Abstraction of synchronous computational model for parallel graph processing

Oleksandr Romaniuk[1], Tetyana Koroteeva[2]

[1]Software Department, Lviv Polytechnic National University,
UKRAINE, Lviv, S. Bandery street 28-a,
E-mail: Alexandr.Romanyuk@gmail.com

[2]Software Department, Lviv Polytechnic National University,
UKRAINE, Lviv, S. Bandery street 28-a,
E-mail: maxim04041997@gmail.com

Abstract – The analysis of current approaches to the development of algorithms for parallel and distributed computing on data graphs. The proposed technique facilitates the writing of parallel algorithms based on graphs by usage of the Actor and the Valiant's Bulk-Synchronous Parallel models. This approach differs by determinism of algorithms, by resilience to different faults and by higher level of abstraction that makes a developer free from implementation details introduced by parallel and concurrent programming primitives.

Key words – high dimensional graphs, parallel algorithms, resilient arhitecture, Actor model, Bulk Synchronous Parallel model.

I. Introduction

Algorithms based on graphs are important for solving of many problems in scientific computing, datamining, they are a part of artificial intelligence. Applications of such algorithms are routing, robotics, software verification techniques, proofs, biology and others.

Dimensions of solution spaces increase and parallel computing becomes an integral part of requirements for implementation of computational complexity and memory constraints. Unfortunately, algorithms and software that were developed for parallel scientific applications aren't necessarily effective for high dimensional graphs.

One of the important issues of parallel algorithms is a great difficulty in implementation of their logic. Even more difficult is to find bugs in these implementations by other developers. The cause of problem is a lack of sufficient level of code abstractions that is available for developers of parallel algorithms. Because of mentioned problems developers have to think beyond the logic of algorithm. Therefore, improvements of algorithms on graphs that increase level of code abstraction, run-time computation and a memory are quite important for modern scientific research.

The aim of this work is to separate parallel algorithm developers from the nitty-gritty details such as: thread synchronization concerns; data races; deadlocks; asynchronous message exchange between nodes; clustering of graphs by providing high-level data model and functions to calculate; increase flexibility of communications among parallel processes (vertices of a search graph); reduce non-determinism in state of parallel computations.

The subject of the research is the high level of abstraction to facilitate an implementation of algorithms based on graphs with horizontal and vertical scalings, that are transparent for developer.

II. Computational model

In this work the abstract model of parallel computations is proposed, which consists of three parts - a graph, an update function and a mechanism for synchronization of vertices. The graph displays a state of a program, stores variable data provided by user and displays computational dependencies. The update function added to every vertex determines local computations and operates with data graph in small areas that overlaps. The synchronization mechanism performs global aggregative functions that cover part or all data graph.

The data model consists of a data graph, a container that manages user defined data. All is processed and represented as graph. Graph mirrors the computational structure that is specific to a particular problem and state of a program that is being executed. The data model is independent of edge directions. Thus, there is the opportunity to associate arbitrary parameters with each vertex and edge of the graph. Together with the data, the graph stores the value to be updated during the iterative process. Thus, the algorithm is implemented by creation of a set of rules according to which, a value of each node may be changed over time. Data updates in vertices executes at synchronous "super steps" that are similar to the Valiant's model of Bulk Synchronous Parallel [1, 2] computations. Super step is a computational iteration, which consists of executing updates for all vertices (defined locally for each vertex separately), and of a synchronization function. At a function's output an updated list of messages alerts for incoming changes of an vertices internal state, which is addressed to a neighbour vertices. These reports are available for recipient vertices during start of a next super step. The result of aggregative function is available for all vertices through incoming messages, which are available at the beginning of each subsequent super step.

Calculations from the model that proposed at this work are presented as functions for status updates of vertices [3, 4, 5]. Update function is a representation of a main element of calculation algorithm and it's operating data within neighboring nodes. Each update function is determined by a developer of an algorithm for each vertex separately. Update function planes it's next update operations (on adjacent vertices) using asynchronous messaging.

Adjacent vertices of the graph define the limit to which the update feature is available. Update function takes vertex with its boundary as input and returns new versions of data packed into a set of boundaries and vertices. After executing update modified data is written back to the graph. A set of vertices computed by the asynchronous updates. Functions are updated to effectively express adaptive computing through control over the data that returns a set of vertices in further calculations. For example, update function is able to put into an execution plan the return of data only in case of significant changes done by it neighbor's vertices.

The synchronization mechanism that aggregates data among all nodes that defined in the graph works on the basis of folding and reduction functions used in the paradigm of functional programming. When synchronization mechanism is started, the algorithm uses the folding function for serial data aggregation among all nodes. Combining function allows to make parallel reduction tree that is used for combination of results from several parallel fold functions.

Update function finalizes the end result before it is written back to the distributed memory.

Suggested execution model allows to define arbitrary update functions to read and modify any data on the adjacent vertices and edges. This approach aims to improve the performance of parallel algorithms due to the lack of excessive synchronization and fewer switches of an execution contexts [2, 3]. Also it simplifies the process of writing parallel algorithms by eliminating the need of making a low level design (e.g. async messaging, synchronization, clustering, scaling, etc.).

III. Results

Performance of implementation of two algorithms based on proposed parallel synchronous model was evaluated. Algorithms that selected for evaluation were the PageRank, and the Single source Shortest Path (SSSP).

Results of performance testing were been compared to the analogous implementations of aforementioned algorithms based on the MapReduce paradigm (on the Apache Hadoop platform).

Results were conducted on the server based on the Intel Xeon processor: 3.1 GHz, 8 physical cores and hyper-threading, 32 GB RAM. Used software configuration: OS Debian Linux 7.1 64bit, HotSpot JVM 7u40 64bit (27 GB of HeapSpace was used), Apache Hadoop 2.1.

PageRank - a popular technique for analyzing the relative "importance" of webpages based on the hyperlink structure. The PageRank value of page depends on the values of pages those links to the source page, equation of PageRank function iteratively applies until the value of each page converges [6].

Single source Shortest Path algorithm solves problem of finding the shortest path from the source vertex to every other vertex in the graph. The problem solved by finding a shortest path tree rooted at source that contains all the desired shortest paths.

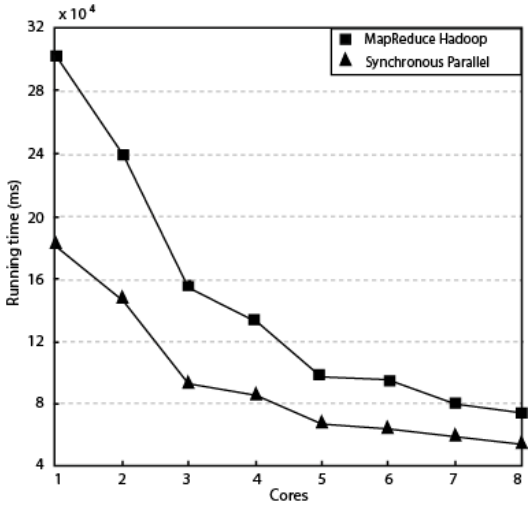

Fig. 1. Running times of the PageRank

Preliminary testisting showed next improvements in the speedup on 8 cores: the PageRank at 18.75% (see Figs. 1, 2) and the SSSP at 22.5% (see Fig. 3) comparing to the MapReduce implementations.

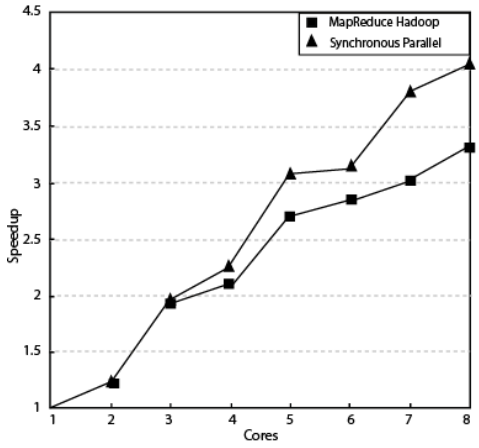

Fig. 2. Speedups of the Pagerank

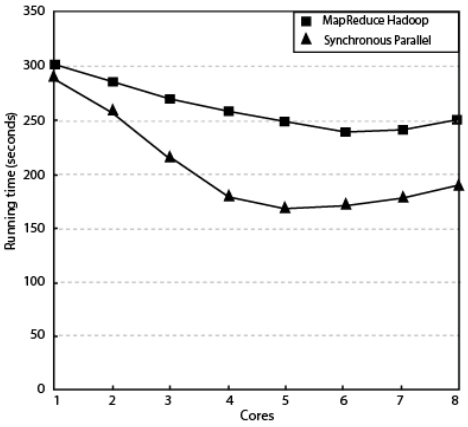

Fig. 3. Running times of the SSSP

Conclusion

Parallelizing algorithms is challenging but higher level of the abstraction like one that was proposed makes it much easier. We could see that parallel algorithms could be implemented with less efforts and with better scalability.

References

[1] L. G. Valiant et al., "Bulk Synchronous Parallel Computing. A Paradigm for Transportable Software", in *Proc. Hawaii international Conf. on system sciences*, Cambridge, MA, 1995, pp. 95-108.

[2] D. P. Krizanc and A. A. Saarimaki, "Bulk Synchronous Parallel: practical experience with a model for parallel computing", in *Parallel Architectures and Compilation Techniques Conf.*, Canada, 1996, pp. 636-640.

[3] G. H. Malewicz et al., "Pregel: a system for large-scale graph processing", in *Proc. 28th ACM Symp. Principles of distributed computing*, New York, NY, 2009, p. 6.

[4] Y. J. Low et al., "Graphlab: A new framework for parallel machine learning", in *Conf. on Uncertainty in Artificial Intelligence*, Catalina Island, CA, 2010, pp. 31-43.

[5] P. B. Haller and H. J. Miller, "Parallelizing machine learning-functionally: A framework and abstractions for parallel graph processing", In *The 2nd Annu. Scala Workshop*, Stanford, CA, 2011, pp. 15-64.

[6] S. Y. Wills, "Google's PageRank: The Math Behind the Search Engine", *Mathematical Intelligencer*, vol. 28, no. 4, p. 16, Apr. 2006.