

An AlgoView Web-visualization System for the AlgoWiki Project

Alexander S. Antonov^(✉) and Nikita I. Volkov

Lomonosov Moscow State University, Moscow, Russia
asa@parallel.ru, volkovnikita94@gmail.com

Abstract. There are countless ways to define an algorithm structure, which are mostly organized by flow of data, by executed tasks or by data decomposition. The so-called information graph provides a combination of these patterns. A possibility to investigate visually the information graph of a particular algorithm is, therefore, an adequate tool that helps to understand the algorithm itself, determining its resource of parallelism and figuring out how to code it better for parallel computing systems. In this paper, we present our approach to the information graphs visualization system, where online availability and low computational cost are the primary goals.

Keywords: Information graph · Parallel form · AlgoWiki · AlgoView · Web · Visualization · Algorithm

1 Introduction

1.1 The AlgoWiki Project

The AlgoWiki project has been on the scene since 2014. Its main purpose is to describe algorithms with special attention to their logical structure, resource of parallelism [1,2], possible parallel implementations, their scalability and efficiency [13]. Attention is also paid to less-known characteristics, such as data locality [3]. Powered by the MediaWiki engine and available online by reference [4], the whole project has already been developed steadily for three years, with more than 30 articles written specifically by its authors, as well as hundreds of articles created by its visitors. It is supported by the Russian Science Foundation and officially led by Dr. J. Dongarra, from the University of Tennessee.

1.2 Why is the Information Graph Important

A very important part of an algorithm description is the so-called information graph, since it allows to show the algorithm's parallel structure both in detail and at the macro level only. An algorithm structure can be defined in countless

The results were obtained at the Lomonosov Moscow State University with the financial support of the Russian Science Foundation (agreement N 14-11-00190).

ways that follow a certain organization pattern. The *information graph* is defined in [1] as an acyclic graph where vertices stand for operations in the algorithm and edges stand for data flows. Its structure is defined by the algorithm itself and the amount of input data. Therefore, the number of vertices in an information graph is not limited in any way, while the number of edges can reach that of a fully connected graph. Such a graph provides by definition a combination of organization by flow of data, by executed tasks and by data decomposition patterns. Graph edges give a complete overview of data flows within the algorithm, while vertices and the common graph structure explain data decomposition and executed tasks in detail. A very similar term, namely DAG, is used to describe the behavior of an algorithm realization in numerous articles (see, for instance, [5]).

1.3 Related Work

Software capable of graph visualization exists for sure. Good examples of such software are [6,7]. But where would common users normally look for an algorithm description nowadays, in case they need one? Definitely, on the Internet. And it is not common to install something specific on one's laptop when willing to read a Wiki article. A notable example in this field is a whole family of molecular visualization systems, such as [8]. Furthermore, common graph visualization software is not designed specifically for displaying information graphs, in which keeping the regular structure of the original algorithm intact is a matter of primary importance. In face of this fact, it is appropriate to present an important definition from [9]: a *parallel form* of an information graph, in which all graph vertices are divided into a number of levels. If two vertices belong to the same level, then the operations they represent may be executed simultaneously. The parallel form is actually a basic restriction of graph representation aimed towards handling its structure carefully.

1.4 Setting up the Goal

Thus, building a desired system involves addressing a set of lesser tasks. The first one is to develop a method of presenting information graphs in 3-dimensional space with minimal obscurement of their regular structure, so that one can comprehend it as easily as possible. This is followed by the creation of a tool capable of performing a transformation sequence from some well-known and developed representation of an algorithm to the actual 3D-model, so that source data for the tool to work would already exist, and it would not be necessary to build them from scratch. Finally, the online-availability goal defines that a displayer should not require large computational resources from the device it is run on. However, we do not bother much about amounts of Internet connection bandwidth required for the tool to work smoothly, since such approach follows a common tendency: increasing bandwidth of networks and increasing popularity of cloud computing.

2 Proposed Solution

In this section, we successively address the tasks described in the Introduction section by giving a detailed description of the methods developed. The whole system is divided into two independent tools. The exporter tool generates a set of 3D-models by means of parsing an information graph description obtained from the reference realization of an algorithm. Doing so beforehand, we relieve users of the task of performing this costly operation. The displayer tool combines the pre-generated set into an interactive 3D-representation available online through any browser with WebGL support. A simple and well-known mechanism is used to integrate html pages containing such 3D-representations directly into Wiki articles.

The builder and displayer software tools mentioned in this section will be depicted in detail in the Implementation part.

2.1 A Method of Organizing a 3-Dimensional Graph Representation

To start with, a method that makes an automatically built 3-dimensional representation of an information graph as easy to comprehend as possible should be developed. In our previous research within the AlgoWiki project, a set of recommendations was given in order to make 2-dimensional manually-built information graph visualizations more uniform and representative. Those recommendations included such things as explicit indication of data flow directions, using different colors for vertices containing different operations, visualizing vertices standing for macro operations (subgraphs of the original information graph, each represented as a single vertex in order to simplify the entire visualization) with circles of different sizes, and many others. Basic recommendations that make common sense are used in the AlgoView system.

But the most important thing is a method we used to create 2-dimensional projections of 3-dimensional representations manually. A 3-dimensional Cartesian space and a grid in it are introduced. With the use of macro operations, many information graphs can be represented with a combination of loop nests not deeper than three levels each. So, each graph vertex is placed in a grid node with coordinates based on iteration parameters of loops in the nest for the given node, while graph edges just connect grid nodes. This approach was further developed in our builder tool to work with loop nests of four and more levels. We also address a problem of graph edges possibly interfering with each other by using square Bezier curves to calculate exact line-positions. Since such a curve is defined by three points, and we have information only about the two grid nodes being connected, the exact position of the third point also depends on the iteration parameters of the loops. The axes of the Cartesian space are added directly into the information graph 3-dimensional representation to make orientation easier.

2.2 Building 3-Dimensional Graph Models

An exact definition of “well-known and developed algorithm representation” should be given in order to address the second task mentioned in the Introduction section. In this paper, we deem the source code of a reference algorithm realization to be such a representation. We propose building a system that performs a series of transformations: source code \rightarrow inner representation \rightarrow XML representation \rightarrow a set of 3D-models \rightarrow 3D-representation. Assuming that transformations here are indexed starting with zero, the displayer tool is responsible for the third transformation, the builder is responsible for the second, an Algo-Graph tool created by our colleagues responds for the first, and performing the zeroth step is among our future plans and projects. Thus, creating a set of 3D-models from an XML file of known structure is the main transformation described in this work.

```
<?xml version="1.0" encoding="UTF-8"?>
<algorithgraph version="0.0.1">
  <algorithm num_ext_params="1" num_groups="2">
    <ext_param name="n"/>
    <loop>
      <group id="0" number="3" depth="1" 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>
  </algorithm>
</algorithgraph>
```

Fig. 1. XML file describing a simple code fragment

The main idea of the builder tool is to handle loop nests defined in an XML file by the previously described method. The XML file has the following structure (see Fig. 1). It describes one algorithm: a sequence of loops or groups; a loop consists of a sequence of loops or groups; a group stands for a set of vertices (solutions) and edges (dependencies); exact coordinates for vertices depend on exact iteration parameters of the loops a group belongs to according to the previously described method with 3-dimensional Cartesian space; an edge coordinates are based on the coordinates of the vertices it connects. Thus, the builder tool generates a list of coordinates for vertices and edges after parsing the given XML file. To create a set of 3D-models from that list, we use the following method. A set of functions is created to build geometrical 3-dimensional primitives, like spheres, cones, etc. This primitives can be transformed by passing parameters

to appropriate functions, and these parameters are taken from the generated list for each edge and each vertex.

2.3 Achieving Online Availability

Creating a set of 3-dimensional models from such an XML file is a very costly operation, that is why it is done in advance without any participation of the client's device. The generated set of models is uploaded to the same web server that stores the web pages used for the final 3-dimensional representation. WebGL technology is used to display the uploaded models on the appropriate web page, so the 3-dimensional representation is accessible via the client's browser if it supports WebGL (and modern browsers do [10]). The displayer tool is basically a set of Java scripts attached to the web page that are responsible for loading pre-generated models into it.

3 AlgoView Toolset

3.1 The Builder Tool Description

The whole builder tool is written in C++ and is platform-independent. It, however, uses external libraries for certain auxiliary jobs and, thus, cannot be called a complete creation of ours. The RapidXML library [11] is used to make parsing the XML file into a DOM-tree easier. The Exprtk library [12] is used to evaluate expression strings contained in XML files. Our own source code is organized in several files as well.

The XML file name and output file names are used as input parameters. The builder tool works in several steps.

Initializing. First of all, objects of Para_parser and Builder classes are created. Input parameters are passed to these objects.

Parsing the DOM-tree. The Build_params method from Para_parser class does the heavy lifting here. The input XML file is, first of all, parsed into a DOM-tree by means of RapidXML. Then all the tree nodes are operated recursively. By the end of this stage, our Para_parser class object has two lists containing, respectively, the coordinates of the graph vertices and pairs of graph vertices (the first vertex is the beginning of an edge and the second is its end). We also plan to add information about loop variables for each vertex created, which can be later used for calculating exact edge-positions in 3-dimensional space, as well as for adding new interactive features for 3-dimensional representation that we do not have yet (see Sect. 4 for more details). The exact form of this information is still a matter of some concern.

Building 3D models in JSON format. The Build_JSON method from the Builder class takes pointers to both lists as parameters. All the list nodes are parsed in sequence, with 3D-model generation being called for each node. The Builder class has private methods used to create geometric primitives both for vertices and edges; these methods take coordinates stored in the list nodes as parameters.

3.2 The Displayer Tool Description

The output of the builder tool is, as we already know, a set of 3D-models, which are handled by the displayer tool to create a final representation. Each web page is used to display a single information graph and is supported by the mentioned set of Java scripts and source files. These include JS libraries, textures, auxiliary 3D-models (for example, axis models), Java scripts used to interact with the representation, etc. The whole set of files contains library files, auxiliary source files and actual elements of the displayer tool.

Displaying an information graph representation in a browser window is an easy task compared to building a set of 3D-models from an XML file. It is done in several steps.

Initializing. Whenever some client decides to open one of our web pages in his browser, the `InitGUI()` function is called, creating a simple GUI to interact with the information graph representation. The `webGLStart()` function creates the canvas used to display the representation; a GL type object is initialized, as well as shaders and textures; an appropriate set of 3D-models is loaded into memory; reaction to user actions is set up. Then the `tick()` function is called, so the system will work in real time.

Animating. The `tick()` function is responsible for drawing the current frame and calculating parameters for the next frame (both implemented in separate procedures: `DrawScene()` and `Animate()`).

3.3 Integrating the System into the AlgoWiki Project

This is a simple but important part. The problem is that the MediaWiki engine used by the AlgoWiki project does not support WebGL technology. So we had to figure out a method for integrating our web pages into AlgoWiki. Fortunately, the `IFrame` MediaWiki widget was designed for performing such actions. All we had to do was to install `IFrame` on AlgoWiki, and add the appropriate pages by its means.

4 Experiments and Results

Here we describe our achievements in creating 3-dimensional representations of information graphs. We have built some simple representations of hypothetic graphs for testing purposes, and moved on to creating actual representations of algorithms described in the AlgoWiki project. For now, automatic building does not support creating macro operations at all, since XML files do not contain that kind of information. Also, the aesthetic side is affected in certain cases. To fix that, the builder tool can also be run in a manual mode in which it creates vertices and edges according to commands given by the user. A basic representation displays the coordinate axes, followed by all the graph vertices and edges without paying attention to the exact nature of specific operations

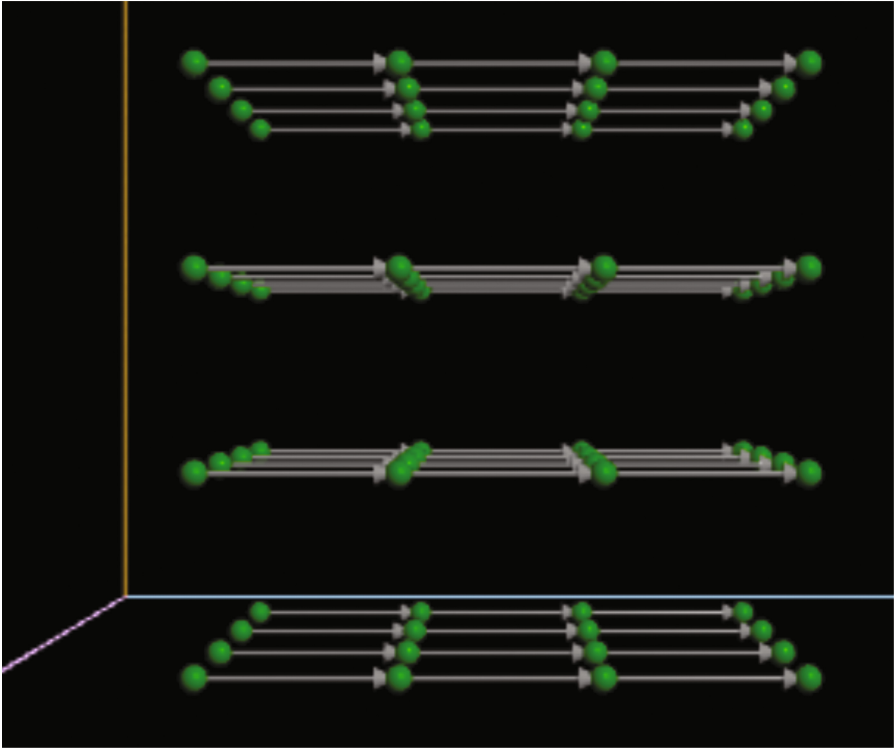


Fig. 2. Parallel matrix multiplication information graph, $n = 4$

and data flows within the information graph. In simple cases, this is sometimes not an issue, as shown in Fig. 2. Here a basic matrix multiplication is shown, all operations being ternary: $a + b * c$, where a is known from the previous operation in the same sequence, and b and c are elements of the input matrices. To address that, we plan to use different colors for different operations, and add auxiliary information about what each color means, like we did in the 2-dimensional representation. Another important idea one can deduce from the set of recommendations given for the information graph 2-dimensional representation was to display input data as vertices of different shapes. For the moment, we do not show them in the 3-dimensional representation.

An important part of our 3-dimensional representation concept is interactivity. That means several actions are available to users. Many “multidimensional” information graphs follow a set of structure patterns, different for each dimension. Thus, the most obvious feature is the adjustable point of view shown in Fig. 3 going along with some preset camera settings that can be used for displaying graph projections onto the coordinate planes, as one can see in Figs. 4 and 5.

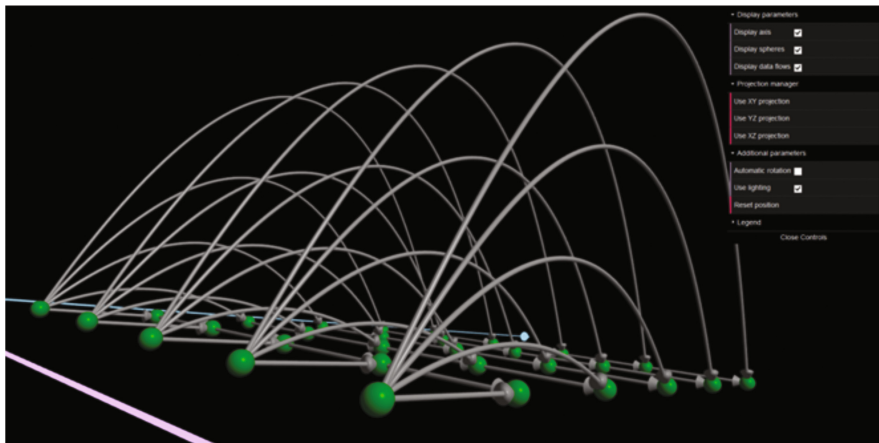


Fig. 3. Parallel matrix-vector multiplication information graph, $n = 5$; note the control panel in the upper right corner

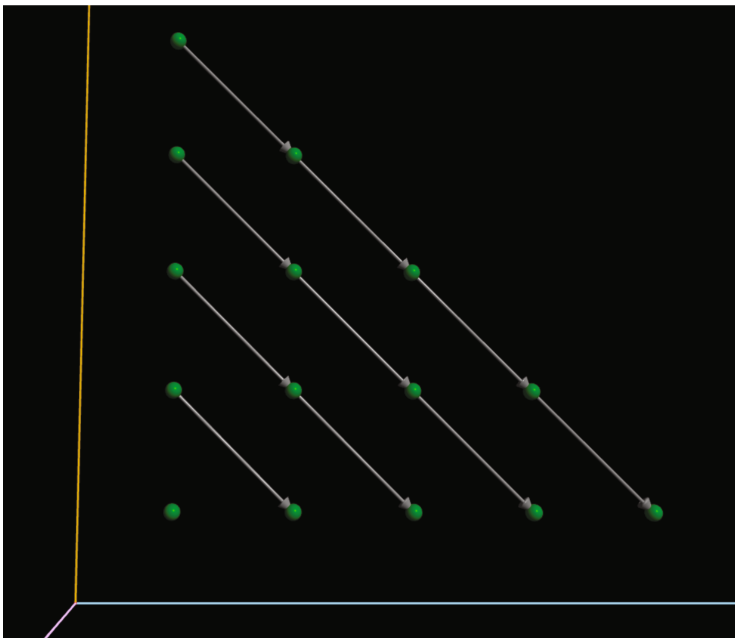


Fig. 4. Horner's scheme visualization, $n = 6$, XY projection

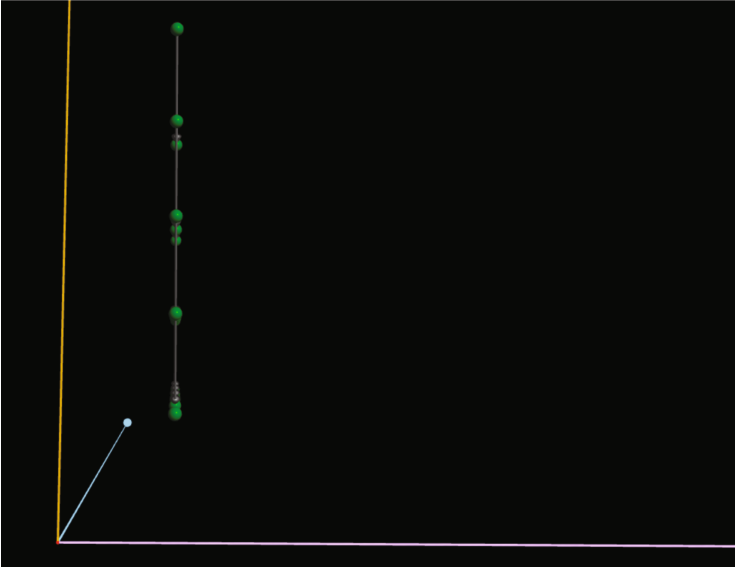


Fig. 5. Horner’s scheme visualization, $n = 6$, YZ projection

Another option that may come in handy is the possibility to choose which parts of a 3-dimensional representation are currently displayed, as it can be noticed in Fig. 2 as well. This approach can ease the understanding of complicated structures common within information graphs, when it comes to something more intricate than matrix-vector operations. Addressing the parallel form of the information graph is also important, and the plan is to allow users to highlight a chosen level inside the graph, as well as to “walk” through levels to see how the algorithm execution goes. Like all the other features that are not present in the current version of 3-dimensional representation, it needs additional information stored in the vertex list described in Sect. 3.

The biggest issue we came across in building up 3-dimensional graph representations is the possible interference of graph edges. Assume we have a very simple information graph of an algorithm that represents a simple loop with n iterations where every iteration, except the first one, uses data from both the previous iteration and the first. Such a loop would result in a line of vertices with edges going from the first vertex to each one of the others, and neighbor vertices connected as well. This basically means that the edges would cover each other in a 3-dimensional representation. To avoid such an issue, the Bezier curve technique previously mentioned in Sect. 2 was introduced. But this technique itself does not completely resolve possible issues, as one can see in Fig. 6.

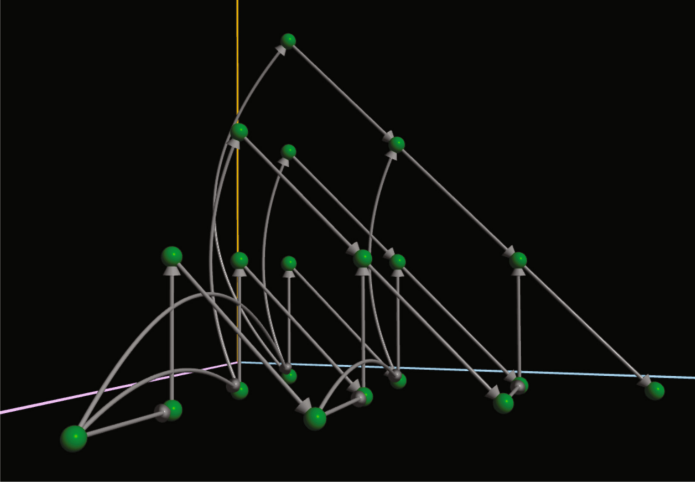


Fig. 6. Cholesky factorization information graph, $n = 4$

5 Conclusion

In this paper, we have reviewed our previously developed approach to the theoretical basis of creating 3-dimensional information graph representations. We have introduced the idea of a representation transformation sequence and have described a toolset implemented to automatically perform the two steps of the sequence that had not been worked out yet. Our toolset has been integrated into the AlgoWiki project, and some algorithm descriptions have been supplemented with our 3-dimensional representations. However, there is still work to do, since, as we have shown, our system sometimes does not work well in the aesthetic aspect, allowing data flows in the representation to interfere each other. Along with improvements to the tool responsible for transforming the source code into the inner representation, minimizing the number of such “difficult” algorithms stands as our primary goal. We have tested the system by creating 3D-representations of several sample algorithms described in AlgoWiki, and they have proved to be successful.

References

1. Voevodin, V.V., Voevodin, V.V.: Parallel Computing. BHV-Petersburg, St. Petersburg (2002). (in Russian)
2. Voevodin, V., Antonov, A., Dongarra, J.: Why is it hard to describe properties of algorithms? Proc. Comput. Sci. **101C**, 3–6 (2016). doi:[10.1016/j.procs.2016.11.002](https://doi.org/10.1016/j.procs.2016.11.002)
3. Alexander Antonov, A., Voevodin, V., Voevodin, V., Teplov, A.: A study of the dynamic characteristics of software implementation as an essential part for a universal description of algorithm properties. In: 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing Proceedings, pp. 359–363 (2016). doi:[10.1109/PDP.2016.24](https://doi.org/10.1109/PDP.2016.24)

4. Open Encyclopedia of Parallel Algorithmic Features AlgoWiki. <https://algowiki-project.org/>
5. Sharp, J.A. (ed.): Data Flow Computing: Theory and Practice. Ablex Publishing Corp, Norwood (1992)
6. The Open Graph Viz Platform. <https://gephi.org/>
7. Graph Visualization Software. <http://www.graphviz.org/>
8. An Open-Source web-application. <http://molview.org/>
9. Voevodin, V., Antonov, A., Dongarra, J.: AlgoWiki: an open encyclopedia of parallel algorithmic features. Supercomput. Front. Innov. **2**(1), 4–18 (2015). doi:[10.14529/jsfi150101](https://doi.org/10.14529/jsfi150101)
10. Chong Lip Phang: Web Coding Bible (HTML, CSS, Javascript, PHP, SQL, XML, SVG, Canvas, WebGL et al.)
11. A RapidXML manual. <http://rapidxml.sourceforge.net/manual.html>
12. Exprtk manual. <http://www.partow.net/programming/exprtk/>
13. Antonov, A., Frolov, A., Kobayashi, H., Konshin, I., Teplov, A., Voevodin, V., Voevodin, V.: Parallel processing model for Cholesky decomposition algorithm in AlgoWiki project. Supercomput. Front. Innov. **3**(3), 61–70 (2016). doi:[10.14529/jsfi160307](https://doi.org/10.14529/jsfi160307)