Semi-Asymmetric Parallel Graph Algorithms for NVRAMs

Laxman Dhulipala[1] Charles McGuffey[1] Hongbo Kang[2] Yan Gu[3]
Guy E. Blelloch[1] Phillip B. Gibbons[1] Julian Shun[4]
[1]Carnegie Mellon University [2]Tsinghua University [3]U.C. Riverside [4]MIT CSAIL

Over the past decade, there has been a steady increase in the main-memory sizes of commodity multicore machines, which has led to the development of fast shared-memory algorithms for processing massive graphs with hundreds of billions of edges on a single machine [4, 11, 12]. Single-machine analytics by-and-large outperform their distributed memory counterparts, running up to orders of magnitude faster using much fewer resources [4, 10–12]. The trend in increasing memory sizes continues today in the form of new non-volatile memory technologies that are now emerging on the market (e.g., Intel's *Optane DC Persistent Memory*). These devices provide an order of magnitude greater memory capacity per DIMM than traditional DRAM, and offer byte-addressability and low idle power, thereby providing a realistic and cost-efficient way to equip a commodity multicore machine with multiple terabytes of non-volatile RAM (NVRAM).

Due to these advantages, NVRAMs are likely to be a key component of many future memory hierarchies, likely in conjunction with a smaller amount of traditional DRAM. However, a challenge of these technologies is to overcome an *asymmetry* between reads and writes—write operations are more expensive than reads in terms of energy and throughput. This property requires rethinking algorithm design and implementations to minimize the number of writes to NVRAM [2, 3, 16]. As an example of the technology and its tradeoffs, the experiments in our work are done on a 48 core machine that has 8x as much NVRAM as DRAM (and we are aware of machines with 16x as much NVRAM as DRAM [6]), where combined read throughput for all cores from the NVRAM is about 3x slower than reads from the DRAM, and writes on the NVRAM are a further factor of about 4x slower [7, 15] (a factor of 12 total).

A property of most graphs used in practice is that they are sparse, but still tend to have many more edges than vertices, often from one to two orders of magnitude more. This is true for almost all social network graphs [8], but also for many graphs that are derived from various simulations [13]. Given that a large graph can have over 100 billion edges (requiring around a terabyte of storage), but only a few billion vertices, a popular and reasonable assumption is that vertices, but not edges, fit in DRAM [1, 5, 9, 14, 17].

With these characteristics of NVRAM and real-world graphs in mind, we propose a *semi-asymmetric* approach to parallel graph analytics, in which (i) the full graph is stored in NVRAM and is accessed in *read-only mode* and (ii) the amount of DRAM is proportional to the number of vertices. Although completely avoiding writes to the NVRAM may seem overly restrictive, the approach has the following benefits: (i) algorithms avoid the high cost of NVRAM writes, (ii) the algorithms do not contribute to NVRAM wear-out or wear-leveling overheads, and (iii) algorithm design is independent of the actual cost of NVRAM writes, which has been shown to vary based on access pattern and number of cores [7, 15] and will likely change with innovations in NVRAM technology and controllers. Moreover, it enables an

Problem	T_1	T_{48h}	SU	Work	Gal	GBBS$_M$
BFS	561	12.2	45.9	$O(m)$	35.2	35.7
Weighted BFS	4420	98	45.1	$O(m)^*$	—	190
Bellman-Ford	3760	82.3	45.5	$O(\mathrm{d}(G)m)$	118	149
1-Src Widest Path	3479	77.5	44.8	$O(\mathrm{d}(G)m)$	—	89.8
1-Src Betweenness	3267	68.5	47.6	$O(m)$	56.4	148
$O(k)$**-Spanner**	2219	55.1	40.2	$O(m)^*$	—	124.8
LDD	985	24.0	41.0	$O(m)^*$	—	62.2
Connectivity	1564	36.2	43.2	$O(m)^*$	76.0	75.8
Spanning Forest	2439	61.3	38.3	$O(m)^*$	—	116
Biconnectivity†	10930	234	46.7	$O(m)^*$	—	482
MIS	2308	52.3	44.1	$O(m)^*$	—	98.8
Maximal Matching†	7280	166	43.1	$O(m)^*$	—	455
Graph Coloring	10880	239	45.5	$O(m)^*$	—	216
Apx Set Cover†	7968	193	41.2	$O(m)^*$	—	246
k**-core**	8348	215	38.8	$O(m)^*$	—	259
Apx Dens. Subgraph	1930	42.2	45.7	$O(m)$	—	106
Triangle Counting†	—	3529	—	$O(m^{3/2})$	—	1665
PageRank Iteration	1033	23.6	43.5	$O(m)$	—	37.5
PageRank	—	827	—	$O(P_{it} \cdot m)$	1706	1318

Table 1. Running times and speedup of our algorithms on the Hyperlink2012 graph (Columns 2–4) using NVRAM where T_1 corresponds to the single-threaded time, T_{48h} corresponds to the running time on 48 cores with hyper-threading, and SU is the parallel speedup. The single-threaded times for triangle counting and PageRank are omitted because they did not finish in a reasonable amount of time. We use † if our algorithm uses $O(n + m/\log n)$ words of memory. Column 5 shows the work of our algorithms in the PSAM model. We use * to denote that a bound holds in expectation. $d(G)$ is the diameter of the graph and P_{it} is the number of iterations of PageRank until convergence. In all cases we assume $m = \Omega(n)$. Column 6 (Gal) shows the running time of Galois implementations obtained on a similarly configured NVRAM-equipped machine reported in [6], with problems not solved by them marked with —. Column 7 (GBBS$_M$) shows running times of unmodified GBBS [4] codes run using Memory Mode [7] on this graph. All times are in seconds.

important NUMA optimization in which a copy of the graph is stored on each socket, for fast read-only access without any cross-socket coordination. Finally, with no graph mutations, there is no need to re-compress the graph on-the-fly when processing compressed graphs [4].

The key question, then, is the following: *Is the (restrictive) semi-asymmetric approach effective for designing fast graph algorithms?* In our work, we provide both theoretical and experimental evidence of the approach's effectiveness.

We consider 18 well-studied graph problems (see Table 1) and design fast and highly scalable semi-asymmetric algorithms for them. The key innovations are in ensuring that the updated state is associated with vertices and not edges, which is particularly tricky (i) for certain edge-based parallel graph traversals and (ii) for algorithms that "delete" edges as they go along in order to avoid revisiting them once they are no longer needed. We provide general techniques to solve both problems. For the latter, used by four of our algorithms, we require relaxing the prescribed amount of DRAM to be on the order of one bit per edge.

From a theoretical perspective, we propose a model for analyzing algorithms in the semi-asymmetric setting. The model, called the Parallel Semi-Asymmetric Model (PSAM), consists of a shared asymmetric large-memory with unbounded size that can hold the entire graph, and a shared symmetric small-memory with $O(n)$ words of memory, where n is the number of vertices in the graph. In a relaxed version of the model, we allow small-memory size of $O(n + m/\log n)$ words, where m is the number of edges in the graph. Although we do not use writes to the large-memory in our algorithms, the PSAM model permits writes to the large-memory, which are $\omega > 1$ times more costly than reads. We prove strong theoretical bounds in terms of PSAM work and depth for all of our parallel algorithms (we show the work bounds in Table 1). Most of our algorithms are work-efficient (performing asymptotically the same work as the best sequential algorithm for the problem) and have polylogarithmic depth (parallel time). Our theoretical guarantees ensure that our algorithms perform reasonably well across graphs with different characteristics, machines with different core counts, and NVRAMs with different read-write asymmetries.

We experiment with implementations of our algorithms on a variety of large-scale real-world graphs using Intel Optane DC Persistent Memory. Our implementations are able to scale to the *largest publicly-available graph*, the Hyperlink2012 graph with over 3.5 billion vertices and 128 billion edges (and 225 billion edges for algorithms running on the undirected/symmetrized graph). Table 1 shows the running times on the Hyperlink2012 graph using a 48-core machine with 375GB of DRAM and 3TB of NVRAM. Note that we cannot fit the entire Hyperlink2012 graph and run algorithms on this graph in the DRAM of this machine. Our NVRAM algorithms are 1.86x faster on average than Galois [6] algorithms (state-of-the-art algorithms designed for NVRAM), and 1.87x faster on average than existing DRAM-only GBBS [4] codes run using Memory Mode [7] on the Hyperlink2012 graph. Moreover, our algorithms running on NVRAM nearly match the running times of GBBS algorithms running *entirely in DRAM*, with all but three algorithms within 17%, by effectively hiding the costs of repeatedly accessing NVRAM versus DRAM.

The main contributions of our work are:

(**1**) We propose a semi-asymmetric approach to parallel graph analytics that avoids writing to the NVRAM and uses DRAM proportional to the number of vertices.

(**2**) We design semi-asymmetric algorithms for 18 fundamental graph problems, and present general techniques for devising such algorithms. Our codes are open source.[1]

(**3**) We introduce the Parallel Semi-Asymmetric Model, and give (near) work-optimal algorithms in the model.

(**4**) We evaluate our algorithms on a state-of-the-art NVRAM system, and show that our algorithms outperform prior work and nearly match DRAM-only performance.

References

[1] J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 2002.

[2] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, Y. Gu, and J. Shun. Sorting with asymmetric read and write costs. In *SPAA*, 2015.

[3] E. Carson, J. Demmel, L. Grigori, N. Knight, P. Koanantakool, O. Schwartz, and H. V. Simhadri. Write-avoiding algorithms. In *IPDPS*, 2016.

[4] L. Dhulipala, G. E. Blelloch, and J. Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *SPAA*, 2018.

[5] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *TCS*, 2005.

[6] G. Gill, R. Dathathri, L. Hoang, R. Peri, and K. Pingali. Single machine graph analytics on massive datasets using Intel Optane DC Persistent Memory. *CoRR*, abs/1904.07162, 2019.

[7] J. Izraelevitz, J. Yang, et al. Basic performance measurements of the Intel Optane DC Persistent Memory module. *arXiv preprint arXiv:1903.05714*, 2019.

[8] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, 2014.

[9] A. McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 2014.

[10] F. McSherry, M. Isard, and D. G. Murray. Scalability! But at what COST? In *HotOS*, 2015.

[11] J. Shun and G. E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. In *PPOPP*, 2013.

[12] J. Shun, L. Dhulipala, and G. E. Blelloch. Smaller and faster: Parallel processing of compressed graphs with Ligra+. In *DCC*, 2015.

[13] SuiteSparse Matrix Collection. https://sparse.tamu.edu/.

[14] P. Sun, Y. Wen, T. N. B. Duong, and X. Xiao. GraphMP: An efficient semi-external-memory big graph processing system on a single machine. In *ICPADS*, 2017.

[15] A. van Renen, L. Vogel, V. Leis, T. Neumann, and A. Kemper. Persistent memory I/O primitives. In *DaMoN*, 2019.

[16] S. D. Viglas. Write-limited sorts and joins for persistent memory. *PVLDB*, 2014.

[17] D. Zheng, D. Mhembere, R. Burns, J. Vogelstein, C. E. Priebe, and A. S. Szalay. Flashgraph: Processing billion-node graphs on an array of commodity SSDs. In *FAST*, 2015.

[1] https://github.com/ldhulipala/gbbs