Personal reviews

José Fuentes Sepúlveda

May 15, 2017

This document contains personal reviews of publications on parallel algorithms. All the reviews are one-page long (**mandatory**). In the vast world of parallel algorithms, I am interested on *multicore algorithms for SMP systems*, *thread-safe data structures* and *(in the near future) GPU algorithms.*

I would be glad if these reviews can help more people (not just myself). Feel free to use or edit it. If you find a mistake, do not hesitate to send me an email at `jfuentess@dcc.uchile.cl` or `jfuentess@udec.cl`.

Note: The first review was written in 20th April of 2017.

§1 A Simple and Practical Linear-work Parallel Algorithm for Connectivity [7]

Authors: Julian Shun, Laxman Dhulipala and Guy Blelloch
URL: `http://doi.acm.org/10.1145/2612669.2612692`
PDF: `https://people.eecs.berkeley.edu/~jshun/connectedComponents.pdf`
Implementation: `https://people.eecs.berkeley.edu/~jshun/connectedComponents.tar`

This work introduced a *simple* linear-work algorithm to solve the problem of finding the connected components of an undirected graph. Let $G = (V, E)$ be an undirected graph, with $|V| = n$ and $|E| = m$. The introduced algorithm returns a labeling L of G such that for vertices $u, v \in V$, $L(u) = L(v)$ iff u and v belong to the same component, and $L(u) \neq L(v)$ otherwise. Additionally, the authors give an implementation using Cilk Plus which is competitive with respect to non-linear implementations (does not overcome them).

The proposed algorithm is based on the parallel algorithm of Miller et al. [2] for the generation of a low-diameter decomposition of a graph. For each vertex $v \in V$, the algorithm of Miller et al. assigns a δ_v value drawn from an *exponential distribution* with parameter β, $0 < \beta < 1$. The maximum δ value is $O(\lg n/\beta)$, w.h.p. Then, multiple BFS's are performed in parallel, assigning the vertex v to the partition S_u that minimizes the distance $d_\delta(u, v) = d(u, v) - \delta_u$ (S_u corresponds to the labeling of the partition). The decomposition algorithm iterates $O(\lg n/beta)$ times, generating partitions with diameter $O(\lg n/\beta)$ and the number of edges between partitions is $O(\beta m)$. The algorithm runs in $O(m)$ work and $O(\lg^2 n/\beta)$ depth w.h.p. in the CRCW PRAM model.

The connectivity algorithm of Shun et al. calls recursively the Miller et al. algorithm. On each call, the decomposition algorithm returns two output: a labeling of the vertices of G and the traversal of all the BFS's. The vertices in the traversal of the BFS's are stored in a *frontier array*, where vertices belonging to the same partition are in consecutive positions in the frontier. Since the decomposition algorithm iterates $O(\lg n/\beta)$ times, there are $O(\lg n/\beta)$ frontiers w.h.p. Additionally, for each BFS, the position of the partition's vertices of each frontier and the total number of edges for these vertices are stored. During the BFS's, all the intra-partition edges (ending vertices belong to the same partition) are marked to be deleted. With that information, by performing parallel prefix sums, the graph G is contracted in graph $G' = (V', E')$. All the vertices belonging to a partition are contracted into a single vertex, the intra-partition edges are packed out and the inter-partition edges are relabeled using the contracted vertices. On each call, the number of edges G' decreases to at most βm, with respect to G. This process is repetead until G' is completely disconnected ($|E'| = 0$). Thus, the total number of calls is $O(\lg_{1/\beta} m)$ w.h.p. After reaching the last call, the contracted vertices are expanded, relabelling the vertices inside it with the same label, until obtaining the original graph G. The complexity of the proposed algorithm is $O(m)$ work and $O(\lg^3 n)$ w.h.p., dominated by the $O(\lg_{1/\beta} m)$ calls of the decomposition algorithm with $O(\lg^2 n/\beta)$ depth ($O(\lg_{1/\beta} m \lg^2 n/\beta) = O(\lg^3 n)$).

Note: It is possible to modify the Shun et al. algorithm to obtain a *non-rooted spanning tree* of an undirected graph in parallel. To do that, we need to stored the spanning tree generated by each BFS. The independent spanning trees of the BFS's are merged in the next recusive call, generating fewer and larger spanning trees. After the last recursive call, all the spanning trees are merged into the final spanning tree. To merge the spanning trees, we need to store the original reference of each edge, since the original algorithm of Shun et al. relabels the edges on each recursive call.

§2 Parallel Graph Decompositions Using Random Shifts [2]

Authors: Gary Miller, Richard Peng and Shen Chen Xu
URL: `http://doi.acm.org/10.1145/2486159.2486180`
PDF: `https://arxiv.org/abs/1307.3692`

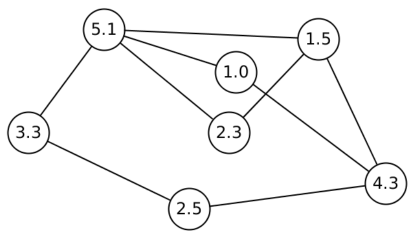
In this work the authors present a CRCW-PRAM algorithm for low-diameter graph decomposition. Given an unweighted graph $G = (V, E)$, with $|V| = n$ vertices, $|E| = m$ edges and a parameter $\beta \leq 1/2$, the algorithm generates a graph decomposition where each partition has diameter at most $O(\lg n/\beta)$ and the number of edges with endpoints belonging to different partitions is at most βm.

The algorithm works as follows:

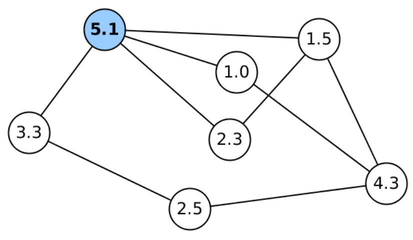
1. For each vertex v, select a δ_v value drawn from an exponential distribution with parameter β. The authors show that the maximum δ_v value, δ_{max}, is $O(\lg n/\beta)$ w.h.p.

2. Let S_u be a partition centered on vertex u. The algorithm assigns a vertex v to S_u if the distance $dist_{-\delta}(u, v) = dist(u, v) - \delta_u$ is minimized. This partitioning can be implemented by using parallel BFS's. We can interpret $\delta_{max} - \delta_v$ as the time when vertex v can start growing a partition or cluster. At iteration t (starting with $t = 0$), all the *unvisited* vertices with $\delta_{max} - \delta_v \leq t$ can start its own BFS traversal (and its own partition). On each new iteration, all the already existing partitions try to insert their unvisited neighbors to their partitions. If a vertex v is reached by two growing partitions, then the algorithm decides the owner by considering the fractional part of the δ value of the centers of both partitions. Alternatively, the decision can be taken by using a random permutation of the vertices. Thus, after $O(\lg n/\beta)$ iterations, all the vertices belong to one and only one partition.

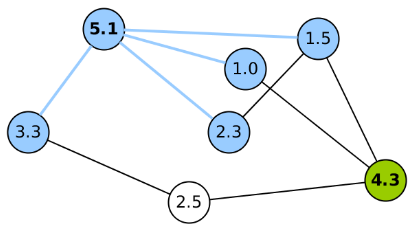
Using the results of [1], each BFS takes $O(m)$ work and $O(\Delta \lg n)$ depth, where Δ is the diameter of the graph. For the partitioning algorithm, $\Delta = O(\lg n/\beta)$. Thus, the partitioning algorithm takes $O(m)$ work and $O(\lg^2 n/\beta)$ depth, w.h.p. As an example:



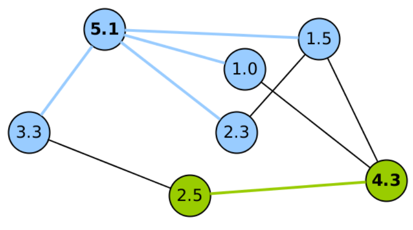
Assigning δ values



t=0



t=1



t=3

§3 Reducing Contention Through Priority Updates [6]

Authors: Julian Shun, Guy Blelloch, Jeremy Finneman and Phillip Gibbons
URL: `http://doi.acm.org/10.1145/2517327.2442554`
PDF: `https://people.eecs.berkeley.edu/~jshun/contention.pdf`

Under construction

§4 A Survey of Techniques for Cache Partitioning in Multicore Processors [3]

Authors: Sparsh Mittal

URL: `http://doi.acm.org/10.1145/3062394`

PDF: `https://www.researchgate.net/publication/314352438_A_Survey_of_Techniques_for_Cache_Partitioning_in_Multicore_Processors`

Under construction

§5 A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures [4]

Authors: Cristbal A. Navarro, Nancy Hitschfeld-Kahler and Luis Mateu
URL: `https://doi.org/10.4208/cicp.110113.010813a`
PDF: `http://www.global-sci.com/openaccess/v15_285.pdf`

Under construction

§6 Ligra: A Lightweight Graph Processing Framework for Shared Memory [5]

Authors: Julian Shun and Guy E. Blelloch
URL: `http://doi.acm.org/10.1145/2442516.2442530`
PDF: `https://www.cs.cmu.edu/~jshun/ligra.pdf`
Implementation: `http://jshun.github.io/ligra/index.html`

Under construction

References

[1] Philip N Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 25(2):205 – 220, 1997.

[2] Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 196–203, New York, NY, USA, 2013. ACM.

[3] Sparsh Mittal. A survey of techniques for cache partitioning in multicore processors. *ACM Comput. Surv.*, 50(2):27:1–27:39, May 2017.

[4] Cristbal A. Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics*, 15(2):285329, 2014.

[5] Julian Shun and Guy E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pages 135–146, New York, NY, USA, 2013. ACM.

[6] Julian Shun, Guy E. Blelloch, Jeremy T. Fineman, and Phillip B. Gibbons. Reducing contention through priority updates. *SIGPLAN Not.*, 48(8):299–300, February 2013.

[7] Julian Shun, Laxman Dhulipala, and Guy Blelloch. A simple and practical linear-work parallel algorithm for connectivity. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 143–153, New York, NY, USA, 2014. ACM.