APPLIED RESEARCH LABORATORY FOR
INTELLIGENCE AND SECURITY

Technical Report

Benchmarking Hardware for Accelerating Intelligence and Security Graph Problems

Kent O'Sullivan, University of Maryland, osullik@umd.edu
Amir Ghaemi, Applied Research Laboratory for Intelligence and Security, aghaemi@arlis.umd.edu
Nandini Ramachandran, University of Maryland, nandinir@umd.edu
Vladimir Rife, Applied Research Laboratory for Intelligence and Security, vrife@umd.edu
William Regli, University of Maryland, regli@umd.edu
Project Github: https://github.com/UMD-ARLIS/Graph-Benchmarking-Project
ReadTheDocs: https://graph-benchmarking.readthedocs.io/en/latest/

January 8, 2024

Abstract

1 Introduction

Purpose. The purpose of this technical report is to describe the design and implementation of the *ARLIS Graph Hardware Acceleration Benchmark* (AGHAB).

Motivation. A recent analysis of the computing requirements of the United States (US) Intelligence and Security (I&S) community shows that many of the core analytic problems are formulated as graph processing problems [30]. Formally, a Graph $\mathcal{G}$ is a data structure with a set of nodes (vertexes) V and a set of connections between those nodes (edges) E such that $\mathcal{G} = \{V, E\}$. A common problem that the I&S community may use graphs to model is a social network graph, where each vertex is a person, and each node represents a social relationship between the two people (nodes) it connects. Intelligence graph datasets like social network graphs quickly grow to billions of edges. The worst-case performance of many graph processing algorithms is quadratic in the number of edges, meaning that the larger a graph becomes, the slower it is to process it.

In an environment where collection streams are constant, a *streaming graph* is continuously updated with new information. Additionally, to maintain compliance with legislative requirements, collected information may need to be purged after some time has passed. With new data changing the shape of a graph, and the requirement to delete data, the graph analysis becomes temporally bound, so timely graph processing of streaming graphs is critical.

The traditional approach to reduce the time required to compute large datasets is to parallelize them. Graph algorithms are difficult to parallelize, and in particular, are poorly suited for the distributed parallelism favoured for processing very large datasets because of the difficulty of selecting partition points, and the lack of spatial locality in graph data structures [3]. Shared memory parallelism approaches have been shown as viable on very large graphs for a wide collection of algorithms [9], but still perform suboptimally compared to other data structures of similar size because of their irregular structure, which manifests as memory-boundedness. That is, operations on graphs themselves are computationally simple, but because the information being processed is not necessarily proximal in memory traditional optimization approaches like caching and branch prediction are ineffective [1]. If caching is problematic, it follows that a processor that makes limited use of caching like a Graphic Processing Unit (GPU) should be a better choice for graph processing. However, GPUs are poorly suited for many graph problems because of the *sparsity* of graphs. *Sparsity* here means that when the graph is stored in memory (particularly as an adjacency matrix) there is a lot of 'empty space'. GPUs in particular are optimized for dense-matrix multiplication operations, so processing these graphs with a lot of 'empty space' means the matrices that represent them are sparse, not dense and so have a lot of wasted computation. The GUNROCK approach in particular is notable in GPU-based graph processing [33] but still suffers from resource under-utilization compared to operations on natively dense data structures, like images.

If the representation of graphs is the issue degrading performance, changing that representation to a more efficient one appears wise. The Graph Basic Linear Algebra Subprogram (GraphBLAS) project is an effort to

evolve the representation of Graphs and implementations of graph processing algorithms to leverage linear algebraic representations and primitive operations to improve the consistency, effectiveness and efficiency of graph processing [26, 25, 18, 19, 6]. Benchmarking of GraphBLAS against other approaches has shown it to execute slower than other approaches, likely due to the computational cost of the abstraction layer that makes it such an attractive option [2].

Related work comparing the original BLAS performance across different hardware systems suggests that Field Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuits (ASIC) can significantly accelerate linear algebraic operations [20, 34].

Exploring how to accelerate the processing of these large streaming graphs is one of the core goals of the Defense Advanced Research Projects Agency (DARPA) Hierarchical Identify Verify Exploit (HIVE) program, and relates to work undertaken in the DARPA Software Defined Hardware (SDH) program [31]. Many emerging hardware architectures optimized for graph processing are emerging, like EMU [10], Graphicianado [15], the Lincoln Labs architecture [32] and PiUMA [1] among others [29] will be similarly limited.

The proliferation of new hardware options presents an imperative for those acquiring and integrating graph hardware accelerators to select the best processor for the task at hand. There is currently no benchmark designed to enable the comparison of relevant I&S algorithms across hardware architectures, on graph problems that reflect the real-world problems that they will be expected to process. The AGHAB aims to fill the gap.

The remainder of this report outlines the architecture of the AGHAB (section 2), describes the benchmarks, reference implementations and telemetry in sections 3, 4 and 5 before providing an example usage of the benchmark (section 6) and summarizing contribution guidelines in section 7.

2 Benchmark Architecture

Our benchmark consists of *Datasets* specific to I&S problem domains, *Reference Implementations* of graph processing algorithms useful to the I&S community and a suite of tools to collect the *Telemetry* required to determine whether a given hardware platform is better suited for a specific graph processing task.

2.1 Design Philosophy

The AGHAB is driven by empirical methods in computer science [16, 17, 27]. Broadly, rather than developing a 'leaderboard' to identify which particular hardware accelerator is 'best', we aim to develop a platform to facilitate research exploring how the interactions of the problem datasets, reference implementations and hardware accelerators relate to the *effectiveness* and *efficency* of a problem. Here, *effectiveness* is the fitness for solving a problem and measures the correctness and quality of the solution produced by the reference implementation. The *efficency* measures the throughput, resource consumption, and execution speed. An optimal solution is efficient and effective, but most approaches accept degradation in one for improvement in the other. The AGHAB is designed to be transparent, reproducible and portable.

2.2 Operational View 1

Figure 2.2 shows the high-level use case for the AGHAB. An evaluation using AGHAB begins with downloading the benchmark from the ARLIS GitHub. They unpack the benchmark datasets, reference implementations and telemetry suite, and define experiments to run. An *experiment* is a single run combining a dataset, reference implementation and hardware platform. The experiment specification drives the experiment framework, controlling the compilation, dataset generation, instrumentation, execution, telemetry and reporting of the benchmark system. Related experiments are collected into an *experiment suite*. An evaluator can define their own compiler and other external dependencies in the experiment specification to facilitate deploying the benchmark on their hardware. Once the experiment suite is complete, the evaluator receives a report created from the collected telemetry, and the system adds the results to the records.

2.3 Operational View 2

The three components of the AGHAB shown in figure 2.2 control variance in the datasets, implementation and measurement so that the focus of the variable under exam remains the hardware platform. The experiment defines the dataset(s) to be tested against. The datasets are usually pre-generated but may be dynamically generated if required. The experiment defines the reference implementation to use, and facilitates compilation where required. If passes the dataset to the reference implementation, and evaluates the output against the ground-truth to generate

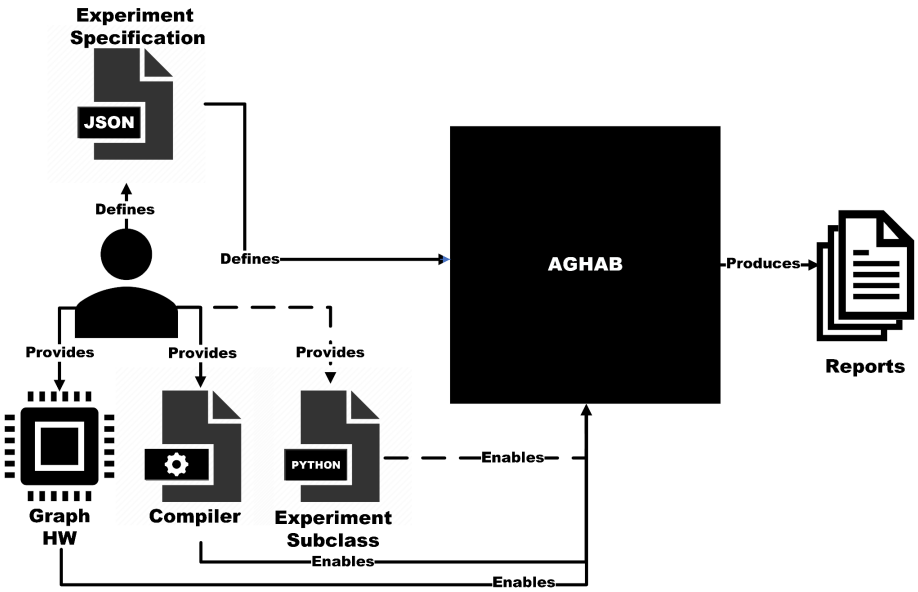

Figure 1: A user of the AGHAB provides their hardware, optionally any required compiler and a specification of the experiments to run.

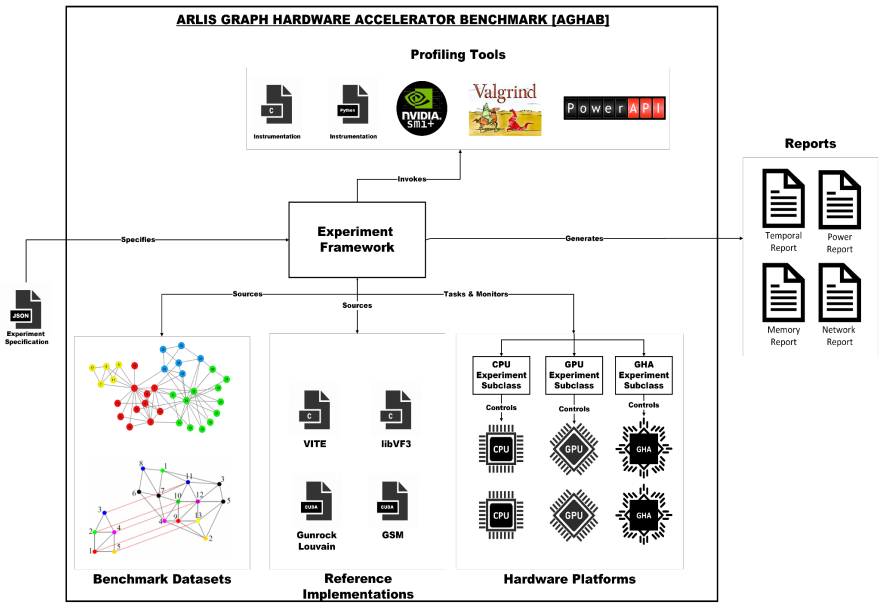

Figure 2: The Experiment Framework reads the experiment specification and collects the specified datasets, the reference implementations and creates an experiment to task the baseline CPU, GPU and Graph Hardware Accelerators (GHA). It controls the data collection and report generation.

the measures of *effectiveness*. The experiment framework initializes the telemetry suite, and controls the collection, fusion and analysis of telemetry. The results of the analysis are passed to the reporting module, which generates reports for the evaluator, and posts results to the system for tracking.

3 Benchmark Data

Benchmark datasets are the 'problems' that must be solved. Problems should vary in complexity and cover the range of tasks expected of the accelerators 'in the real world'. Datasets can be synthetic, real-world or somewhere in between. Previaling wisdom says that real-world datasets shold be used for graph problems, because of the intrinsic relationship between the shape of the input graph and the computational complexity of the problem [4]. Most of these datasets are sourced from the Stanford Network Analysis Project (SNAP) [22], an excellent resource for general-purpose graph datasets. Ideally, we would use real I&S datasets, however classification and privacy concerns preclude the use of real data. The best alternative sees us distil I&S datasets to their statistical properties and using those properties to generate graphs that mimic the properties of the original graph while enforcing confidentiality of the source data. Research in generating graphs from statistical properties is ongoing Kent: cite as are other privacy-preserving techniques Kent: cite Without access to real data, or a realistic proxy, we must decide whether to use synthetic graphs, general-purpose graph datasets or both. The advantage of synthetically generated datasets is that, carefully designed, they offer a controlled environment for testing the various structural, algorithmic and computational stressors, while preventing the inadvertant over-fitting of solutions to benchmark problems [16, 17]. Random graphs like Kent: Erdos-R'enyi are used extensively for theoretical analysis of graph algorithims, but typically generate results inconsistent with real-world datasets where factors like mean degree, degree skew and diameter. Many synthetic graphs generators exist, but because of the varying requirements of the algorithms that they test, they are typically domain-constrained. That is, there is no general-purpose generator that can create graphs suitable to evaluate the *effectiveness* and *efficency* of all graph algorithms. So, we must generate graphs specific to the problem being solved to evaluate the effectiveness of the solution. The two problems we examine are *Community Detection* and *Subgraph Matching*.

3.1 Community Detection

We want a suite of problems that vary from 'easy' to 'hard'. The difficulty range applies to both the *effectiveness* and *efficency* dimensions. While they often overlap, they are sometimes disjoint.

Based on a review of the community detection literature over the last three decades, we identify the features of a community detection problem that make them 'easy' or 'hard'. Some reflect general graph properties, others are domain specific.

'Easier' graphs have smaller node counts, are sparser and communities are well-defined cliques. 'Harder' graphs have larger node counts, are denser and have 'fuzzy' communities. 'Harder' graphs have heterogeneous communities where both the number and size of communities follow power-law distriibutions. 'Harder' graphs have overlapping communties, are dynamic and feature no further information beyond the topology of the graph. Many of the newer approaches use additional information sources like edge weights, node labels or external constriants to improve community detection results.

So, when paramaterized, we need to be able to control the number of nodes, the mean degree density, the distribution of community size, the fuzziness of communities and the inclusion of metadata.

Several community detection dataset generators exist, with the Girvan-Newman Generator [14] originating the idea of community detection problems. It is limited in only producing random graphs. The Lancichinetti Fortunato Radicchi (LFR) generator remains the defacto standard for generating community detection problems [21]. The Advantage of LFR is that it is available in common graph libraries, but it is brittle to variance in parameters when solving a graph structure, making automated paramaterization difficult and is not able to include metadata used by many newer approaches. The newer acMark [24] and GenCAT [23] generators offer control over these metadata additions but are brittle in their implementation, mostly only available as jupyter notebooks.

Kent: our approach takes...

3.2 Subgraph Matching

4 Reference Implementations

4.1 Requirements

Reference implementations were selected to maximize the following selection criteria:

1. Relevance to intelligence and security,
2. Is or was SOTA,
3. Intrinsic telemetry,
4. Cross-architecture availability,
5. Ease of integration, and
6. Community acceptance and support.

4.2 Problem Domains

Two problem sets, community detection and subgraph matching are the chosen graph problems relevant to the I&S domain. Both are NP-Complete problems with real-world applications, and so balance the complexity and utility for a useful benchmark.

4.2.1 Community Detection

The Community Detection problem asks: given a graph, are there elements of that graph more tightly connected than other parts? The goal is to uncover the inherent organizational structure or modular patterns within complex systems such as social networks, biological networks, or information networks. In a real-world setting, a military force undertaking peacekeeping might want to understand the population demographics of a region. They could analyze to determine what the local cliques, tribes, or groups are and then determine how those groups interact with each other, which are likely to be friendly and which are likely to be hostile.

4.2.2 Subgraph Matching

In terms of a graph, subgraph matching asks, given a query graph and a target graph, are there any instances of that query graph contained within the target graph? In this context, the pattern graph represents a specific structure or motif that the algorithm seeks to identify within the larger graph. The goal is to locate all instances where the pattern occurs as a connected subgraph in the target graph. In a real-world context, we might know that a precursor to a terrorist attack is for a cell member to visit three different hardware stores and buy a particular item at each one. Knowing the pattern, we could monitor credit card transactions to observe if any single credit card makes that purchasing pattern and flag it for investigation. Efficiently solving the subgraph matching problem is computationally challenging, especially for large graphs, and various algorithms and heuristics are employed to address this problem in different application domains.

4.3 Reference Implementations

Ideally, a single code base will be recompiled to automatically optimize for each hardware architecture – the GraphBLAS suite offers this layer of abstraction as it reaches maturity and porting the implementations to GraphBLAS is important future work.

Currently, the AGHAB uses widely accepted parallel implementations specific to each architecture (CPU, GPU, etc), aiming to identify where the same algorithm is optimized to run on different hardware, rather than forcing all hardware to run the same code. The serial performance is derived by running the parallel implementation with a single worker node.

The AGHAB Experiment Framework abstracts away the operation of each implementation. Instead, an experimenter specifies the number of workers, the datasets to use, the metrics to generate and the implementations to run, and the experiment framework checks the compilation status, routes the instructions and collects the results.

4.3.1 Community Detection – The Louvain Algorithm

The Louvain algorithm is a widely used heuristic approach for community detection with numerous implementations available [5]. It optimizes modularity, a metric that quantifies the quality of community assignments based on the density of connections within communities compared to those between communities [28]. The Vite [11, 12, 13] (CPU) and Gunrock [33] (GPU) implement The Louvain algorithm in AGHAB, with minor changes.

Vite Vite is a Message Passing Interface (MPI)+Open Multi-Processing (OpenMP) distributed memory parallel implementation of the Louvain method for community detection developed by Washington State University and the Pacific Northwest National Laboratory[1]. We only use it on a single node, to emulate shared memory parallelism. It accepts input graphs in most of the common formats, converting them into a binary format that the Vite Louvain implementation uses.

[1] `https://github.com/ECP-ExaGraph/vite`

Gunrock Louvain Gunrock is a specialized graph-processing library designed for GPU acceleration created by the University of California at Davis for the HIVE program [33]. We use the Gunrock implementation of the Louvain algorithm from the dev branch of version 1.2 of the Gunrock library[2]. It was not part of the stable release, so the AGHAB version has minor modifications to support compilation and execution. The Gunrock library expects the Matrix Market Graph Format (MTX) as input, and graphs stored in other formats are converted to MTX before processing. The process of cloning the original Gunrock repository, compiling it, gathering data input, running the algorithm, and extracting metrics has been automated and abstracted using the benchmarking framework.

4.3.2 Subgraph Matching – The VF3 Algorithm

The VF3 [8] and VF3P [7] algorithms are among the leading heuristic approaches to Subgraph Matching currently available for CPU. The GPU implementation uses a custom algorithm, Gunrock Subgraph Matching (GSM) [33], to better leverage the capabilities of a GPU, but we note is inspired by VF3.

VF3LIB vf3lib is a software library created by Mivia Lab containing all the currently published versions of VF3, an algorithm to solve subgraph isomorphism[3]. This library contains both serial and parallel implementations that can be run on the CPU. Both versions require input graph files in a binary format that input graphs must be converted to before processing.

Gunrock Subgraph Matching GSM is BFS-based, allowing it to leverage the parallel processing capabilities of the GPU. We use the implementation from the dev branch of version 1.2 of Gunrock[4] in AGHAB GSM borrows ideas from VF3, specifically during preprocessing when precomputing the query nodes‘ order based on the Maximum Likelihood Estimation (MLE) method.

5 Telemetry

The purpose of the benchmark suite is to measure the dimensions of time, memory, throughput, power, and network utilized by each hardware platform to facilitate deeper study.

5.1 Metrics

5.1.1 Temporal Metrics

The *time* dimension should cover both execution time for the algorithm, as well as the Extract-Transform-Load (ETL) time required to prepare the input graph for analysis.

5.1.2 Throughput Metrics

Throughput counts the number of graph nodes that can be processed in a given time box, and is measured as Traversed Edges Per Second (TEPS)

5.1.3 Memory Measurement

Many graph applications are memory-bound [1, 3] and so understanding memory latency, Last Level Cache (LLC) miss rates, peak memory usage, spatial locality and temporal locality offer important insights into the operation of the underlying hardware given a dataset and algorithm, helping to describe whether memory acceleration is required and useful.

5.1.4 Network Metrics

In a distributed setting the bottleneck in performance may be the latency sending messages between worker nodes across the network. Measuring latency and round-trip time allows us to estimate the impact of the network on performance, and judge if network acceleration is required and useful.

5.1.5 Power Metrics

The bounding factor for many computing applications is the Size, Weight and Power (SWaP). Perhaps a system is limited to a small form factor, and so can only access a limited amount of power, and weight when processing graph data. Alternately, power is an ongoing sustainment cost. Budgetary constraints and fiscal responsibility encourage us to minimize the amount of power used to complete a task. The measure of consumed Watts (W) is a leveling metric that is used in conjunction with other metrics to estimate the hardware efficiency (e.g. TEPS per Watt, Watts per minute).

[2] https://github.com/gunrock/gunrock/tree/dev/examples/louvain
[3] https://github.com/MiviaLab/vf3lib
[4] https://github.com/gunrock/gunrock/tree/dev/examples/sm

5.2 Measurement

Measuring *Time* uses wall and processor timers in the experiment framework to record the ETL and execution times. Measuring *Throughput* requires instrumentation of the underlying code, to count the number of 'traversal' actions. AGHAB currently uses a mix of instruction counting and in-code counters to track the node 'traversals'. AGHAB uses Valgrind to measure a workload's peak and total memory consumption. Because of the substantial degradation to execution time introduced by Valgrind, it has proven infeasible for us to profile memory consumption above the *Medium* size datasets. We initially intended to measure the spatial and temporal locality of memory. We discovered that these factors are highly dataset-dependent. Still, we believe that characterizing the locality of a given workload remains important to quantifying potential performance differences between the hardware architectures. To measure power we are implementing the Open-Source PowerAPI to measure the power consumption of the benchmark tests[5] To measure network we Kent: Add here

5.3 Reporting

The AGHAB produces reports on a per-experiment basis and can generate comprehensive reports across multiple experiments. The reports are automatically generated by the experiment framework and summarize the metrics extracted from each experiment run.

6 Benchmarking Example

7 Contributing

The AGHAB is designed as an extensible framework for benchmarking graph hardware accelerators. The reference CPU and GPU implementations are designed to provide points of reference for an accelerator device under test (DUT). We envision contributions in two forms. First, is deploying the AGHAB on different CPU and GPU baseline hardware, using the datasets and reference implementations provided. Second is adding a graph hardware accelerator to the AGHAB.

[5] `https://powerapi.org/`

7.1 Baseline Hardware Extension

Each deployment of AGHAB requires CPU and GPU available to generate the relevant baselines. To port between hardware, the experimenter need only provide the relevant compiler for their hardware as a parameter for the experiment framework. Currently, the GPU code from the Gunrock project is in CUDA, so is limited to NVIDIA devices.

7.2 Graph Hardware Accelerator Extension

To add a new hardware accelerator to the AGHAB, the experimenter will need to create a new subclass of the *Experiment* class in the Experiment Framework. The *Experiment* class structure automates the inclusion of hardware into the experiment pipeline, and requires detail about the location of compilers, translating the experiment parameters into the parameters expected by the DUT and routing any internal telemetry back to the reporting functions. Detail on constructing a new *Experiment* subclass is in the ReadTheDocs for the project. Should the Graph Hardware Accelerator require a reference implementation distinct from the provided reference implementations, it will also have to be provided by the experimenter and placed in the *graph_problems* directory and included in the new experiment subclass.

8 Conclusion

The ARLIS Graph Hardware Accelerator Benchmark (AGHAB) provides the framework to evaluate the impact of incorporating graph hardware accelerators into Intelligence and Security (I&S) workflows. Specifically, it serves as a mechanism to decide which, if any, accelerator is suitable to improve the speed, throughput or resource consumption of a graph processing task. It provides a platform, and a framework for further empirical study of graph-basedanalytics in time and resource-constrained environments. The AGHAB is extensible, and designed to serve as the abstraction layer between the experimenter and the hardware. The experimenter need only provide the hardware, relevant compilers and experiment definitions to link a new graph hardware accelerator into AGHAB. Then, it can leverage the datasets, analytics, reference implementations and baselines built-in to the experimental framework to run experiments, collect data and

generate reports. AGHAB is in its protoform at present, but continuing work will create a platform to evaluate the rise of Graph-Based Artificial Intelligence, and the impact of hardware acceleration on Graph-Based AI into the future.

9 References

[1] Sriram Aananthakrishnan et al. "PIUMA: programmable integrated unified memory architecture". In: *arXiv preprint arXiv:2010.06277* (2020). URL: https://arxiv.org/pdf/2010.06277.

[2] Ariful Azad et al. "Evaluation of graph analytics frameworks using the gap benchmark suite". In: *2020 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2020, pp. 216–227.

[3] Scott Beamer, Krste Asanovic, and David Patterson. "Locality exists in graph processing: Workload characterization on an ivy bridge server". In: *2015 IEEE International Symposium on Workload Characterization*. IEEE. 2015, pp. 56–65.

[4] Scott Beamer, Krste Asanović, and David Patterson. "The GAP benchmark suite". In: *arXiv preprint arXiv:1508.03619* (2017). URL: https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Beamer2017.pdf.

[5] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008. URL: https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Blondel2008.pdf.

[6] Benjamin Brock et al. "Introduction to graphblas 2.0". In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2021, pp. 253–262.

[7] Vincenzo Carletti et al. "A parallel algorithm for subgraph isomorphism". In: *Graph-Based Representations in Pattern Recognition: 12th IAPR-TC-15 International Workshop, GbRPR 2019, Tours, France, June 19–21, 2019, Proceedings 12*. Springer. 2019, pp. 141–151.

[8] Vincenzo Carletti et al. "Introducing VF3: A new algorithm for subgraph isomorphism". In: *Graph-Based Representations in Pattern Recognition: 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16–18, 2017, Proceedings 11*. Springer. 2017, pp. 128–139.

[9] Laxman Dhulipala, Guy E Blelloch, and Julian Shun. "Theoretically efficient parallel graph algorithms can be fast and scalable". In: *ACM Transactions on Parallel Computing (TOPC)* 8.1 (2021), pp. 1–70.

[10] Timothy Dysart et al. "Highly scalable near memory processing with migrating threads on the Emu system architecture". In: *2016 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3)*. IEEE. 2016, pp. 2–9.

[11] Sayan Ghosh et al. "Distributed Louvain Algorithm for Graph Community Detection". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2018, pp. 885–895. DOI: 10.1109/IPDPS.2018.00098. URL: https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Ghosh2018.pdf.

[12] Sayan Ghosh et al. "Scalable distributed memory community detection using vite". In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE. 2018, pp. 1–7. URL: https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Ghosh2018a.pdf.

[13] Sayan Ghosh et al. "Scaling and quality of modularity optimization methods for graph clustering". In: *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2019, pp. 1–6. URL: https://github.com/osullik/summer2023/blob/main/Papers/bibliography/reference_papers/Ghosh2019.pdf.

[14] Michelle Girvan and Mark EJ Newman. "Community structure in social and biological networks". In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826.

[15] Tae Jun Ham et al. "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2016, pp. 1–13.

[16] John N Hooker. "Needed: An empirical science of algorithms". In: *Operations research* 42.2 (1994), pp. 201–212.

[17] John N Hooker. "Testing heuristics: We have it all wrong". In: *Journal of heuristics* 1 (1995), pp. 33–42.

[18] Jeremy Kepner et al. "Graphs, matrices, and the GraphBLAS: Seven good reasons". In: *Procedia Computer Science* 51 (2015), pp. 2453–2462.

[19] Jeremy Kepner et al. "Mathematical foundations of the GraphBLAS". In: *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2016, pp. 1–9.

[20] Srinidhi Kestur, John D Davis, and Oliver Williams. "Blas comparison on fpga, cpu and gpu". In: *2010 IEEE computer society annual symposium on VLSI*. IEEE. 2010, pp. 288–293.

[21] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. "Benchmark graphs for testing community detection algorithms". In: *Physical review E* 78.4 (2008), p. 046110.

[22] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[23] Seiji Maekawa et al. "GenCAT: Generating attributed graphs with controlled relationships between classes, attributes, and topology". In: *Information Systems* 115 (2023), p. 102195.

[24] Seiji Maekawa et al. "General generator for attributed graphs with community structure". In: *Proceeding of the ECML/PKDD graph embedding and mining workshop*. 2019, pp. 1–5.

[25] Tim Mattson et al. "LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS". In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2019, pp. 276–284.

[26] Tim Mattson et al. "Standards for graph algorithm primitives". In: *2013 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2013, pp. 1–2.

[27] Catherine C McGeoch. "Toward an experimental method for algorithm simulation". In: *INFORMS Journal on Computing* 8.1 (1996), pp. 1–15.

[28] Mark EJ Newman and Michelle Girvan. "Finding and evaluating community structure in networks". In: *Physical review E* 69.2 (2004), p. 026113.

[29] Biagio Peccerillo et al. "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives". In: *Journal of Systems Architecture* 129 (2022), p. 102561.

[30] William Regli et al. "Operationalizing Emerging Hardware for AI Applications: A Survey of Transition Opportunities and Datasets". In: (2022).

[31] William Regli et al. "Software-Defined Hardware:A Study of Emerging Hardware for AI Applications". In: (2022).

[32] William S Song et al. "Novel graph processor architecture, prototype system, and results". In: *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2016, pp. 1–7.

[33] Yangzihao Wang et al. "Gunrock: A high-performance graph processing library on the GPU". In: *Proceedings of the 21st ACM SIGPLAN symposium on principles and practice of parallel programming*. 2016, pp. 1–12.

[34] Chong Xiong and Ning Xu. "Performance comparison of BLAS on CPU, GPU and FPGA". In: *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*. Vol. 9. IEEE. 2020, pp. 193–197.

10 Glossary

AGHAB ARLIS Graph Hardware Acceleration Benchmark
ASIC Application Specific Integrated Circuits
ARLIS Applied Research Laboratory for Intelligence and Security.
BFS Breadth-First Search
CPU Central Processing Unit
DARPA Defense Advanced Research Projects Agency
DUT Device Under Test **ETL** Extract-Transform-Load
FPGA Field Programmable Gate Array
GPU Graphics Processing Unit
GSM Gunrock Subgraph Matching
HIVE Hierarchical Identify Verify Exploit
KGA Knowledge Graph Analytics
LLC Last Level Cache
MLE Maximum Likelihood Estimation
MPI Message Passing Interface
MTX Matrix Market Format
OpenMP Open Multi-Processing
PIUMA Programmable Integrated Unified Memory Architecture
SGM Sub Graph Matching
SWaP Size, Weight and Power **TEPS** Traversed Edges Per Second
TEPS/W Traversed Edges Per Second Per Watt