A Parallel Graph Environment for Real-World Data Analytics Workflows

Vito Giovanni Castellana*, Maurizio Drocco*, John Feo*, Jesun Firoz†, Thejaka Kanewala†,
Andrew Lumsdaine*, Joseph Manzano*, Andrés Marquez*, Marco Minutoli*, Joshua Suetterlein*,
Antonino Tumeo*, Marcin Zalewski*
*High Performance Computing
Pacific Northwest National Laboratory
Richland, WA, USA
{vitoGiovanni.castellana, maurizio.drocco, john.feo, andrew.lumsdaine, joseph.manzano,
andres.marquez, marco.minutoli, joshua.suetterlein, antonino.tumeo, marcin.zalewski}@pnnl.gov
‡School of Informatics, Computing, and Engineering
Indiana University
Bloomington, IN, USA
{jsfiroz,thejkane}@iu.edu

***Abstract*—Economic competitiveness and national security depend increasingly on the insightful analysis of large data sets. The diversity of real-world data sources and analytic workflows impose challenging hardware and software requirements for parallel graph platforms. The irregular nature of graph methods is not supported well by the deep memory hierarchies of conventional distributed systems, requiring new processor and runtime system designs to tolerate memory and synchronization latencies. Moreover, the efficiency of relational table operations and matrix computations are not attainable when data is stored in common graph data structures. In this paper, we present HAGGLE, a high-performance, scalable data analytics platform. The platform's hybrid data model supports a variety of distributed, thread-safe data structures, parallel programming constructs, and persistent and streaming data. An abstract runtime layer enables us to map the stack to conventional, distributed computer systems with accelerators. The runtime uses multithreading, active messages, and data aggregation to hide memory and synchronization latencies on large-scale systems.**

***Index Terms*—Graph Analytics, Attributed Graphs**

I. Introduction

As data analytics problems grow in complexity and scale, neither large-scale computing systems nor single data platforms (relational tables or graphs) developed over the years are proving suitable for the complex workflows emerging in science, security, human behavior, and commerce [1], [2]. Notably, modern architectures have been optimized for problems exhibiting high degrees of spatial and temporal locality, properties that due to their fundamental nature, large-scale graph analytics problems do not possess. As a result, the performance of graph analytics problems is fundamentally limited by data movement costs.

This research is in part supported by the Defense Advanced Research Projects Agency's (DARPA) Hierarchical Identify Verify Exploit Program and the High Performance Data Analytics Program (HPDA) at DOE Pacific Northwest National Laboratory (PNNL). PNNL is operated by Battelle Memorial Institute under Contract DE-AC06-76RL01830.

To address this gap, we have designed the Hybrid Attributed Generic Graph Library Environment (HAGGLE), a scalable data analytics platform composed of new distributed data structures and parallel programming constructs, and an abstract runtime layer that allows us to quickly port the software stack to novel, purposely design hardware, as well as modern conventional, distributed high-performance computing systems with accelerators.

The paper proceeds as follows. Section II provides a general overview of HAGGLE and its objectives. Section III discusses a motivating example for the hybrid data model. Sections IV, V, and VI respectively discuss the data structures and algorithms library, the runtime layer, and the model for orchestrating execution of parallel graph algorithm. Section VII discusses related frameworks and their differences with respect to HAGGLE. Finally, Section VIII concludes the paper.

II. HAGGLE overview

Figure 1 provides a high-level overview of the HAGGLE platform. We are developing HAGGLE for Technical Area 2 (TA2) of the Defense Advanced Research Projects Agency's (DARPA) HIerarchical Verify and Exploit (HIVE) program. HAGGLE is complete software stack devised to run on the novel custom processor architectures being designed by Technical Area 1 (TA1) performers, and driven by the application requirements identified by the Technical Area 3 (TA3) performers. HAGGLE comprises several layers. It provides an extensible and generic SDK with STL-complaint distributed, thread-safe data structures that support static, persistent, dynamic, and streaming data. The SDK's hybrid data model simplify the development of method libraries for relational, graph, and linear algebra methods. HAGGLE integrates an Extended Abstract Graph Machine (EAGM) to reason about data flow, data locality, asynchronous operations, and task scheduling. It leverages an abstract runtime model supporting a variety of memory and execution models and providing

978-3-9819263-2-3/DATE19/©2019 EDAA

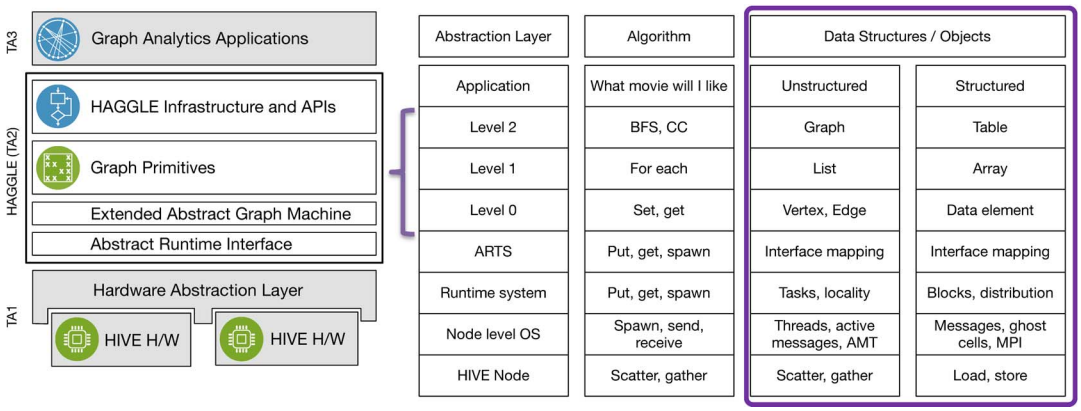

Fig. 1. Overview of the HAGGLE platform

introspection mechanisms to enable performance monitoring and adaptation.

The computation and operation primitives are exposed through the HAGGLE APIs and are organized in levels, from runtime operations, to data structures and objects, to algorithms. HAGGLE's APIs, primitives, distributed objects, and data structures are actually implemented in the Scalable High Performance Algorithms and Data structures (SHAD) [3]. Underneath, SHAD interfaces to HAGGLE's Abstract Runtime System (ARTS) provide the set of necessary runtime functionalities to support the data-dependent behaviors and fine-grained, unpredictable data accesses of graph algorithms, coupled with locality requirements for the dense tables of attributes. The EAGM bridges SHAD with ARTS. The previously introduced Abstract Graph Machine (AGM) [4] model decomposes algorithms into processing functions and strict/weak orderings of work items, enabling to devise an algorithm taxonomy along these lines. Its extended version describes hybrid hierarchical algorithms using different orderings at different algorithmic levels to match hardware performance, providing ARTS with the information to better schedule and load-balance the hybrid data structures exploration.

III. A MOTIVATING EXAMPLE

Critical analytics examine not only the basic link structure of data sets, but also the deeper relationships among heterogeneous data from independent sources, that often include different types of uniquely attributed entities and relationships. *Property graphs* are powerful mathematical abstractions that encode structural metrics and enable reasoning about relationships (edges) among heavily attributed entities (vertices). Consider the data tables in Figure 2 for a consumer product scenario. The relational tables concisely store the data, but obfuscate the relationship among rows in the same and different tables.

Figure 3 shows a graph-based representation, in terms of Resource Description Framework (RDF) [5], of the data tables and the corresponding edge matrix. We have numbered the vertices left-to-right and bottom-to-top for the purpose of constructing the edge matrix, and have removed the edge labels in the graph to improve clarity. The edge matrix lacks the semantic information necessary to compute many important quantities; for example, the betweenness centrality of *banks* on paths from *producers* to *vendors*. The single-label vertices and edges of the RDF graph result in "starification" (see the person nodes near the top of the graph) as we are forced to represent entities and attributes as vertices, and create an edge between an entity and each of its attributes. The resulting "starification" wastes memory and significantly increases the execution time of many methods. The edge attributes in the third column of *Sells* and the third and fourth columns of *Customer_Of* are difficult to express and are not shown in the figure. They require value specific labels such as *sells_GTI_for* and *opened_BAC_Acct_1111_on*. Finally, numeric and date attributes often result in a vertex per instance (i.e., each person has a different account number and each product has a different price) rather than a small set of vertices shared among many instances (i.e., country names).

It is possible to define the vertices and edges of an RDF graph as structs of attributes. While that eliminates the stars, value unique labels, and plethora of integer and date vertices, it distributes like attributes throughout memory increasing the cost of quickly finding all nodes with a particular attribute. For example, if country were stored locally with each person vertex, then there would be no USA vertex from which we could visit all USA persons in one hop.

To cope with these real world data, we designed a *hybrid data model* that uses tables to store the dense property information of entities and relationships, and hierarchical index tables to define a graph view of the data comprised of strongly typed, attributed vertices and edges [6]. Real data analytics requires the data to be first class and the overlay of multiple simultaneous graphs to represent the multitude of relationships between the data. The HAGGLE framework generalizes our hybrid data model to support a wider range of analytics and redesigns methods to scale in performance and energy on new generation architectures as well as commodity clusters with accelerators.

Person

Name	Email	City	Country
John	john@...	New Yoek	USA
Suzy	suzy@...	Chicago	Canada
Fred	fred@...	Erie	USA
Mary	mary@...	Dallas	USA

Product

Name	Type	Enterprise
GTI	Car	VW
52" Bravia	TV	Sony
48" Bravia	TV	Sony
Explorer	Car	Ford

Sells

Enterprise	Product	Price
Lemons	GTI	$25000
Costco	52" Bravia	$730
Lemons	Exploroer	$28000
Costco	48" Bravia	$450

Enterprise

Name	Type	Country
Ford	Autos	USA
Sony	Electronics	Japan
Lemons	Car Dealer	USA
Costco	Wholesaler	USA
BAC	Bank	USA
VW	Autos	Germany

Customer_Of

Name	Enterprise	Acct #	From
Suzy	BAC	1111	10/5/205
John	Costco	2222	9/28/2002
Costco	Sony	34211	8/8/1990
Lemons	VW	5023	7/14/1970
Fred	BAC	3333	5/23/2007
Lemons	BAC	9876	6/8/1965

Fig. 2. Consumer Product Data Tables

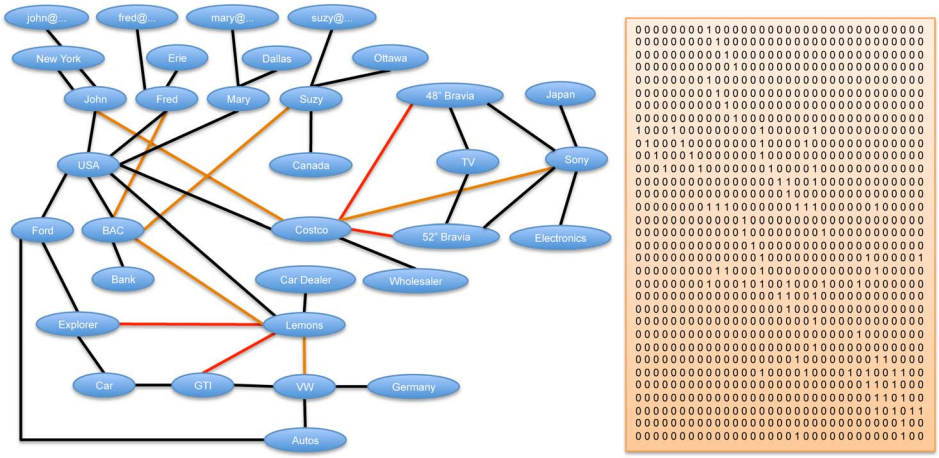

Fig. 3. Consumer Product RDF graph and edge matrix

IV. SHAD: Scalable High-performance Algorithms and Data-structures

Figure 4 provides a high-level overview of the functionalities provided by the SHAD library [3] in the HAGGLE framework. At its core, SHAD provides *distributed* C++ General Purpose Data Structures. These include fundamental data structures such Array, Vector, Set, and Map. All the SHAD data structures are implemented as parallel and thread-safe global objects, providing the user with a shared memory abstraction on distributed memory systems and efficient parallel execution. Furthermore, their API is compliant with the Standard Template Library (STL), thus their use is similar to the conventional C++ STL data structures (that, however, are not parallel and not thread safe). Users can define higher-level libraries, named SHAD Extensions, by composing the fundamental data structures and other extensions. Two examples specifically developed for the HAGGLE framework are the Graph Library, also supporting attributed graphs, and the Linear Algebra Library. SHAD interacts with the underlying system through an Abstract Runtime Interface. The interface

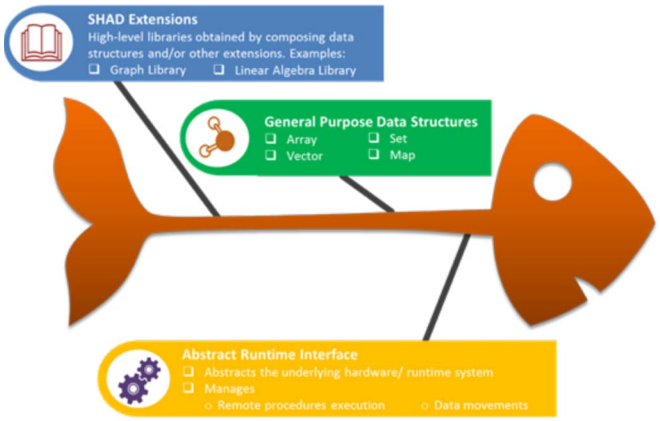

Fig. 4. Functionalities provided by SHAD

invokes the underlying runtime functions to manage data movement and remote procedure execution.

Figure 5 provides a more detailed view of the SHAD design

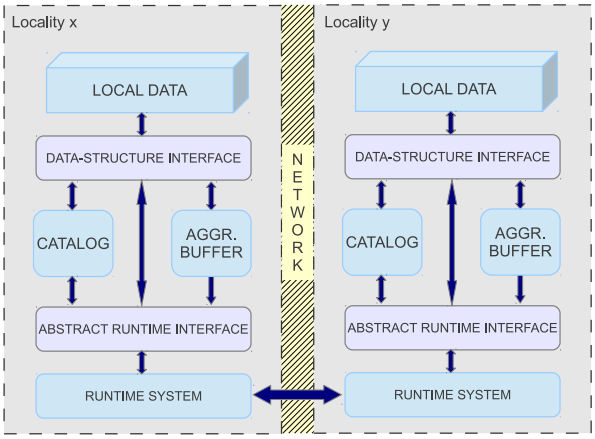

Fig. 5. SHAD architecture

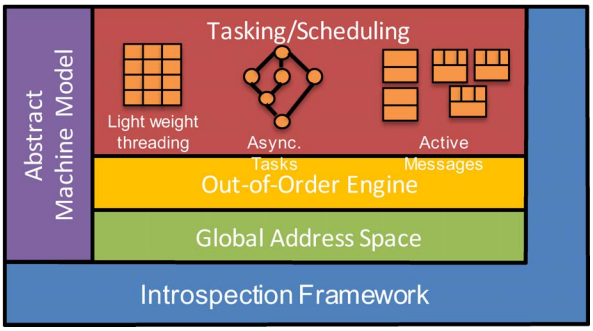

Fig. 6. Functionalities provided by ARTS

Definition

An Abstract Graph Machine(AGM) is a 6-tuple (G, WorkItem, Q, π, $<_{wis}$, S), where

1. $G = (V, E, vmaps, emaps)$ *is the input graph,*
2. $WorkItem \subseteq (V \times P_0 \times P_1 \cdots \times P_n)$ *where each* P_i *represents a state value or an ordering attribute,*
3. *Q - Set of states represented as mappings,*
4. $\pi : WorkItem \longrightarrow 2^{WorkItem}$ *is the processing function,*
5. $<_{wis}$ *- Strict weak ordering relation defined on workitems,*
6. *S* ($\subseteq$ *WorkItem) - Initial workitem set.*

Fig. 7. Definition of an Abstract Graph Machine

and its data-structure layout. The actual data structures are partitioned across (distributed) memory domains (e.g., cluster nodes, numa nodes, etc), named localities. Each locality hosts a portion of the data (*local data*) and holds an object *catalog* that manages the object lifecycle, along with an aggregation buffer to coalesce fine-grained data transfers across locality and improve bandwidth utilization. The data-structure interface, which constitutes the API for the library users, allows the various components to interact and call the methods implemented in the abstract runtime interface to operate on the data structures, which will then use the functionalities provided by the actual runtime system. In the HAGGLE framework, the runtime system is ARTS. As previously highlighted, the data-structure interface provides a global, shared memory view of the associated objects and allows operating on them with functions similar to those provided by the C++ STL on a single node machine. The data management mechanisms enable, for dynamic data structures, dynamic insert, delete, and updates, including resizing.

V. ARTS: Abstract RunTime System

Figure 6 provides a high-level overview of the functionalities provided by ARTS in the HAGGLE framework. ARTS' design is derived from previous extensive experience with other runtimes that targeted large-scale scientific simulations [7], as well as irregular applications [8], [9]. ARTS considers an abstract machine model that enables it to run on custom designed architectures as well as modern commodity clusters with accelerators. The runtime supports a global address space, allowing the implementation of higher-level libraries assuming a shared-memory abstraction, and thus, not requiring data partitioning. ARTS implements an out-of-order engine that takes care of memory and operation dependencies. It enables implementing different memory models and supporting several distributed coherence protocols that allows exploring trade-offs between data replication and data movement. ARTS has an extensive support for efficient, resource-aware task scheduling. It supports asynchronous tasking with implicit and explicit task dependencies (through a directed acyclic graph). It implements active messages with efficient termination detection for messages that do not need to return anything back to the originating site, and futures. ARTS provides lightweight multithreading with explicit thread yielding. This allows tolerating access latencies (by switching to another active thread) when executing remote operations. In combination with the threading, ARTS also allows aggregating communication for those network substrates that are able to reach peak bandwidth only with large, batched, transactions. Graph algorithms and, in general, irregular applications tend to exhibit unpredictable, fine-grained network and memory accesses that benefit from coalescing together multiple operations destined to the same location (e.g., cluster node, memory node).

Finally, ARTS provides a generalized introspection framework that allows the runtime system to monitor its operations and collect profiling information. For data-dependent applications, the monitoring allows application/system to take runtime actions to optimize scheduling, communication, synchronization events, and dynamic buffer. It also allows to quickly identify bottlenecks in the runtime for each supported target architecture.

VI. EAGM: Extended Abstract Graph Machine

AGM is a model for designing distributed memory parallel graph algorithms. AGM attempts to maximize the computation-to-communication ratio by minimizing remote accesses, by minimizing synchronization overhead, by avoiding expensive distributed computations (e.g., subgraph com-

putations), and by avoiding operations that could increase the number of communicated messages (e.g., pointer jumping). Figure 7 shows the definition of an AGM. An AGM consists of: a definition of a graph, a set of states, a definition of a set *WorkItem*, an initial set of *workitem*s, a processing function, and a strict weak ordering relation. An AGM maintains states against vertices or edges. States are changed by invoking the processing function (π). A *workitem* is the basic unit of data that invokes a processing function. the set *WorkItem* represents all the *workitem*s that an algorithm could generate. The strict weak ordering relation separates the WorkItem set in ordered partitions—*workitem*s belonging to the same partition can execute in parallel. A processing function is further decomposed in a conditional check (C) function, that verifies wether the input *workitem* is applicable to π, in the states update (U) function, and in the new *workitem*s construction (N) function.

The Extended AGM model maps an AGM algorithm to an architecture. AGM does not specify spatial or temporal characteristics: for example, it does not distinguish whether the parallel algorithm targets a shared-memory or a distributed-memory architecture. An EAGM consists of an architecture hierarchy with annotated strict weak orderings, and a function that specifies how *workitems* should be distributed across different levels of the memory hierarchy.

In the HAGGLE framework, EAGM bridges the high-level API, with the data structures and algorithm description, with the runtime layer, to provide opportunities to organize the parallel execution of the algorithm.

VII. Related frameworks

Table I lists selected related graph systems, which differ significantly in their purpose and design. BGL is a well-known example of a widely adopted C++ graph library. One of the main principles of BGL is a rigorous analysis of the problem domain, resulting in a hierarchy of graph concepts and generic algorithms, and data structures defined in terms of these concepts. Parallel BGL, PXGL, and PBGL2, based on AM++, extend this approach to efficient distributed computing, while MTGL applied this approach to highly multithreaded shared-memory systems. Other systems concentrate on targeting specific architectures through abstractions (e.g., Gunrock for GPU), on providing comprehensive runtime systems (e.g., Galois and STAPL), on portable domain-specific languages (e.g., CombBLAS), on dynamic graphs (e.g., STINGER), or on large-scale scalability (e.g., Giraph). Also shown are graph database systems that provide property graphs. These seek to provide efficient and scalable access to the property graph through query interfaces (e.g., SPARQL). Query languages are an integral component of graph databases, and some systems such as TinkerPop specifically concentrate on portability and interoperability by providing a model and a query language and then implementing backends to specific graph database implementations.

VIII. Conclusions

We have presented the Hybrid Attributed Generic Graph Library Environment (HAGGLE), a high-performance, scalable data analytics platform. We motivated, as key feature of the platform, the support for a hybrid data model that uses tables to store the dense property information of entities and relationships, and hierarchical index tables to define a graph view of the data comprised of strongly typed attributed vertices and edges. We discussed SHAD, the library of distributed data structures and algorithms that also provides the HAGGLE's API. We presented the features integrated in ARTS, the Abstract Runtime System that enables HAGGLE to run on a variety of custom and commodity high performance systems. We discussed the Extended Abstract Graph Machine, the model through which parallel graph algorithms are mapped to an architecture. HAGGLE provides unique features in an integrated stack, not available with any other frameworks. Development of the platform is currently quickly progressing, with demonstrators of the various components already available.

References

[1] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington: Microsoft Research, 2009. [Online]. Available: http://research.microsoft.com/en-us/collaboration/ fourthparadigm/

[2] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, "Challenges in parallel graph processing," *Parallel Processing Letters*, vol. 17, no. 01, pp. 5–20, 2007.

[3] V. G. Castellana and M. Minutoli, "Shad: The scalable high-performance algorithms and data-structures library," in *CCGRID 2018: 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2018, pp. 442–451.

[4] T. A. Kanewala, M. Zalewski, and A. Lumsdaine, "Abstract graph machine," *CoRR*, vol. abs/1604.04772, 2016. [Online]. Available: http://arxiv.org/abs/1604.04772

[5] D. Brickley and R. V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema," http://www.w3.org/TR/rdf-schema, Feb. 2004.

[6] D. G. Chavarría-Miranda, V. G. Castellana, A. Morari, D. Haglin, and J. Feo, "Graql: A query language for high-performance attributed graph databases," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016*, 2016, pp. 1453–1462.

[7] J. Landwehr, J. Suetterlein, A. Márquez, J. Manzano, and G. R. Gao, "Application characterization at scale: Lessons learned from developing a distributed open community runtime system for high performance computing," in *Proceedings of the ACM International Conference on Computing Frontiers*, ser. CF '16, 2016, pp. 164–171.

[8] A. Morari, A. Tumeo, D. G. Chavarría-Miranda, O. Villa, and M. Valero, "Scaling irregular applications through data aggregation and software multithreading," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*, 2014, pp. 1126–1135.

[9] J. J. Willcock, T. Hoefler, N. G. Edmonds, and A. Lumsdaine, "Am++: A generalized active message framework," in *PACT 2010: 19th International Conference on Parallel Architectures and Compilation Techniques*, 2010, pp. 401–410.

[10] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.

[11] J. Siek, L.-Q. Lee, and A. Lumsdaine, "The Boost Graph Library (BGL)," http://www.boost.org/libs/graph, Apr. 2016.

[12] A. Buluç and J. R. Gilbert, "The Combinatorial BLAS: Design, Implementation, and Applications," *International Journal of High Performance Computing Applications*, vol. 25, no. 4, pp. 496–509, 2011.

System	Arch.[2]	Lang.	Avail.	Active	Algs.
BGL [10], [11]	C	C++	Yes	Yes	69
CombBLAS [12], [13]	D	C++	Yes [14]	Yes (yearly)	0/8[1]
Galois [15]	S	C++	Yes [16]	No	0/21[1]
GEMS [17]	D	C++	Yes	Yes	1+ (search)
Giraph [18]	D	Java	Yes [19]	Yes	8
GraphLab [20]	D	C++	Yes [21]	Yes	27
Gunrock [22]	G	C++	Yes	Yes	10
STAPL [23]	D	C++	No	Yes	Unknown
MTGL [24]	S	C++	Yes [25]	No	14
P[B,X]GL/AM++ [26]–[28]	D	C++	Yes	Yes	20+
STINGER [29], [30]	S	C	Yes [31]	Yes [31]	9
Tinkerpop	S/D	Java	Yes	Yes	0+
Urika/Cray Graph Engine [32]–[34]	S/M	Unknown	Yes	Yes	Unknown

[1] Complex applications, no built-in algorithms.
[2] Parallel architecture is indicated by **D**istributed, **S**hared, single **C**ore, highly **M**ultithreaded, or **G**PU.

TABLE I
SELECT RELATED SYSTEMS SUMMARY.

[13] A. Lugowski, D. Alber, A. Buluç, J. R. Gilbert, S. Reinhardt, Y. Teng, and A. Waranis, "A flexible open-source toolbox for scalable complex graph analysis," in *SIAM International Conference on Data Mining (SDM)*, Apr. 2012, pp. 930–941.

[14] A. Buluç, J. R. Gilbert, and A. Lugowsk, "Combinatorial blas library (mpi reference implementation)," http://gauss.cs.ucsb.edu/ aydin/CombBLAS/html/index.html, Apr. 2016.

[15] D. Nguyen, A. Lenharth, and K. Pingali, "A Lightweight Infrastructure for Graph Analytics," in *Proc. 24th ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 456–471.

[16] "The galois project," http://iss.ices.utexas.edu/?p=projects/galois, Apr. 2016.

[17] V. G. Castellana, A. Morari, J. Weaver, A. Tumeo, D. Haglin, O. Villa, and J. Feo, "In-Memory Graph Databases for Web-Scale Data," *Computer*, vol. 48, no. 3, pp. 24–35, Mar. 2015.

[18] "Giraph," Apr. 2016, http://giraph.apache.org.

[19] "Github repository," https://git-wip-us.apache.org/repos/asf/giraph.git, Apr. 2016.

[20] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: A Framework for Machine Learning and Data Mining in The Cloud," *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[21] "Github repository," https://github.com/dato-code/PowerGraph, Apr. 2016.

[22] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A High-performance Graph Processing Library on the GPU," in *Proc. 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2016, pp. 1–12.

[23] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger, "STAPL: An adaptive, generic parallel C++ library," in *Languages and Compilers for Parallel Computing*, ser. LNCS. Springer, Aug. 2001, vol. 2624, pp. 193–208.

[24] J. W. Berry, B. Hendrickson, S. Kahanz, and P. Konecny, "Software and algorithms for graph queries on multithreaded architectures," in *IPDPS '07: Proceedings of the 21st International Parallel and Distributed Processing Symposium*. Long Beach, CA: IEEE, March 2007.

[25] J. Berry and G. Mackey, "The multithreaded graph library," https://software.sandia.gov/trac/mtgl, Apr. 2016.

[26] N. Edmonds and A. Lumsdaine, "Parallel boost graph library 2 (active messages version)," http://www.crest.iu.edu/projects/pbgl2/, Oct. 2016.

[27] J. Willcock, M. Zalewski, and A. Lumsdaine, "Am++," http://www.crest.iu.edu/projects/am++/, Oct. 2016.

[28] D. Gregor and A. Lumsdaine, "The Parallel BGL: A Generic Library for Distributed Graph Computations," *Parallel Object-Oriented Scientific Computing (POOSC)*, vol. 2, pp. 1–18, 2005.

[29] D. Ediger, J. Riedy, D. A. Bader, and H. Meyerhenke, "Computational Graph Analytics for Massive Streaming Data," in *Large Scale Network-Centric Distributed Systems*, H. Sarbazi-Azad and A. Y. Zomaya, Eds. Wiley, 2013, pp. 619–648.

[30] D. A. Bader, J. Berry, A. Amos-Binks, D. Chavarría-Miranda, C. Hastings, K. Madduri, and S. C. Poulos, "STINGER: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation," Georgia Institute of Technology, Tech. Rep., May 2009, http://cass-mt.pnnl.gov/docs/pubs/pnnlgeorgiatechsandiastinger-u.pdf.

[31] "Github repository," https://github.com/stingergraph/stinger/tree/dev, Apr. 2016.

[32] "Cray graph engine (cge): Graph analytics for big data," http://www.cray.com/products/analytics/cray-graph-engine, Oct. 2016.

[33] "Urika-gx: The first agile analytics platform," http://www.cray.com/products/analytics/urika-gx, Oct. 2016.

[34] K. Maschhoff, R. Vesse, and J. Maltby, "Porting the urika-gd graph analytic database to the xc30/40 platform," in *Cray User Group Conference (CUG)*, 2015.