

Procedia Computer Science

Procedia Computer Science 101, 2016, Pages 4 – 7

YSC 2016. 5th International Young Scientist Conference on Computational Science

Why is it hard to describe properties of algorithms?

Vladimir Voevodin[1], Alexander Antonov[1], and Jack Dongarra[1]

Moscow State University, Moscow, Russia
info@algowiki-project.org

Abstract

Everyone knows what parallel algorithms are. However, as you start describing properties of real algorithms, you realize that creating a complete description of an algorithm is not a challenge, it is a large series of challenges! In the paper we will try to explain our position.

Keywords: supercomputer, parallel computing, algorithm properties, information structure, dynamic characteristics, AlgoWiki

Each new computing platform (vector, distributed memory, manycore, GPU-based, etc. [1]) requires software developers to analyze algorithms over and over, each time having to answer the same two questions. Does the algorithm possess necessary properties to meet the architectural requirements? How can the algorithm be converted so that the necessary properties can be easily reflected in parallel programs? In spite of the basic principle: *changes in computer architecture do not change algorithms*, this analysis had to be performed again and again when a program was ported from one generation of computers to another, largely repeating the work that had been done previously.

This begs a natural question: is it possible to do the analysis "once and for all," describing all of the key properties of an algorithm so that all of the necessary information can be gleaned from this description any time a new architecture appears? As simple as the question sounds, answering it raises a series of other hard questions. What does it mean "to perform analysis" and what exactly needs to be studied? What kind of "key" properties need to be found in algorithms to ensure their efficient implementation in the future? What form can (or should) the analysis results take? What makes a description of algorithm properties really "complete"? How does one guarantee that a description is complete and that all of the relevant information for any computer architecture is included?

The questions are indeed numerous and non-trivial. Obviously, a complete description needs to reflect many ideas: computational kernels, determinacy, information graphs, communication profiles, a mathematical description of the algorithm, performance, efficiency, computational intensity, the parallelism resource, serial complexity, parallel complexity, etc. [2].

Many of the notions mentioned above are very well known. However, as you start describing the properties of real algorithms, you realize that *creating a complete description of an algorithm* is not a challenge, it is *a large series of challenges*! Unexpected problems arise at each step, and a seemingly simple action becomes a stumbling block. Let us look at several practical examples which are briefly outlined below.

Peer-review under responsibility of organizing committee of the scientific committee of the 5th International Young Scientist Conference on Computational Science

doi:10.1016/j.procs.2016.11.002

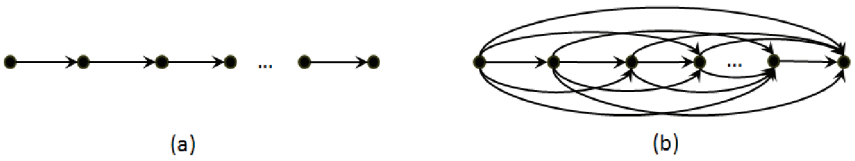

Figure 1: Transitive closure of the linear graph: quadratic growth of the output data

Try to perceive structure of an algorithm. Classical straightforward matrix multiplication is a simple fragment of five statements: three nested loops (i, j, k), zeroing of an element, and its calculation. Let us assume that we shuffle the loops to get other order. Will we keep the result of matrix multiplication the same for the new order? The answer is "yes". At least two questions arise... Why the result is the same for any order of loops in this fragment? What did we decide to change loops location for? An answer for the first question is inside information structure of the algorithm. Considerations behind the second question are that different orders can bring 5–6-fold speedup for the majority of modern processors.

Simple details matter. Amount of input and output data of an algorithm is a quite evident parameter, so it could be easily omitted or forgotten. If you try to compute a transitive closure of a graph with n vertices then in some cases you will have to deal with $O(n^2)$ arcs in a result (fig. 1). It is not a problem for small graphs but if you try to apply the similar algorithm for analysis of a social network with billions of vertices you will simply fail to store all the output data.

Input and output data and computational intensity of an algorithm. There is another reason why we need to think about amount of input and output data, it is computational intensity of an algorithm. If computational intensity is small it could be costly to use accelerators due to large overheads on data transfers. Sum of two vectors, dot product, and block matrix multiplication are examples of algorithms with different computational intensity and therefore with different perspectives regarding target computing platforms.

Serial complexity. Criteria changes. We got used to measure complexity of algorithms as a total number of additions, multiplications, divisions, square roots and other arithmetic operations. Today data movements are often more expensive. Read/write operations affect execution time and undoubtedly they should be seriously considered if we try analyzing power consumption issues.

Information structure. If you have detected information structure of an algorithm then a half of its parallel implementation has done. But how to detect, describe, and express information structure? Information structure can be represented as a graph but this graph is potentially unlimited and multidimensional object. How to present it in a form that is easy to perceive? Even if you know the information structure of an algorithm you need to find a method to explain or show a way of its parallel execution. Look at the information graph on the fig. 2, its structure looks quite intricate. At the same time, more careful analysis of its projections to the coordinate planes reveals large potential parallelism.

Mathematics and parallelism. We can't rely only on classical information structure of an algorithm: to compose the complete description of the algorithm we have to analyze a problem and mathematics behind it. Let us consider a minimum spanning tree problem and Boruvka's algorithm for its solving. Using the simple mathematical equation you easily increase resource of parallelism which can be used very effectively for various parallel computing platforms. This is especially important for processing of extra large graphs.