Brief Announcement: PASGAL: Parallel And Scalable Graph Algorithm Library

Xiaojun Dong
UC Riverside
xdong038@ucr.edu

Yan Gu
UC Riverside
ygu@cs.ucr.edu

Yihan Sun
UC Riverside
yihans@cs.ucr.edu

Letong Wang
UC Riverside
lwang323@ucr.edu

ABSTRACT

We introduce PASGAL (Parallel And Scalable Graph Algorithm Library), a parallel graph library that scales to a variety of graph types, many processors, and large graphs. One special focus of PASGAL is the efficiency on *large-diameter graphs*, which is a common challenge for many existing parallel graph processing systems due to the high overhead in synchronizing threads when traversing the graph in the breadth-first order.

The core idea in PASGAL is a technique called *vertical granularity control* (VGC) to hide synchronization overhead by careful algorithm redesign and new data structures. We compare PASGAL with existing parallel implementations on several fundamental graph problems. PASGAL is always competitive on small-diameter graphs, and is significantly faster on large-diameter graphs.

CCS CONCEPTS

• **Theory of computation** → **Graph algorithms analysis**; **Parallel algorithms**; **Shared memory algorithms**.

KEYWORDS

Parallel Algorithms, Graph Algorithms, Graph Processing

ACM Reference Format:
Xiaojun Dong, Yan Gu, Yihan Sun, and Letong Wang. 2024. Brief Announcement: PASGAL: Parallel And Scalable Graph Algorithm Library. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '24), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3626183.3660258

1 INTRODUCTION

Processing and analyzing large-scale graphs in parallel have become increasingly important. The increasing number of cores and memory size allows a single machine to easily process graphs with billions of vertices in a few seconds, even for reasonably complicated tasks. As a result, a huge number of in-memory graph processing algorithms and systems have been developed (e.g. [2, 4]).

Despite the hardware advances, the increasing number of cores does not provide "free" performance improvement. We observed that many existing parallel systems suffer from scalability issues, even in fundamental tasks such as breadth-first search (BFS), strongly connected components (SCC), biconnected components (BCC), and single-source shortest paths (SSSP). In Fig. 1, we compare the running time of some existing parallel implementations over a sequential one for each problem. Tested on a 96-core machine, existing parallel implementations can perform worse than a sequential one, especially on large-diameter graphs.

One major reason for such performance degeneration is the high overhead of managing and synchronizing threads. While enabling the potential of better parallelism, more cores also bring up great challenges and overhead. This is more pronounced when using BFS-like primitives on large-diameter graphs: when traversing the graph in BFS order, the number of rounds (and thus the cost of synchronizing threads between them) is proportional to the diameter of the graph. As a result, when the diameter is large, the synchronization overhead can be more expensive than the computation.

We propose an open-source library ***PASGAL***: the ***Parallel And Scalable Graph Algorithm Library***, that implements various graph algorithms scalable to diverse graph types, many processors, and large graphs. PASGAL includes several problems where existing parallel solutions suffer from high synchronization costs, such as BFS, SCC, BCC, and SSSP. We plan to include more in the future.

To overcome the scalability issues, the key technique in PASGAL is called *vertical granularity control (VGC)* proposed in our recent paper [13] to hide scheduling overhead. Accordingly, we need to redesign algorithms and data structures to facilitate VGC, as well as to synergistically optimize work, span and/or space usage. In Sec. 2, we introduce our techniques in more details.

With VGC and other techniques, PASGAL achieves high performance on a variety of graphs, especially large-diameter graphs. We present some experimental results in Fig. 1. Compared to the baselines, PASGAL is always competitive on small-diameter graphs, and is almost always the fastest on large diameter graphs. Our code is publicly available [5]. Due to space limit, we put more references and experiments in the full version [6].

Preliminaries. Given a graph $G = (V, E)$, we denote $n = |V|$ and $m = |E|$. We use D to denote the diameter of G.

Most algorithms in PASGAL are ***frontier-based***. At a high level, the algorithm maintains a *frontier*, which is a subset of vertices to be explored in each round. In round i, the algorithm *processes* (visits their neighbors) the current frontier $\mathcal{F}_i$ in parallel, and puts a subset of their neighbors to the next frontier $\mathcal{F}_{i+1}$, determined by a certain condition. For example, in BFS, a vertex u will add its neighbor v to the next frontier iff. u is the first to visit v (based on some linearization order if there are concurrent visits). The algorithm requires $O(D)$ rounds. One key challenge of in parallel BFS or similar approaches is the large cost to create and synchronize threads between rounds, which is especially costly for large-diameter graphs (more rounds needed). In this paper, we will show how PASGAL reduces the scheduling overhead to achieve better parallelism.

SPAA '24, June 17–21, 2024, Nantes, France

ACM ISBN 979-8-4007-0416-1/24/06.
https://doi.org/10.1145/3626183.3660258

2 ALGORITHMS

We now introduce the algorithms in PASGAL. For page limit, we only elaborate on SCC and briefly overview the others.

2.1 Parallel SCC

Most existing parallel SCC algorithms are based on ***reachability search***, which finds all vertices u that are reachable from a given vertex v. Most systems implements the reachability search by a BFS from v. This directly implies several (interrelated) challenges on large-diameter graphs. First, this incurs many rounds of creating and synchronizing threads with high overhead. Second, many real-world large-diameter graphs (e.g., road networks) are sparse with small average degrees. As a result, every parallel task (processing one vertex in the frontier) is small, and the cost of scheduling the thread may dominate the actual computation. Finally, because of sparsity, each frontier size is also likely small, which makes the algorithm unable to utilize all available threads.

Algorithm Redesign. To resolve this challenge, PASGAL uses the SCC algorithm in [13]. The key observation is that a reachability search does not require a strong BFS order. Therefore, one can relax the BFS order and visit vertices in an arbitrary order. In this way, the algorithm employs a technique called ***vertical granularity control*** to hide scheduling overhead, as introduced below.

Vertical Granularity Control. *Granularity control* (a.k.a. coarsening) is widely used in parallel programming, which also aims to avoid the overhead caused by generating unnecessary parallel tasks. For computations with sufficient parallelism, e.g., for a divide-and-conquer algorithm of size $n \gg P$ (P is the number of processors), a common practice is to stop recursively creating parallel tasks at a certain subproblem size and switch to a sequential execution to hide the scheduling overhead.

Inspired by this, the idea of VGC is also to increase each task size to hide the scheduling overhead. For reachability searches, we simply perform a *local search* to visit at least τ vertices, possibly in multiple hops. While globally the vertices are not visited in the BFS order, this does not affect the correctness of reachability. Note that here τ is equivalent to the base-case task size of granularity control, and is a tunable parameter. In this way, VGC 1) greatly reduces the number of rounds, since each round may advance multiple hops from the current frontier, and 2) quickly accumulates a large frontier size since the next frontier contains multiple-hop neighbors from the current frontier, and thus yields sufficient parallel tasks throughout the algorithm. Both outcomes effectively reduce (or hide) synchronization costs and enable much better parallelism.

Data Structure Design. Another useful technique for the SCC algorithm is a data structure called *hash bag* [13]. It maintains a dynamic set of vertices as the frontiers for parallel graph algorithms. For page limit, we refer the readers to [13] for more details.

2.2 Other Algorithms

Parallel SSSP. The SSSP algorithm in PASGAL is based on the *stepping algorithm framework* [7], and uses VGC and hash bags introduced in Sec. 2.1 to accelerate the frontier traversing.

Parallel BFS. Using VGC, we implemented a new BFS algorithm in PASGAL. We also use hash bags to maintain the frontiers. Our BFS algorithm generates the hop distance from the source to all vertices. For any vertex encountered in a local search from vertex v, we add it to the corresponding frontier if its hop distance can be updated by v. Due to local search, a vertex can be visited multiple times instead of exactly once as in standard BFS, since the updated distance in a local search is not necessarily the shortest distance, leading to extra work. To reduce this overhead, one special technique for BFS is that we maintain multiple frontiers, where frontier i maintains vertices with distance 2^i from the current frontier. In this way, we obtain the benefit of BFS by exploring multiple hops in one round, and also avoid visiting too many "unready" vertices in the frontier.

	Road				Social				
	AF	**NA**	**AS**	**EU**	**LJ**	**FB**	**OK**	**TW**	**FS**
n	33.5M	87.0M	95.7M	131M	4.85M	59.2M	3.07M	41.7M	65.6M
m'	44.8M	113M	123M	169M	69.0M	N/A	N/A	1.47B	3.61B
m	88.9M	220M	244M	333M	85.7M	185M	234M	2.41B	3.61B
D'	8276	9337	16660	11814	22	N/A	N/A	18	N/A
D	3948	4835	8794	4410	19	22	9	22	37
	k-NN				Web				
	CH5	**GL5**	**GL10**	**COS5**	**WK**	**SD**	**CW**	**HL14**	**HL12**
n	4.21M	24.9M	24.9M	321M	6.63M	89.2M	978M	1.72B	3.56B
m'	21.0M	124M	249M	1.61B	165M	2.04B	42.4B	64.4B	129B
m	29.7M	157M	310M	1.96B	300M	3.88B	74.7B	124B	226B
D'	5683	17268	13982	1390	62	145	506	800	5279
D	14479	21601	9053	1180	9	35	254	366	650

	Synthetic			
	REC	**SREC**	**TRCE**	**BBL**
n	100M	100M	16.0M	21.2M
m'	297M	204M	N/A	N/A
m	400M	336M	48.0M	63.6M
D'	59075	102151	N/A	N/A
D	50500	54843	5527	7849

Underline: undirected graphs
m': #edges in directed graphs
m: #edges in undirected or symmetrized graph
D': diameter of the directed graph
D: diameter of the undirected or symmetrized graph

Table 1: Tested graphs. Since it is hard to obtain the exact value of D and D', the number shown is a lower bound obtained by at least 1000 sampled searches on each graph.

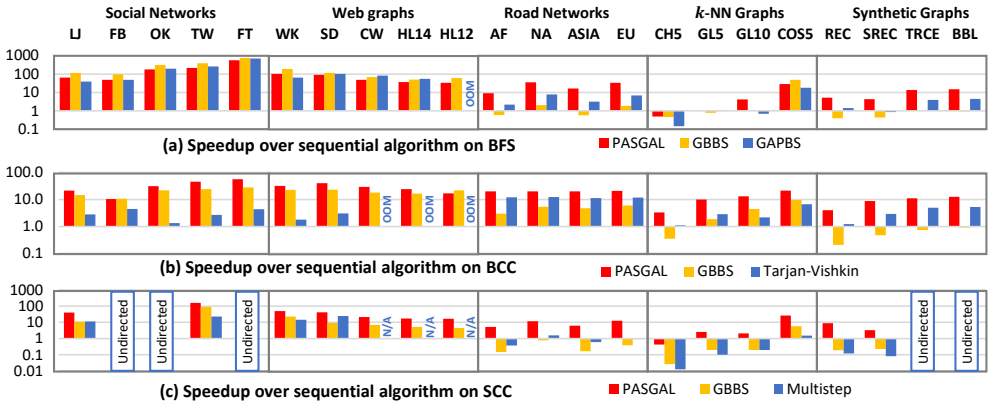
Parallel Biconnectivity. Different from other problems, the major performance gain of the BCC algorithm in PASGAL is due to algorithm redesign to achieve *stronger theoretical bounds*. The performance bottleneck of previous BCC algorithms either comes from the use of BFS that requires $O(D)$ rounds of global synchronizations (e.g., GBBS [4]), or requires $O(m)$ auxiliary space and does not scale to large graphs (e.g., Tarjan-Vishkin [12]). PASGAL uses the FAST-BCC algorithm in [8]. By redesigning the algorithm, FAST-BCC avoids the use of BFS, and achieves $O(n+m)$ work, polylogarithmic span, and $O(n)$ auxiliary space. We also use VGC and hash bags to further improve the performance.

3 EXPERIMENTAL RESULTS

Library Design. We release the code of PASGAL [5]. PASGAL is implemented in C++ using ParlayLib [3] for fork-join parallelism and some parallel primitives (e.g., sorting). Four algorithms (BFS, BCC, SCC and SSSP) are included. A readme file about compiling and running the library is provided in the repository.

Setup. We run our experiments on a 96-core (192 hyperthreads) machine with four Intel Xeon Gold 6252 CPUs and 1.5 TB of main memory. We use `numactl -i all` in parallel experiments.

We tested on 22 graphs, including social networks, web graphs, road networks, k-NN graphs, and synthetic graphs. All the graphs



Baselines for BFS:
GBBS [4], GAPBS [2]
Sequential: a queue-based implementation

Baselines for BCC:
GBBS [4], Tarjan-Vishkin [12] from [8]
Sequential: Hopcroft-Tarjan [9]

Baselines for SCC:
GBBS [4], Multistep [10]
Sequential: Tarjan's algorithm [11]

Figure 1: Speedup of parallel algorithms over the standard sequential algorithm. y-axis is in log-scale. Bars below 1.0 mean the parallel algorithm is slower than a sequential one. Some bars are invisible because they are close to 1. "N/A": not applicable. "OOM": out-of-memory.

are from existing research papers and public datasets. The graph information is given in Tab. 1. We provide the full graph information and corresponding citations in the full paper. We call the social and web graphs *low-diameter graphs* as they have diameters mostly within a few hundred. We call the road, k-NN, and synthetic graphs *large-diameter graphs* as their diameters are mostly more than a thousand. We symmetrize directed graphs for testing BCC. SCC does not apply to undirected graphs.

We present the performance comparison in Fig. 1. For page limit, we only show results for SCC, BCC, and BFS. For each problem, we compare the relative speedup of all parallel algorithms to a sequential one. All baselines are introduced in Fig. 1.

In all the tests, PASGAL is always competitive on small-diameter graphs: across all graphs, PASGAL is within 1.3× of the running time compared to the fastest baseline on BCC, 2× on BFS, and always faster than all baselines on SCC. Parallel BFS on social networks is one of the most well-studied parallel graph algorithms, and all parallel algorithms achieve superlinear speedup on some social networks due to various optimizations (e.g., the direction optimization [1]). PASGAL achieves good scalability and is 49–570× faster than the standard sequential algorithm using 192 threads.

On large-diameter graphs, PASGAL achieves much better performance than *all baselines*. On BCC, due to theoretical efficiency, PASGAL consistently outperforms the sequential Hopcroft-Tarjan algorithm. It is up to 3.45× faster than the best baseline on each graph. On SCC and BFS, PASGAL is always faster than the sequential baseline except for one graph CH5, which has very large diameter compared to its small size. Different from BCC, our SCC and BFS algorithms do not have strong span bound. Using VGC can only alleviate the scalability issue on large-diameter graphs, but may still be unable to eliminate the issue on adversarial graphs (e.g., a chain). Still, PASGAL is the fastest among all parallel implementations on most real-world large-diameter graphs. It is up to 5× faster than the best baseline on BFS, and up to 46× on SCC.

4 CONCLUSION AND FUTURE WORK

In this paper, we present PASGAL, a scalable graph library specially optimized for large-diameter graphs. Some interesting future directions include further seeking new ideas to improve the performance of BFS on small-diameter graphs that also work well with VGC, as well as improving the performance for BFS and SCC on graphs with very large diameters. We believe the techniques in PASGAL can be extended to more problems, including k-core and other peeling algorithms, k-connectivity, point-to-point shortest paths, etc. We plan to add them to PASGAL in the future.

Acknowledgement This work is supported by NSF grants CCF-2103483, CCF-2238358, CCF-2339310, and IIS-2227669, the UCR Regents Faculty Development Awards, and the Google Research Scholar Program.

REFERENCES

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. In *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, pages 1–10, 2012.
[2] S. Beamer, K. Asanović, and D. Patterson. The gap benchmark suite. *arXiv preprint arXiv:1508.03619*, 2015.
[3] G. E. Blelloch, D. Anderson, and L. Dhulipala. Parlaylib — a toolkit for parallel algorithms on shared-memory multicore machines. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 507–509, 2020.
[4] L. Dhulipala, G. E. Blelloch, and J. Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. *ACM Transactions on Parallel Computing (TOPC)*, 8(1):1–70, 2021.
[5] X. Dong, Y. Gu, Y. Sun, and L. Wang. Pasgal: Parallel and scalable graph algorithm library. https://github.com/ucrparlay/PASGAL, 2024.
[6] X. Dong, Y. Gu, Y. Sun, and L. Wang. Pasgal: Parallel and scalable graph algorithm library. *arXiv preprint:2404.17101*, 2024.
[7] X. Dong, Y. Gu, Y. Sun, and Y. Zhang. Efficient stepping algorithms and implementations for parallel shortest paths. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 184–197, 2021.
[8] X. Dong, L. Wang, Y. Gu, and Y. Sun. Provably fast and space-efficient parallel biconnectivity. In *ACM Symposium on Principles and Practice of Parallel Programming (PPOPP)*, pages 52–65, 2023.
[9] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.
[10] G. M. Slota, S. Rajamanickam, and K. Madduri. Bfs and coloring-based parallel algorithms for strongly connected components and related problems. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 550–559. IEEE, 2014.
[11] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. on Computing*, 1(2):146–160, 1972.
[12] R. E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM J. on Computing*, 14(4):862–874, 1985.
[13] L. Wang, X. Dong, Y. Gu, and Y. Sun. Parallel strong connectivity based on faster reachability. *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1(2):1–29, 2023.