A Hierarchical Approach to Reducing Communication in Parallel Graph Algorithms

Harshvardhan Nancy M. Amato Lawrence Rauchwerger

Parasol Laboratory
Department of Computer Science and Engineering
Texas A&M University
{ananvay, amato, rwerger}@cse.tamu.edu

Abstract

Large-scale graph computing has become critical due to the ever-increasing size of data. However, distributed graph computations are limited in their scalability and performance due to the heavy communication inherent in such computations. This is exacerbated in scale-free networks, such as social and web graphs, which contain *hub vertices* that have large degrees and therefore send a large number of messages over the network. Furthermore, many graph algorithms and computations send the same data to each of the neighbors of a vertex. Our proposed approach recognizes this, and reduces communication performed by the algorithm without change to user-code, through a hierarchical machine model imposed upon the input graph. The hierarchical model takes advantage of locale information of the neighboring vertices to reduce communication, both in message volume and total number of bytes sent. It is also able to better exploit the machine hierarchy to further reduce the communication costs, by aggregating traffic between different levels of the machine hierarchy. Results of an implementation in the STAPL GL shows improved scalability and performance over the traditional level-synchronous approach, with $2.5\times - 8\times$ improvement for a variety of graph algorithms at $12,000+$ cores.

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Concurrent Programming—Parallel Programming; D.2.13 [*Software Engineering*]: Reusable Software—Reusable Libraries

Keywords Parallel Graph Processing, Graph Analytics; Big Data; Distributed Computing.

1. Techniques

Our approach allows users to write fine-grained vertex-centric algorithms in order to expose the maximum parallelism and abstract the user from details of parallelism, while taking advantage of the machine hierarchy and locality information to optimize communication and improve scalability and performance. The graph algorithms do not change, allowing the reuse of existing vertex-centric algorithms.

PPoPP'15, February 7–11, 2015, San Francisco, CA, USA
ACM 978-1-4503-3205-7/15/02
http://dx.doi.org/10.1145/2688500.2700994

A graph hierarchy is formed by imposing multiple levels of the machine hierarchy on the input graph. For each level, all vertices on the same locale, along with the edges between them are coarsened into a single *super-vertex* representing the underlying sub-graph, and all edges between locales are represented by a single *super-edge*, representing the communication between the locales.

For a large class of graph algorithms, including breadth-first search, PageRank, k-core decomposition, connected components, topological sort, betweenness centrality, community detection, graph coloring, etc., the vertex sends the same information to all its neighbors. In such cases, using a hierarchical approach allows the communication to be reduced by sending a single update per location (processor, node, etc.) and applying it to all local vertices on the receiving location.

1.1 Implementation

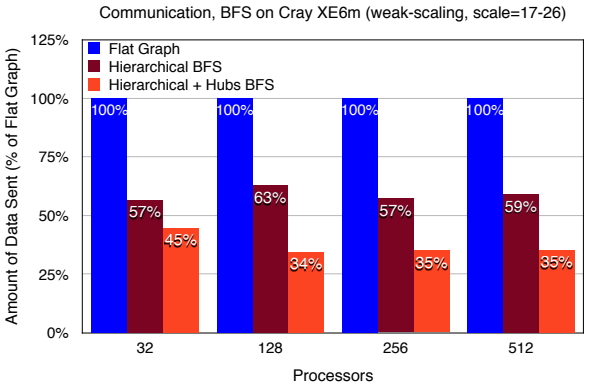
Our approach is implemented in the STAPL framework [2], by extending the STAPL Graph Library (STAPL GL) [3, 4], a distributed-memory graph library. STAPL GL expresses algorithms using two operators [4] – a vertex operator that performs computation, and a neighbor operator that updates neighboring vertices with the result of that computation. The two operators are executed on the input graph by the *graph paradigm*, which controls the execution of the algorithm. We added a hierarchical level-synchronous paradigm to take advantage of the machine hierarchy. The API of STAPL GL did not change as our technique is implemented at the paradigm level, and is therefore transparent to the user. This also allowed the use of existing STAPL GL algorithms *without modification*.

We create a hierarchy for the input graph based on our knowledge of the machine hierarchy and the data-distribution information of the input graph. The hierarchical graph is then passed to the paradigm, which only needs to send the result of a vertex's computation once for every *super-edge* instead of once for every edge.

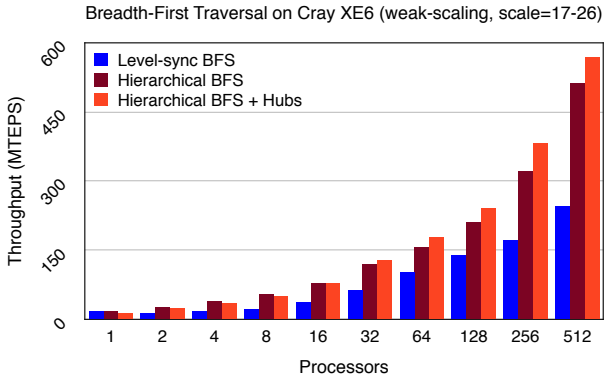
We also create a hierarchical level specialized for hub-vertices, which reduces all messages to a hub-vertex from the same location to a single value, which is then communicated to the hub. The hub-vertex can then reduce the incoming values to a single value.

2. Results

We evaluated the Graph500 benchmark application [1] that simulates data-intensive HPC workloads. The benchmark performs breadth-first traversals of the input graph from multiple source vertices. The Graph 500 input graph simulates social networks and web-graphs and exhibits small-world scale-free behavior, i.e., it has a short diameter (< 16 hops) and vertices with very high out-degrees. These high out-degree vertices (hubs) cause scalability bottlenecks due to communication.



(a)



(b)

Figure 1. Communication reduction using hierarchical approach and (b) Scalability (Throughput) of BFS on Graph500 benchmark.

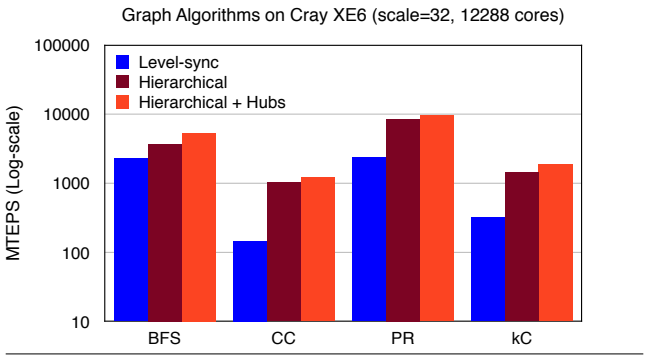


Figure 2. Scalability (Throughput, log-scale) of various algorithms using hierarchical approach. Graph500 input graph with **4 billion vertices and 64 billion edges** at **12,288 cores**.

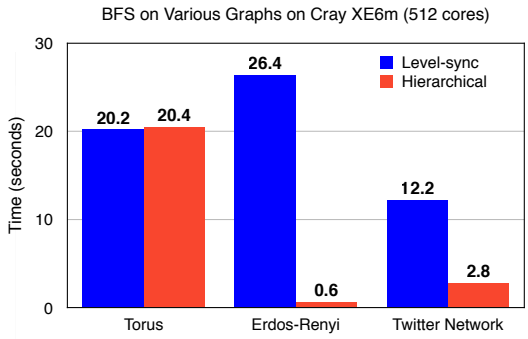


Figure 3. Running times of BFS on various input graphs using flat and hierarchical approaches.

Figure 1(a) compares the number of bytes communicated in the base-line level-synchronous approach with that in our hierarchical approach. As can be observed, the hierarchical approach is able to significantly reduce the number of bytes sent over the network by $2.7\times - 3.3\times$. This directly translates to the performance of the algorithm (Figure 1(b)), where we observe around a $1.8\times - 2.1\times$ improvement in performance over the base level-synchronous algorithm. For example, at 512 cores, BFS on the Graph 500 input graph communicated (sent/received) 33.87 GB of data across the system in the base (flat) case. This was reduced to 12.56 GB for our hierarchical approach and then further to 10.3 GB for the hierarchical with hubs approach. With the network sending $\frac{1}{3}^{rd}$ the data, we observe an improvement in performance.

Our approach also naturally scales to larger machines and other algorithms, as shown in Figure 2, which shows a $2.35\times$ performance improvement for breadth-first search, a $7.26\times - 8.54\times$ improvement for connected components, a $3.6\times - 3.95\times$ improvement for PageRank, and a $4.43\times - 5.84\times$ improvement in k-core decomposition. We attribute these improvements to lower communication and better load-balance provided by the hierarchical approach, which improves scalability, as we will demonstrate in this section on $12,288$ cores on a Cray XE6 machine.

Furthermore, as shown in Figure 3, the performance improvement provided by the hierarchy is highly dependent on the graph structure, where denser graphs observe a larger benefit. However, even for the worst-case, the performance is on par with the non-hierarchical algorithm.

Acknowledgments

We would like to thank our anonymous reviewers. This research is supported in part by NSF awards CCF 0702765, CNS-0551685, CCF-0833199, CCF-1439145, CCF-1423111, CCF-0830753, IIS-0917266, by DOE awards DE-AC02-06CH11357, DE-NA0002376, B575363, by Samsung, IBM, Intel, and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST). This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] The graph 500 list. http://www.graph500.org, 2013.

[2] A. Buss, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, G. Tanase, N. Thomas, X. Xu, M. Bianco, N. M. Amato, and L. Rauchwerger. STAPL: Standard Template Adaptive Parallel Library. In *Proc. Annual Haifa Experimental Systems Conference (SYSTOR)*, pp. 1–10, New York, NY, USA, 2010.

[3] Harshvardhan, A. Fidel, N. M. Amato, and L. Rauchwerger. The STAPL Parallel Graph Library. In *Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Science, pp 46–60, Springer, 2012.

[4] Harshvardhan, A. Fidel, N. M. Amato, and L. Rauchwerger. Kla: A new algorithmic paradigm for parallel graph computations. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques*, PACT '14, pp. 27–38, USA, 2014. ACM.