accessed 2020.
11. V. Voevodin, A. Antonov, and J. Dongarra, "AlgoWiki: An open encyclopedia of parallel algorithmic features," Supercomput. Front. Innov. **1** (2), 4–18 (2015). https://doi.org/10.14529/jsfi150101
12. RAPIDXML Manual. http://rapidxml.sourceforge.net/manual.html. Accessed 2020.
13. WebGL, OpenGL ES for the Web. https://www.khronos.org/webgl/. Accessed 2020.
14. A. S. Antonov and N. I. Volkov, "An web-visualization system for the AlgoWiki project," Commun. Comput. Inform. Sci. **753**, 3–13 (2017). https://doi.org/10.1007/978-3-319-67035-5_1
15. A. Antonov and N. Volkov, "Interactive 3D representation as a method of investigating information graph features," Commun. Comput. Inform. Sci. **965**, 587–598 (2018). https://doi.org/10.1007/978-3-030-05807-4_50
16. MediaWiki. https://www.mediawiki.org. Accessed 2020.
17. Extension: IframePage. https://www.mediawiki.org/wiki/Extension:IframePage. Accessed 2020.
18. Introducing JSON. https://www.json.org. Accessed 2020.
19. Givens Method. http://algowiki-project.org/en/Givens_method. Accessed 2020.
20. A. Antonov, N. Popova, and Vl. Voevodin, "Computational science and hpc education for graduate students: Paving the way to exascale," J. Parallel Distrib. Comput. **118P1**, 157–165 (2018). https://doi.org/10.1016/j.jpdc.2018.02.023