GPU Architectures in Graph Analytics: A Comparative Experimental Study

Peichen Xie
Wuhan University
Wuhan, China
xpc2000@whu.edu.cn

Zhigao Zheng*
Wuhan University
Wuhan, China
zhengzhigao@whu.edu.cn

Yongluan Zhou
University of Copenhagen
Copenhagen, Denmark
zhou@di.ku.dk

Yang Xiu
Wuhan University
Wuhan, China
xiuyang@whu.edu.cn

Hao Liu
Shanghai Jiao Tong University
Wuhan Digital Engineering Research Institute
Shanghai, China
liuhao2020@sjtu.edu.cn

Zhixiang Yang
CSSC Lingjiu Hi-Tech (Wuhan) Co., LTD
Wuhan, China
yzhx_cq@163.com

Yu Zhang
Huazhong University of Science and Technology
Wuhan, China
zhyu@hust.edu.cn

Bo Du*
Wuhan University
Wuhan, China
dubo@whu.edu.cn

ABSTRACT

In recent years, there has been a lot of focus on developing graph analytics algorithms that utilize the high parallelism of GPUs to speed up graph analytics tasks. Meanwhile, the two main GPU manufacturers, NVIDIA and AMD, have made different design decisions that can impact the performance of graph computation. In this paper, we seek to understand the effects of those decisions through experimentation. Since there is currently no graph analytics software for the ROCm-like platform used by AMD-like GPUs, we have created adGRAPH by porting nvGRAPH, a mature graph analytics library optimized for CUDA, to ROCm-like platforms. AdGRAPH[1] allows us to use AMD-like GPUs to accelerate graph analytics and compare the performance of the two types of GPUs.

We tested the performance of several commonly used graph algorithms of varying complexities on two NVIDIA GPUs, A100 and V100, and two AMD-like GPUs developed in China, Z100 and Z100L. Through thorough experiments, we discovered that while NVIDIA GPUs perform better on complex graph analytics algorithms, thanks to their SIMT paradigm, the larger warp size and independent shared memory of AMD-like GPUs make them more efficient than NVIDIA GPUs for graph algorithm implementations with lower branching complexity.

1 INTRODUCTION

Graphs are fundamental data structures that play a crucial role in simulating and analyzing relational data due to their ability to represent complex relationships with flexibility [1]. Graphs are extensively utilized in various applications such as social network analysis, recommendation systems, biological modeling, and traffic optimization. They demonstrate an inherent ability to represent and capture intricate connections.

High-performance graph analytics is an emerging focus of academic research and a crucial element of practical solutions, helping organizations make data-driven decisions, uncover hidden knowledge, and achieve research breakthroughs or competitive advantages. In recent years, the volume of real-world data has increased exponentially, creating a high demand for high-performance graph analytics tools and technologies. Taking social networks as an example, the number of users on platforms like Twitter and Instagram has surpassed one billion, and the number of connections between likes and followers is even greater. The scale and complexity of graph data in these fields necessitate efficient computational methods.

A promising direction is the incorporation of graph processing units (GPUs) into graph analytics workflows. Originally designed for graphic rendering, GPUs have evolved into highly parallel and large-scale processors, proficient at managing data parallel tasks [2]. Therefore, GPUs have become powerful tools for accelerating graph analytics. However, a significant challenge of using GPUs to speed up graph algorithms is that most graph algorithms access memory irregularly, leading to low memory access efficiency. Moreover, the high variance of the vertex degrees leads to load imbalance, hindering the efficiency of GPU's parallel computation. Therefore, leveraging the GPU's memory bandwidth and computing power for graph analytics is challenging. To address this challenge, it is crucial to optimize both graph algorithms and their implementations and explore hardware-level optimization.

The hardware architecture and characteristics of GPUs can impact the performance of different graph algorithms. Certain algorithms may benefit from parallelism on specific GPUs, while others may need optimizations to run efficiently on different GPUs. An in-depth understanding of GPU characteristics and their impacts on graph algorithms provides valuable information not only for optimizing graph systems and algorithms for GPUs but also for developing hardware for accelerating graph analytics. Hence, such a study is crucial to advance this research direction.

*Zhigao Zheng, Bo Du are the corresponding authors (zhengzhigao, dubo@whu.edu.cn).
[1]The source code of adGRAPH is open and available at https://anonymous.4open.science/r/adGRAPH-E319

To close this gap, this paper provides an in-depth comparison between two popular types of GPU architecture: NVIDIA GPUs and AMD-like GPUs. Our experimentation includes representative GPUs of each architecture, namely A100 and Z100L. A100 is one of NVIDIA's most successful GPGPU products, widely used in AI and high-performance computing. Z100L is a relatively mature GPGPU developed by a well-known Chinese company[2] based on authorization from AMD. In addition, we have added experiments on two older GPUs, NVIDIA's V100 and the AMD-like Z100, to provide more experimental evidence, especially regarding the scalability and generalizability of the findings. To establish a practical benchmark, we adopt nvGRAPH, a graph analytics library developed by NVIDIA, which includes parallel algorithms for high-performance graph analytics. As there is a general lack of graph analytic systems or libraries on AMD-like GPUs, we ported nvGRAPH to Z100L and Z100 to enable a level-ground comparison between the GPUs. We provided both the overall performance metrics as well as various detailed profiling metrics to reveal the differences between the two GPUs in terms of running graph algorithms.

In summary, our contributions are as follows:

- We analyze the differences between the two existing mainstream GPU architectures and their possible impacts on large-scale graph analytics.
- We implemented adGRAPH, a ported version of nvGRAPH, to allow generic graph analytics algorithms to be accelerated by AMD-like GPUs. We are making adGRAPH available as open-source software, which will allow others not only to reproduce the experiments described in this paper but also to adopt AMD-like GPUs for real-world graph applications.
- We conducted a set of experiments to evaluate graph analytics performance on four GPUs: V100, Z100, A100, and Z100L, and reported various metrics to provide insights into how the choice of GPU architecture can impact the efficiency of graph analytics.
- The experimental results provide evidence for several interesting findings and insights. It has been observed that some graph traversal algorithms, such as BFS, work more efficiently on AMD-like GPUs using adGRAPH as compared to NVIDIA GPUs using nvGRAPH, while algorithms with more complex branching have a better performance on NVIDIA GPUs. Additionally, it has been noted that only when the branching complexity of graph algorithms is low can the larger GPU warp size and independent shared memory of AMD-like GPUs significantly enhance their execution efficiency.

This paper is organized as follows: In Section 2, we present related work and background, including architectural differences between NVIDIA GPUs and AMD-like GPUs. In Section 3, we discuss existing libraries of high-performance graph analytics on GPUs and explain the reason for choosing nvGRAPH to be used in the experiments. Then we present our work on porting nvGRAPH for an AMD-like GPU named adGRAPH. Section 4 compares the performance of various graph algorithms on NVIDIA's A100 and V100, and AMD-like Z100 and Z100L to assess the impact of different GPU architectures on high-performance graph analytics. Additionally, we conduct a more in-depth discussion through fine-grained profiling. Finally, we further discuss the insights from the experimental results in Section 5 and conclude our study in Section 6.

[2]We are not allowed to mention the name of GPUs, ROCm-like Toolkit and company because of the agreement with the GPU provider. The Z100 and Z100L used in our experiments are matured commercial products and have been widely applied in data centers across China.

2 BACKGROUND

In this paper, we selected the most common NVIDIA and AMD-like GPU architectures for evaluation. In this section, we first present the related work, followed by a presentation of the two GPU architectures and their respective strengths, weaknesses, and expected impacts on graph analytics.

2.1 Related work

Processing large-scale graphs, e.g., with billions of vertices and edges, is challenging in computation power. Employing GPUs to accelerate graph processing is a promising solution due to their parallelism and high memory bandwidth. However, researchers face several challenges in using GPUs to accelerate large-scale graph computations. First, graph computations often exhibit irregular data access patterns [3]. Therefore, it is difficult to parallelize them efficiently with GPU architectures. Furthermore, the limited memory of GPUs compared to CPU may become a potential performance bottleneck. Finally, the condition branches (like an if-else statement) in graph computations do not fully utilize the high parallelism offered by GPU, which leads to branch divergence and may dramatically degrade the performance [4].

Existing studies on general graph processing systems and optimization strategies of graph analytics on GPUs can be categorized into four main directions. Some studies, such as GraphIt [5] and SIMD-X [6], have investigated programming models for graph processing on GPUs. Other studies like C-SAW [7] and Real-GraphGPU [8] have focused on workload mapping and thread management. Additionally, some other studies, such as Frog [9] and GunRock [10], have concentrated on optimizing memory access. Furthermore, research like EMOGI [11], DiGraph [12], EvoGraph [13] and DBR [14] has targeted specific optimizations for certain scenarios, such as BFS, graph traversal and path-based Iteration.

Despite the abundant studies of graph processing on GPUs mentioned above, there is a lack of investigation into the differences in GPU architectures and their impacts on computation efficiency. Non-NVIDIA GPUs were only briefly introduced in the literature [15–17] when necessary. With the increased attention to GPU options other than NVIDIA, there are some recent studies on performance portability [18] of parallel programs on AMD and NVIDIA GPUs. However, these studies have mostly focused on tasks of deep learning [19], which exhibit significant differences from graph analytics. For example, the weak locality problem of graph analytics does not exist in deep learning workloads. Consequently, previous conclusions cannot be directly applied to high-performance graph analytics on GPUs.

2.2 NVIDIA GPU

NVIDIA, as a prominent figure in the field of parallel accelerators like GPUs, has sequentially introduced the Fermi, Kepler, Volta, Turing, and Ampere architectures. In this section, a brief introduction to NVIDIA's GPU architecture is provided [20], shown in Figure 1.

To support large-scale parallel computing, an NVIDIA GPU is equipped with a substantial number of Streaming Multiprocessors (SM), which are the fundamental components of the GPU. Each SM contains multiple Stream Processor Cores, specialized

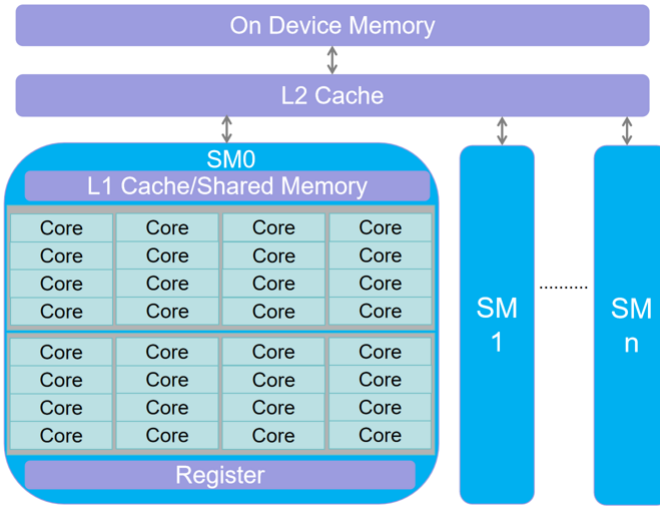

Figure 1: NVIDIA GPU Architecture Diagram

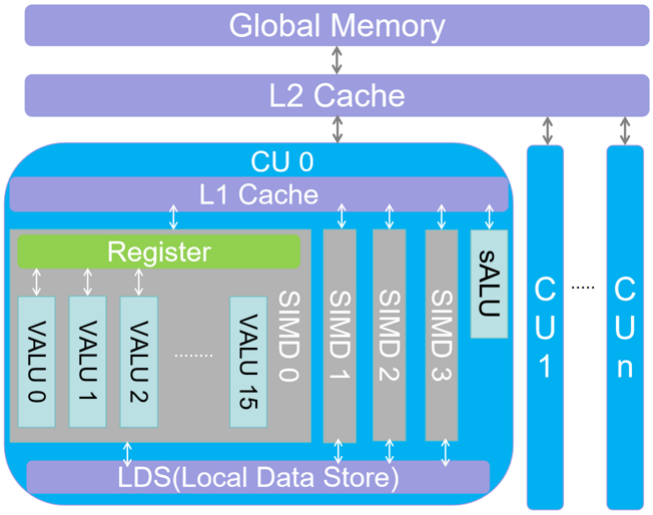

Figure 2: AMD-like GPU Architecture Diagram similar to AMD GCN [22]

functional units, registers, double precision units, and a thread scheduler. GPU utilizes its large number of Cores to handle many computing tasks simultaneously, achieving parallel computing efficiency far beyond CPUs. In NVIDIA's thread hierarchy, 32 threads make up one warp, several warps form a block, and multiple blocks compose a grid. Moreover, NVIDIA GPUs adopt the SIMT (Single Instruction Multiple Threads) paradigm. With SIMT, multiple threads in an NVIDIA GPU execute the same instruction asynchronously on different data without the need to organize data into particular vector lengths, and each thread is allowed to have different logical branches. So, it enables each thread to execute a conditional branch statement in parallel according to the situation, making SIMT more flexible than SIMD (Single Instruction Multiple Data).

The storage units in NVIDIA GPUs can be classified into two hierarchies based on their location: on-device memory and on-chip memory. On-device memory consists of DRAM, which primarily includes global memory with large capacity but high latency. On-chip memory comprises register, shared memory (a specialized cache in GPUs that facilitates data sharing and collaboration between threads), L1 cache (which shares the same space with shared memory), and L2 cache. While on-chip memory has limited capacity, it offers exceptionally fast memory access speeds.

2.3 AMD-like GPU

AMD has also launched GPU architectures such as TeraScale, VLIW, GCN, RDNA, and CDNA. The AMD-like GPU utilized in our study bears a resemblance to the AMD GCN (Graphics Core Next) [21] accelerator. In this section, we briefly introduce an AMD-like GPU architecture launched by a Chinese manufacturer who has previously collaborated with AMD, as depicted in Figure 2.

AMD-like GPUs employ a parallel microarchitecture that offers a robust platform for not only computer graphics applications but also for general data-parallel applications. The component unit of an AMD-like GPU is referred to as the "Compute Unit" (CU), analogous to the SM in NVIDIA GPU architecture. Additionally, each CU is comprised of four SIMD units and equipped with data registers and various auxiliary functional units, collectively forming a comprehensive computing module. Each SIMD unit serves as the primary computing core component of the CU, which corresponds to the Core in NVIDIA GPU. Each SIMD unit contains numerous VALU (Vector Arithmetic logic Units) and supports ten wavefronts, with each wavefront having a size of 64, in contrast to the warp size of 32 in NVIDIA GPUs. The VALU within an SIMD unit adopt the SIMD (Single Instruction Multiple Data) paradigm. In this paradigm, all threads within one SIMD can synchronously execute the same instruction on different data, but cannot concurrently execute different conditional branch statements.

The storage hierarchy of AMD-like GPUs is basically comparable to Nvidia's, except for the Local Data Store (LDS), which has the same functionality of the shared memory in NVIDIA GPUs but is separated from the L1 cache in AMD-like GPUs.

2.4 Architectural Differences

The architectural distinctions between AMD-like GPUs and NVIDIA GPUs can be summarized as follows:

- AMD-like GPUs adopt the SIMD (Single Instruction, Multiple Data) paradigm, while NVIDIA's GPUs adopt the SIMT (Single Instruction, Multiple Thread) paradigm.
- Warpsize or wavefront size in AMD-like and AMD GPUs is 64, which is twice that of NVIDIA GPUs.
- The local data store (LDS) and L1 cache in AMD-like GPUs are completely separated including data paths, unlike the corresponding shared memory in the NVIDIA GPU storage hierarchy.

2.5 Strengths, Weaknesses, and Expected Impacts

The aforementioned architectural choices have their respective advantages, disadvantages, and expected impacts on performance.

The larger wavefront size of AMD-like GPUs means that threads on them have theoretically higher parallelism than NVIDIA GPUs. Moreover, the separated LDS of AMD-like GPUs, which serve similar functions as shared memory in NVIDIA GPUs, allows the L1 cache and LDS to have independent data transfer paths [23], meaning that LDS's bandwidth is not affected by the data transfer of L1 cache. Previous experimental studies [24] have shown that non-shared cache designs, like those

in NVIDIA's Maxwell GPUs, significantly outperform shared designs regarding bandwidth utilization and other performance metrics of shared memory. The independence of the shared memory is a major factor contributing to this performance difference. However, this design comes with a drawback: it results in higher access latency between the LDS (or shared memory) and L1 cache.

The use of SIMT by NVIDIA GPUs implies that compared to AMD-like, they have higher flexibility during program execution, making them more suitable for handling parallel tasks with complex program branches. The L1 cache and shared memory of NVIDIA GPUS bring a lower data-transmission latency between the two components with the cost of potential mutual interferences between the L1 cache and shared memory, especially when the one occupies so much bandwidth that it hinders the operations on the shared memory.

The expected impacts discussed above are intertwined, and some may potentially offset the others. Without empirical evidence, it is challenging to determine the overall impact and relative significance of each of these factors. The objective of this paper is to measure such impacts to gain insight into the relationship between graph analytics and GPU architectures by conducting extensive experiments.

3 GRAPH PROCESSING LIBRARY ON GPUS

3.1 High-performance graph analytics

Researchers have been developing specialized libraries for GPUs to speed up large-scale graph analytics [25]. These libraries can be broadly categorized into two categories based on the abstractions that they adopted: linear algebra-based and graph abstraction-based.

The linear algebra-based libraries abstract a graph as a sparse matrix and transform graph algorithms into matrix operations. When combined with algebraic semirings, a small set of primitives can form a wide range of graph algorithms. There exist various implementations of linear algebra-based graph analytics models, such as the GBTL [26], nvGRAPH [27], and GraphBLAST [28].

Most graph abstraction-based analytics libraries adopt the vertex-centric or edge-centric models to optimize processing. The optimizations mainly focus on workload mapping and memory access. The vertex-centric model emphasizes parallelization over vertices, while the edge-centric model parallelizes over edges. Examples under this category comprise CuSha [29], MapGraph [30], and Gunrock [10]. Meanwhile, there are some other frameworks that employ the path-centric [31] or subgraph-centric [32] models, which can be regarded as variants of vertex-centric and edge-centric models.

3.2 nvGRAPH and adGRAPH

3.2.1 nvGRAPH for NVIDIA. For various reasons, we chose nvGRAPH as the graph analytics library for the experiments in this paper. Firstly, nvGRAPH [27], as a mature graph analytics library developed by NVIDIA, is included in CUDA ToolKit, which is widely used in practice. Secondly, although nvGRAPH is not currently SOTA, the implementation of graph algorithms in nvGRAPH has been widely adopted in more recent graph analysis libraries like cuGRAPH [33] and GraphBLAST [28], and serves as the cornerstone of optimization strategies in these libraries (more details will be explained along with the specific benchmark algorithms in Section 4). This means the insights obtained from the nvGRAPH experiment are more likely to be extended to other scenarios.

The following are the principal characteristics of the nvGRAPH library:

- nvGRAPH represents many graph analytics problems through linear algebra and matrix computations, introducing semiring Sparse Matrix-Vector Product (SPMV) for executing some graph operations.
- nvGRAPH incorporates a range of performance optimization techniques in its underlying implementation, such as triangle counting operations that rely on set intersection and direction-optimizing BFS.

3.2.2 adGRAPH for AMD-like GPUs. Due to NVIDIA's first-mover advantage in GPU hardware and software, almost all GPU graph analytics software only supports NVIDIA GPUs. This limitation restricts their use on different hardware. Due to the general lack of graph analytics software on non-NVIDIA GPUs, we have to implement one on AMD-like GPUs in order to conduct the study of this paper.

However, developing a software library[3] on an AMD-like GPU from scratch poses significant engineering challenges. Developing such libraries involves time-consuming and labor-intensive work and requires strong development capabilities and a thorough understanding of the relevant software stacks, such as NVIDIA's CUDA or AMD's ROCm. Therefore, many developers now opt to port CUDA software to non-Nvidia devices. The heterogeneous programming model HIP developed by AMD makes such porting feasible.

In addition, porting can better help us draw correct conclusions in comparative experiments. As our objective is to study the impact of GPU architectures, we aim to minimize the differences in the software running on the GPUs so that the observed impacts are more likely caused by the GPU hardware. Compared to redevelopment from scratch, porting can better ensure code consistency across different hardware to the greatest extent possible, allowing us to focus on observing the impacts brought by GPU differences without the need to consider the architecture and characteristics of the different graph analytics libraries. So, we opt for porting over redevelopment. Our ported version of nvGRAPH for AMD-like GPUs is called adGRAPH.

3.3 Implementation of adGRAPH

To implement adGRAPH, we designed and implemented a porting process as shown in Fig 3. The key steps in the entire process are as follows. First, we utilized automated tools to substitute most CUDA interfaces with HIP's counterparts. Then, we conducted manual adjustments or modifications to replace all the CUDA's variables with HIP's and resolve system heterogeneity issues.

Nonetheless, porting large-scale software, such as nvGRAPH, is non-trivial, which is also concurred by industrial feedback [34] [35]. First, automated tools [36] are unable to recognize complex code constructs like templates, structs, encapsulation classes, and other user-defined data types. This defect can lead to automation tools easily missing CUDA elements in complex code structures when executing the porting. Second, there is a high level of heterogeneity in the interface between HIP and CUDA, especially in some interface parameters. For example, compared to the cusparseXcsriluo interface of CUDA, the hipsparseXcsriluo interface of HIP requires a larger number of parameters. Such mismatchings

[3] As executing CUDA software through a translation layer like ZLUDA on non-NVIDIA GPUs is banned by NVIDIA, we do not consider this option.

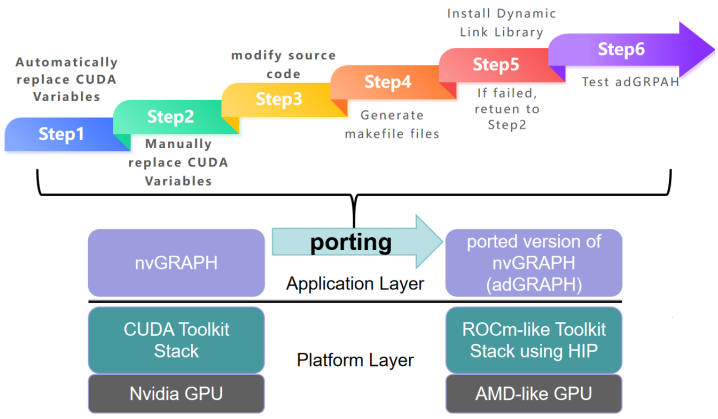

Figure 3: Porting Process of nvGRAPH

occurred very frequently during porting. We overcome these challenges through manual intervention, which, unfortunately, involves extraordinarily time-consuming and laborious tasks.

Due to the porting method, the main differences between nvGRAPH and adGRAPH lie in the interfaces and variables. AdGRAPH replaces the CUDA variables or interfaces in nvGRAPH with their HIP counterparts (e.g., replacing cudaStream_t with hipStream_t, and cudaMemcpy with hipMemcpy, and so on). Moreover, the HIP elements are functionally identical to their counterparts of CUDA. At the platform layer shown in Fig 3, the ROCm-like Toolkit corresponding to AMD-like GPUs is designed and developed to mimic and be compatible with the CUDA Toolkit and is relatively close in terms of software architecture and application ecology. So, for the convenience of conducting the experiment, we assume that the difference between the ROCm-like Toolkit and the CUDA Toolkit will not have a significant impact on the experimental phenomena. We did not make special optimization modifications to adGRAPH, thereby preserving the consistency of the graph algorithms in the two libraries. Therefore, even though we cannot ensure the two software stacks are identical and do not have any impact on the observed performance differences, it is the best that one can do, with a reasonable amount of resources, to minimize the interference from the software stacks.

4 EVALUATION

In this section, we conduct the experimental of this paper.

4.1 Experimental Metrics

We evaluate the nvGRAPH or adGRAPH on different GPUs through two main performance metrics: runtime and traversed edges per second (TEPS). We report runtimes in milliseconds and TEPS as millions of traversals per second [MTEPS].

To explore nvGRAPH or adGRAPH performance on different hardware platforms, we utilized two profiling tools: Nsight Compute for CUDA (ncu) and its ROCm-like equivalent (hiprof) to measure various profiling metrics of GPUs. We select 8 sets of micro-profiling metrics commonly used to evaluate GPU components and categorize them into two categories: fine-grained metrics and coarse-grained metrics.

- For **fine-grained level metrics**, we record the number of instructions issued by different components (including SIMD/Core, shared memory, and global memory) of the GPU, as well as the four similar metrics of adGRAPH. The result of these metrics combined with operation runtime expresses the speed of the above components in GPUs during operation. Table 1 provides a detailed explanation of these metrics.
- For the **coarse-grained metrics**, we record the utilization efficiency of four components(including L2 cache, SIMD/Core, shared memory and global memory) of GPUs. These metrics reveal the efficiency of utilizing the potential of above components on the GPU during operation. Table 2 lists detailed explanations.

Due to the fact that fine-grained metrics are related to the runtime of the program, in subsequent profiling, we will divide the metric values by the run time of each program to obtain the speed of instruction issuance for the different components and use such speed values to compare the efficiency of different components. Due to the differences in the statistical methods adopted by the different types of GPUs, we are unable to compare the coarse-grained profiling metrics across them directly. Therefore, we only separately report them to observe the operational characteristics of the graph framework on the two types of GPUs. All metrics, except runtime, are better with higher values.

Table 1: Specification of Fine-Grained Metrics

CUDA metrics and its implication	ROCm metrics and its implication
inst_issued(number of instructions issued)	SQ_INSTS_VALU(Number of VALU instructions issued)
inst_executed_shared_stores(Warp level instructions for shared stores)	SQ_INSTS_LDS(Number of LDS instructions issued)
inst_executed_global_loads(Warp level instructions for global loads)	SQ_INSTS_VMEM_RD(Number of VMEM read instructions issued)
inst_executed_global_stores(Warp level instructions for global stores)	SQ_INSTS_VMEM_WR(Number of VMEM write instructions issued)

Table 2: Specification of Coarse-Grained Metrics

CUDA metrics and its implication	ROCm metrics and its implication
achieved_occupancy(Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor)	VALUBusy(Percentage of GPUTime vector ALU instructions are processed.)
shared_efficiency(Ratio of requested shared memory throughput to required shared memory throughput expressed as percentage)	1-ALUStalledByLDS(Percentage of GPUTime ALU units are stalled by the LDS input queue being full or the output queue being not ready.)
l2_tex_hit_rate(Hit rate at L2 cache for all requests from texture cache)	L2CacheHit(Percentage of fetch, write, atomic, and other instructions that hit data in L2 cache.)
gld_efficiency(Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage.)	MemUnitBusy(Percentage of GPUTime the memory unit is active. The result includes the stall time.)

4.2 System Configuration

In this experimental study, we use NVIDIA's A100 [37] and V100 and the AMD-like Z100 and Z100L, representing NVIDIA GPU architecture and AMD-like GPU architecture, respectively. A100 and Z100L are the latest products we can obtain based on our resources, while V100 and Z100 are their prior-generation products. Table 3 outlines the hardware specifications of the four GPUs. As illustrated in Table 3, despite the gap in RAM specifications,

Z100L and A100 are nearly identical in key metrics such as FP64 computing power. This enables us to assess the actual impact of different GPU architectures on the performance of graph algorithms, given their similar hardware specifications. Similarly, although Z100 is slightly inferior to V100 in some specifications, it is comparable to V100 in terms of computing power and most specifications as shown in Table 3.

Table 3: Specification of GPUs

Features	Z100	V100	Z100L	A100
FP64	5.9TFLOPS	7.0TFLPS	10.1TFLOPS	9.7TFLOPS
FP32	11.8TFLOPS	14.0TFLOPS	12.2TFLOPS	19.5TFLOPS
RAM Volume	16GB	32GB	32GB	80GB
RAM Bandwidth	800GB/s	900GB/s	1024GB/s	1935GB/s
RAM Bitwidth	4096bit	4096bit	4096bit	5120bit
RAM Type	HBM2	HBM2	HBM2	HBM2e
SM/CU	64	80	64	108
Cores/SP	4096	5120	4096	6912

All experiments in this article were conducted on workstations running CentOS Linux 7.6.1810, whose kernel version is 3.10.0 − 957.$el7.x86\_64$. One workstation is equipped with an NVIDIA GPU, while the host is equipped with a memory size of 192GB, and Toolkit using CUDA 10.2; the other is equipped with an AMD-like GPU, while the host is equipped with 128GB memory and Toolkit is developed by manufacturers of Z100 and Z100L based on open source ROCm 5.0.

In this experiment, all graph data was presented in double-precision floating-point format (represented in 64-bit binary) on the computer, so the GPU's computing power benchmark is FP64 instead of FP32.

4.3 DataSet

As the major challenges of graph analytics using GPUs are the irregular access and workload imbalance brought by power law distribution often exhibited in large real-world graphs, the design of most graph libraries for GPUs like nvGRAPH mainly focused on optimizations to overcome such challenges. Therefore, we selected seven real-world graph datasets from the SNAP (Stanford Network Analysis Project) [38] and Network Repository websites [39], taking into account the characteristics of power law and large diameter in real-world graphs. More detailed information about these graphs is provided in Table 4.

Table 4: Specifications of DataSet

DataSet	Number of vertex	Number of edge	maxDegree
web-Stanford	281,903	2,312,497	38,626
web-Google	916,428	5,105,039	6,353
cit-Patents	6,009,554	16,518,948	739
soc-liveJournal1	4,847,571	68,475,391	22,887
soc-sinaweibo	58,655,849	261,321,071	278,489
web-uk-2002-all	18,520,486	298,113,762	194,955
twitter-mpi	52,579,682	1,963,263,821	3,691,240

These selected datasets encompass various real-world disciplines, such as social networks, citation links, and web data, covering many classical application scenarios for graph analysis. More specifically, Web-Stanford, web-Google, and web-uk-2002-all are examples of web graphs, while soc-liveJournal1, soc-sinaweibo, and twitter-mpi are social networks. Additionally, cit-patents represents a citation network, and the three largest graphs, soc-sinaweibo, web-uk-2002-all, and twitter-mpi, contain vertices with very high degrees, which could potentially lead to communication or computing bottlenecks due to workload imbalance.

4.4 Benchmark Algorithm

For the evaluation of the graph analytics library's performance on A100 and Z100L, we use three algorithms that maintain consistency in the implementation of nvGRAPH and adGRAPH: Breadth First Search (BFS), Triangle Count (TC), and Subgraph Extraction by Vertex (ESBV). These algorithms cover two major categories of graph analytics: traversal and mining. BFS represents a low-complexity algorithm, TC represents a moderate-complexity algorithm, and ESBV represents a high-complexity algorithm. Collectively, they provide a representative sample of graph analytics workloads.

BFS. BFS is the simplest of the three algorithms. In both adGRAPH and nvGRAPH, it is executed using a conventional bottom-up approach, focusing on expanding the frontier layer by layer as the main operation [40]. Subsequent research results often adopt further optimization techniques on the basis of this method, such as developing a special programming model [41] that combines custom hardware microarchitecture to address intra-warp load imbalance issues. However, it is unknown whether such special optimizations can be applied to AMD-like GPUs.

TC. In nvGRAPH and adGRAPH, the counting of triangles is done by summarizing the intersection elements in the adjacent node sets of adjacent nodes [42], which relies on bitmaps and atomic operations. This process requires more conditional judgments and branching statements than BFS, depending on the topology of the graph. This approach has pioneered one of the two mainstream paradigms for solving the TC problem, the other being the binary search-based paradigm. Specific optimizations for each of the two solutions have been proposed. E.g., a recent work [43] followed the latter paradigm and proposed a hash-based trie data layout to improve efficiency.

ESBV. Subgraph extraction involves not only traversing and retrieving subgraphs but also constructing them into new graph objects. So, ESBV is the most complex among the three, as it requires more frequent conditional checks and branching statements. In nvGRAPH and adRGAPH, the solution follows canonical computing of partial clustering [44]. But in recent years, new research has introduced methods such as Signed Graph Representation Learning [45] and other machine learning approaches [46] based on partial clustering to improve efficiency, but these implementations go beyond the scope of the graph analysis algorithms targetted by this study.

4.5 Performance Result

We divide the four GPUs used in the experiment study into two groups. The first consists of NVIDIA's V100 and AMD-like Z100, while the second consists of NVIDIA's A100 and AMD-like Z100L. We run adGRAPH on AMD-like GPUs and nvGRAPH on NVIDIA GPUs. Through experiments, we observe and compare the performance differences between NVIDIA and AMD-like GPUs within each group, respectively. Table 5 shows the performance results of the experiment.

Table 5: Performance Result of nvGRAPH and adGRAPH

Task	Workload	Group 1				Group 2			
		Runtime(ms)		Edge Throughout (Million/s)		Runtime(ms)		Edge Throughout (Million/s)	
		Z100	V100	Z100	V100	Z100L	A100	Z100L	A100
BFS	web-Stanford	15.68	**10.04**	147.48	**230.33**	**9.28**	9.6	**249.19**	240.89
	web-Google	**0.91**	4.11	**5609.99**	1242.12	**0.58**	2.93	**8801.88**	1742.35
	cit-Patents	**3.13**	3.61	**5277.62**	4575.88	**1.91**	2.13	**8648.66**	7755.37
	soc-liveJournal1	**7.27**	8.29	**9418.90**	8260.00	**4.6**	5.04	**14885.95**	13586.29
	soc-sinaweibio	**4.25**	9.08	**61487.31**	28779.85	**2.55**	5.22	**102478.85**	50061.51
	web-uk-2002-all	**24.86**	35.73	**11991.70**	8343.51	**13.09**	13.87	**22774.16**	21493.36
	twitter-mpi	95.85	**89.36**	20482.67	**21970.28**	65.21	57.59	30106.79	**34090.42**
TC	web-Stanford	37.15	**26.52**	62.25	**87.20**	21.98	**18.18**	105.21	**127.20**
	web-Google	**40.72**	43.44	**125.37**	117.52	**34.64**	64.05	**147.38**	79.70
	cit-Patents	**60.31**	60.49	**273.90**	273.09	**35.62**	57.29	**463.75**	288.34
	soc-liveJournal1	**212.95**	398.43	**321.56**	171.86	**137.37**	234.58	**498.47**	291.91
	soc-sinaweibio	4953.03	**1064.35**	52.76	**245.52**	2436.70	**388.43**	107.24	**672.76**
	web-uk-2002-all	628.57	**189.46**	474.27	**1573.49**	397.76	**163.86**	749.48	**1819.32**
	twitter-mpi	146945	**107912**	13.36	**18.19**	103518	**55642.12**	18.97	**35.28**
ESBV	web-Stanford	**3**	3.42	**770.83**	676.17	**1.78**	1.81	**1299.16**	1277.62
	web-Google	3.24	**2.6**	1575.65	**1963.50**	2.11	**1.41**	2419.48	**3620.63**
	cit-Patents	4.17	**4.13**	3961.38	**3999.74**	2.44	**1.25**	6770.06	**13215.15**
	soc-liveJournal1	9.39	**8.39**	7292.37	**8161.55**	5.65	**2.77**	12119.54	**24720.36**
	soc-sinaweibio	30.41	**21.01**	8593.26	**12437.94**	18.43	**11.36**	14179.11	**23003.62**
	web-uk-2002-all	39.42	**38.74**	7562.50	**7695.24**	17.87	**14.17**	16682.36	**21038.37**
	twitter-mpi	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM

It is interesting to note that neither nvGRAPH nor adGRAPH can form a crushing advantage over the other in all algorithms. For simple BFS, the performance of adGRAPH on AMD-like accelerators (Z100L or Z100) is much better than that of nvGRAPH on the NVIDIA counterparts (A100 or V100). On the other hand, for the more complex ESBV, the performance of nvGRAPH is far superior to that of adGRAPH. Regarding TC, which is of moderate complexity, the performance of nvGRAPH and adGRAPH varies depending on different datasets.

BFS. The throughput of adGRAPH is higher than that of nvGRAPH when executing BFS. In group 1, the acceleration ratio of adGRAPH on Z100 compared to nvGRAPH on V100 ranges from 0.64x to 4.52x, with an average speedup ratio of 1.69x, as shown in Figure 4. In group 2, the acceleration ratio of adGRAPH on Z100L compared to nvGRAPH on A100 ranges from 0.88x to 5.05x, with an average speedup ratio of 1.76x, as shown in Figure 5. Among the seven graph datasets, adGRAPH has a lower runtime than nvGRAPH on almost all graphs except for twitter-mpi in both groups, which has an average degree of 74.68, at least twice that of the other datasets.

TC. The overall difference in throughput between adGRAPH and nvGRAPH for triangle counting is not significant. In group 1, the acceleration ratio of adGRAPH on Z100 compared to nvGRAPH on V100 ranges from 0.21 to 1.87x, with an average speedup ratio of 0.84x, as shown in Figure 4. In group 2, the acceleration ratio of adGRAPH on Z100L compared to nvGRAPH on A100 ranges from 0.15 to 1.85x, with an average speedup ratio of 1.01x, as shown in Figure 5. Among experiments on the seven datasets, the throughput of adGRAPH was greater than that of nvGRAPH

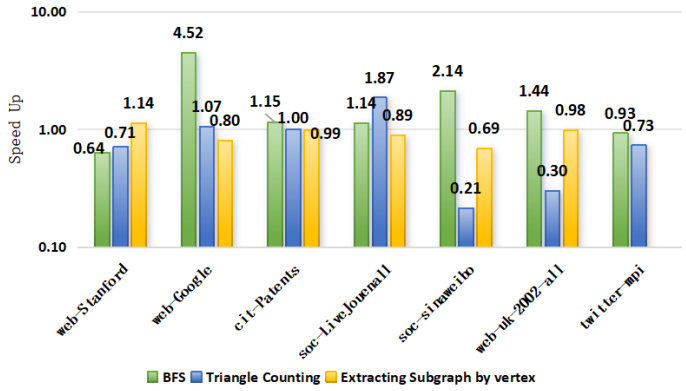

Figure 4: Speed Up of nvGRAPH on Z100 relative to V100

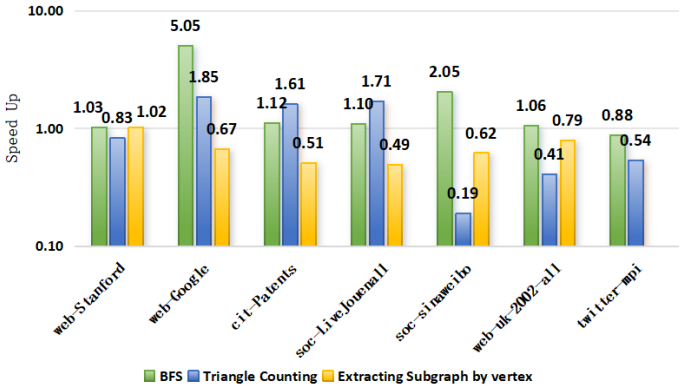

Figure 5: Speed Up of nvGRAPH on Z100L relative to A100

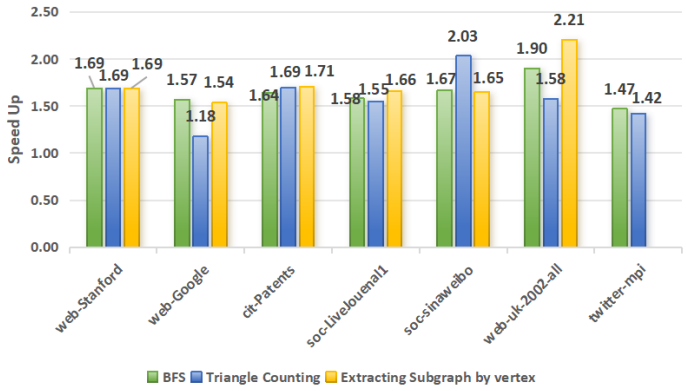

Figure 6: Speed Up of nvGRAPH on Z100L relative to Z100

in cit-Patents, soc-liveJournal1, and web-Google, accounting for 3 out of 7 cases.

ESBV. Due to the requirement of edge weight data and the substantial storage space occupied by edge weights, the workstations used in the experiments are unable to support executing the ESBV algorithm of nvGRAPH or adGRAPH on the twitter-mpi dataset. Among the remaining six graph datasets, the throughput of adGRAPH was consistently lower than the throughput of nvGRAPH across all datasets except for web-Standford. In group 1, the acceleration ratio of adGRAPH on Z100 relative to nvGRAPH on V100 ranges between 0.69x and 1.14x, with an average acceleration ratio of 0.92x, as shown in Figure 4. In group 2, the acceleration ratio of adGRAPH on Z100L relative to nvGRAPH on A100 ranges between 0.49x and 1.02x, with an average acceleration ratio of 0.68x, as shown in Figure 5.

Verification of adGRAPH. In both Group 1 and Group 2, even though the hardware parameters of the AMD-like GPUs are slightly inferior to the NVIDIA ones, nvGRAPH cannot fully outperform adGRAPH and is often even overtaken by adGRAPH. Due to the high similarity in the code between nvGRAPH and adGRAPH and the high throughput demonstrated on their respective GPUs, it is reasonable to confirm that adGRAPH is on par with the original nvGRAPH in terms of exploiting the potential of GPUs' acceleration capability. Besides, while the FP64 computing power of Z100L is nearly 1.71 times that of Z100, the average acceleration ratio of adGRAPH on Z100L compared to Z100 can reach 1.65, with an average acceleration ratio of 1.64 on BFS tasks and 1.59 on TC tasks and 1.74 on ESBV tasks respectively (shown in Fig 6). This indicates that the parallel efficiency of adGRAPH is high. In summary, the evaluation results demonstrate that adGRAPH is effective and sufficient to serve as a benchmark on AMD-like GPUs.

Sensitivity to Graph Properties. It is not difficult to see from the overall trends that adGRAPH has an advantage on relatively smaller graphs, while nvGRAPH has an advantage on relatively larger graphs, especially when the number of vertices is more than 100 million. AdGRAPH has an advantage on BFS, but lags behind nvGRAPH on the largest graph twitter-mpi, and its advantage gradually degrades as the graph size and maximum degree increase. The overall trend on TC is more obvious: adGRAPH dominates on most small graphs, while nvGRAPH dominates on large graphs. Although most datasets have similar topological structures except for cit-patents, the two libraries may have different inclinations in different scenarios. AdGRAPH is often more advantageous when the maximum degree is much relatively smaller than total number of edges, which helps to enhance its advantages on small graphs such as running BFS on web-Google, or partially offset its disadvantages on large graphs such as running TC tasks on twitter-mpi.

4.6 Profiling Result

Table 6 presents the results of profiling analysis at the fine-grained level for the two GPUs. It is worth noting that the data presented in Table 6 is the speed at which different components issue instructions, obtained by dividing the number of instructions by the program runtime. We compare the efficiency of two GPU architectures by comparing speed rather than instruction number. Since Z100L and A100 are the more advanced and widely used varieties in AMD-like and NVIDIA architectures, it is more interesting to explore the process of running adGRAPH or nvGRAPH on these two GPUs. Therefore, our profiling will only be conducted on these two GPUs. We identify the following:

- Z100L outperforms A100 on inst_issued, which means that Z100L can issue more instructions, especially those related to warp or wavefront when executed BFS and TC algorithm.
- A100 performs significantly better than Z100L when executed ESBV, while performing equally to Z100L in other algorithms when it comes to inst_executed_global_loads. This indicates that A100 is slightly more efficient than Z100L in global memory read operations.
- Z100L lags far behind A100 in global memory write operations, as indicated by its significantly inferior performance in inst_executed_global_stores.

Table 6: Fine-grained Profiling Results of 2 GPUs running nvGRAPH or adGRAPH

Metrics Type	Workload	BFS		ESBV		TC	
		A100	Z100L	A100	Z100L	A100	Z100L
Type 1*	web-Stanford	**5.18K/ms**	3.91K/ms	**2.39K/ms**	2.22K/ms	18.57M/ms	**19.30M/ms**
	web-Google	147.48K/ms	**994.35K/ms**	**2.67K/ms**	1.67K/ms	18.86M/ms	**21.75M/ms**
	cit-Patents	129.73K/ms	**191.68K/ms**	**4.80K/ms**	2.61K/ms	23.93M/ms	**28.21M/ms**
	soc-liveJournal1	1.74M/ms	**6.04M/ms**	**4.44K/ms**	1.86K/ms	22.43M/ms	**59.48M/ms**
	soc-sinaweibio	3.95M/ms	**5.16M/ms**	**384.75/ms**	215.55/ms	48.75M/ms	**86.65M/ms**
	web-uk-2002-all	**1.07M/ms**	650.17K/ms	**361.25/ms**	262.15/ms	**81.94M/ms**	70.26M/ms
Type 2**	web-Stanford	773.22/ms	**846.23/ms**	90.73/ms	**200.80/ms**	391.22K/ms	**437.36K/ms**
	web-Google	24.44K/ms	**165.17K/ms**	86.05/ms	**123.29/ms**	195.46K/ms	**345.36K/ms**
	cit-Patents	16.68K/ms	**25.98K/ms**	42.27/ms	**257.12/ms**	**502.03K/ms**	406.06K/ms
	soc-liveJournal1	1.47M/ms	**4.27M/ms**	**320.62/ms**	129.52/ms	2.66M/ms	**7.70M/ms**
	soc-sinaweibio	438.95K/ms	**693.49K/ms**	14.46/ms	**15.72/ms**	6.00M/ms	**6.25M/ms**
	web-uk-2002-all	**166.05K/ms**	126.83K/ms	15.91/ms	**19.07/ms**	11.25M/ms	**11.99M/ms**
Type 3***	web-Stanford	**501.91/ms**	72.69/ms	**133.10/ms**	116.05/ms	245.20K/ms	**531.55K/ms**
	web-Google	6.11K/ms	**36.31K/ms**	**138.32/ms**	83.43/ms	135.47K/ms	**572.70K/ms**
	cit-Patents	**3.20K/ms**	2.02K/ms	**209.66/ms**	119.86/ms	383.43K/ms	**920.56K/ms**
	soc-liveJournal1	76.36K/ms	**103.74K/ms**	**319.48/ms**	128.70/ms	1.57M/ms	**2.67M/ms**
	soc-sinaweibio	**146.32K/ms**	119.97K/ms	**21.68/ms**	11.26/ms	**4.25M/ms**	3.68M/ms
	web-uk-2002-all	**44.85K/ms**	16.42K/ms	**21.87/ms**	14.34/ms	**6.03M/ms**	2.26M/ms
Type 4****	web-Stanford	**42.36/ms**	16.27/ms	**112.43/ms**	81.98/ms	**1.07M/ms**	617.12K/ms
	web-Google	5.50K/ms	**38.33K/ms**	**129.59/ms**	80.74/ms	**1.20M/ms**	613.29K/ms
	cit-Patents	**4.28K/ms**	3.47K/ms	**251.18/ms**	156.64/ms	**1.33M/ms**	584.33K/ms
	soc-liveJournal1	16.41K/ms	**115.54K/ms**	**189.82/ms**	82.85/ms	**317.46K/ms**	150.94K/ms
	soc-sinaweibio	**158.02K/ms**	21.51K/ms	**18.11/ms**	10.44/ms	**166.74K/ms**	8.20K/ms
	web-uk-2002-all	**9.35K/ms**	4.44K/ms	**16.99/ms**	12.75/ms	**391.58K/ms**	45.31K/ms

* inst_issued for nvGRAPH or SQ_INSTS_VALU for adGRAPH;
** inst_executed_shared_stores for nvGRAPH or SQ_INSTS_LDS for adGRAPH;
*** inst_executed_global_loads for nvGRAPH or SQ_INSTS_VMEM_RD for adGRAPH;
**** inst_executed_global_stores for nvGRAPH or SQ_INSTS_VMEM_WR for adGRAPH;

- As shown by the metric inst_executed_shared_stores, A100 is far inferior to Z100L in almost all cases, which implies that the performance of Z100L in memory access operations on shared memory is much better than that of A100.

From the fine-grained profiling data mentioned above, we can make two observations. Firstly, the AMD-like architecture represented by Z100L has better optimization in warp and shared memory than the NVIDIA architecture represented by A100. Secondly, in addition to shared memory, the performance of nvGRAPH and adGRAPH also depends on the complexity of the algorithm being executed. Generally, nvGRAPH performs better in complex algorithms like ESBV, while adGRAPH is more competitive in simpler algorithms like BFS.

Figure 7 and 8 displays coarse-grained profiling results for the two GPUs. We can draw the following conclusions from Figure 7 and 8:

- The utilization efficiency of warp computing resources on both A100 and Z100L is low. This indicates that although nvGRAPH and adGRAPH have undergone extensive optimization, the weak locality of the graph algorithms has not been fully eliminated.

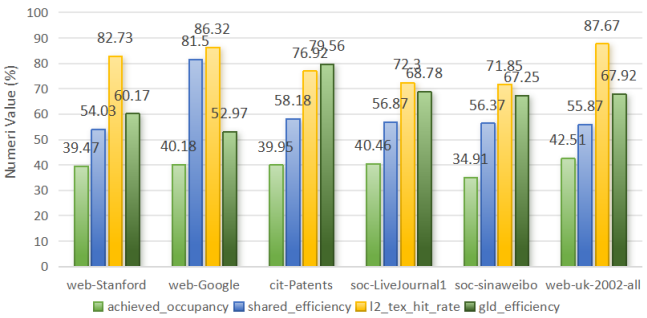

Figure 7: Coarse-grained Profiling Results of nvGRAPH on A100

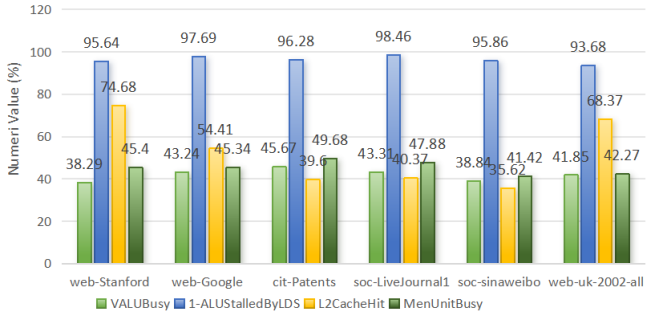

Figure 8: Coarse-grained Profiling Results of adGRAPH on Z100L

- Regarding the efficiency of shared memory utilization, Z100L performs better than A100. This also indirectly supports the assertion made in the fine-grained profiling that Z100L has shared memory features that A100 cannot match.
- When comparing the efficiency of L2 cache hit and global memory utilization, it is observed that Z100L does not perform as well as A100. This difference can be attributed to the use of advanced storage technologies such as the HBM2e by A100(Z100L use inferior HBM2 and its volume is smaller than A100). Due to this, Z100L lags behind A100 in terms of L2 cache hit rate and global memory efficiency.

5 DISCUSSION

In this section, we discuss three questions related to the results in Section 4. (1) Why does an AMD-like GPU outperform a NVIDIA GPU on certain specific graph algorithms? (2) Why do some algorithms exhibit higher execution efficiency on AMD-like GPUs than on NVIDIA GPUs, while some others demonstrate the opposite? (3) Why does an AMD-like GPU perform better in shared memory than an NVIDIA GPU for all three benchmark algorithms?

As discussed in Section 2, the architecture design of a GPU is expected to have various impacts on the graph analytics performance, which can be summarized in the following hypothesis. **Hypothesis 1**: A larger wavefront size of AMD-like GPUs helps them outperform NVIDIA GPUs due to their theoretically higher degree of parallelism. **Hypothesis 2**: Independent LDS (shared memory) can help AMD-like GPUs perform better than NVIDIA in terms of shared memory efficiency. **Hypothesis 3**: NVIDIA GPUs using SIMT are more suitable for handling parallel tasks with complex program branches than AMD-like counterparts.

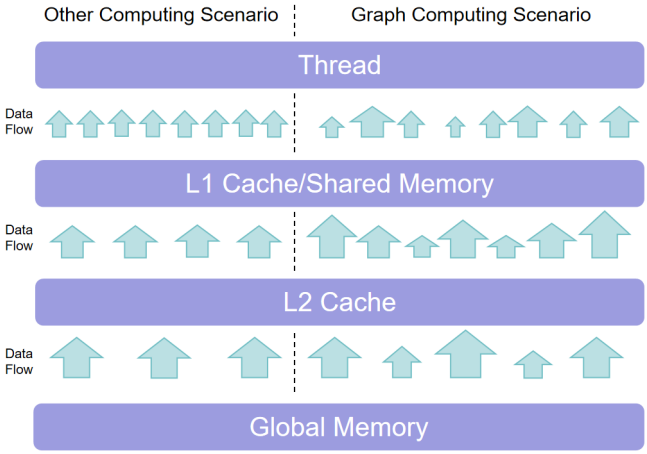

Figure 9: Locality of GPU architecture under graph computing and other scenarios.[4]

Hypothesis 4: The non-independent shared memory in NVIDIA GPUs reduces the latency of transmission between the L1 cache and shared memory.

Besides the architecture, the other hardware parameters of GPUs also have an impact on the performance of graph analytics. Although the Z100L and A100 have almost the same computing power, with Z100L having a slight advantage, A100 adopts a more advanced RAM technology and has a significantly higher RAM capacity and bandwidth than Z100L. In the comparative experiment between V100 and Z100, the hardware parameters of the two GPUs also have a similar impact on the experiment. Therefore, we have **Hypothesis 5**: based solely on the hardware parameters, the overall expected performance of nvGRAPH on A100 or V100 should be higher than adGRAPH on Z100L or Z100.

However, according to the experimental results, we found that the effect we observed in Section 4 was not fully consistent with our expectations mentioned above. In the rest of this section, we analyze the overall impact and relative significance of each of these factors based on the empirical evidence.

5.1 Analysis of experimental phenomena

To answer the three questions mentioned at the beginning of this section, we need to compare the expected impact with the observed impact found in Section 4 and analyze if they are consistent or not.

5.1.1 Independent Shared Memory. As discussed earlier, the independent shared memory in AMD-like GPUs is unaffected by the L1 cache but has longer memory access latency compared to Nvidia GPUs. However, the profiling results, where Z100 outperforms A100 in metrics such as inst_executed_stores and shared_efficiency, only highlight the positive effects of independent shared memory. They do not demonstrate the anticipated negative impact. This necessitates a more in-depth discussion.

We illustrate the differences between computation with good locality versus graph computation with poor locality in Fig. 9. The hierarchical storage architecture of GPUs is designed based on the assumption of memory locality. However, given the pronounced irregular memory access pattern of graph algorithms, the spatiotemporal locality of memory access is limited. This

[4]The size of the arrow represents the size of the transmitted data, and number of arrows represents the frequency of data transmission.

has led to high variability of memory access, such as frequent loading of large amounts of data from the on-device memory to the on-chip jjmemory because of the low cache hit rate [47]. Due to the inherent load imbalances and irregular memory access patterns typical in graph analytics, the L1 cache–often with a low hit rate–frequently needs to fetch data from the L2 cache or even global memory. Consequently, this exacerbates the issue of bandwidth contention between shared memory and L1 cache in Nvidia. But due to the independent data transfer paths discussed in Section 2, AMD-like LDS does not suffer from interference of L1 cache's data transmission. So, in principle, AMD-like LDS should have a higher data transfer performance.

The profiling results of the Type 2 metrics in Table 6 show that the AMD-like GPU outperforms NVIDIA GPUs in most cases regarding the metric of inst_executed_stores, which measures the performance of shared memory data transfer. This fully supports the above inference and also embodies the advantages of independent shared memory observed by the previous experimental study [24] as mentioned in Section 2. Additionally, the independent data path of LDS also enhances the operational efficiency of the AMD-like LDS due to its exemption from any bandwidth fluctuations caused by the L1 cache. This is reflected in the significantly lower shared_efficiency of the NVIDIA GPU shown in Figure 7, compared to the corresponding metric of the AMD-like GPU in Figure 8. This also supports our discussion above.

All in all, in graph computing, the positive effects of independent shared memory outweigh its negative effects, resulting in AMD-like GPUs outperforming NVIDIA GPUs in terms of shared memory efficiency, which supports Hypothesis 2 but not Hypothesis 4.

5.1.2 Warpsize. We have hypothesized that a larger wavefront size for AMD-like GPUs would result in higher parallelism than NVIDIA GPUs. This impact was manifested by the fine-grained profiling results of inst_issued where Z100L performs better in BFS and TC tasks, confirming Hypothesis 1. However, the impact of warp size can be hedged by other factors, which can be observed through Z100L performing worse than A100 in inst_issued when executing ESBV, which has a higher algorithm complexity. We will discuss the impact of Algorithm Complexity and SIMT in the next subsubsection.

5.1.3 Algorithm Complexity and SIMT. Based on our findings in Section 4, we observed that in terms of runtime and throughput during evaluation, as well as various metrics in fine-grained profiling, simpler algorithms tend to perform better on AMD-like GPUs, whereas more complex algorithms perform better on NVIDIA GPUs, with the exception of shared memory. We believe that this is closely related to NVIDIA GPUs adopting the SIMT paradigm and AMD-like GPUs adopting the SIMD paradigm. As we discussed the expected impact of the SIMT paradigm on NVIDIA GPUs and the SIMD paradigm on AMD-like GPUs, We found that NVIDIA GPUs are more flexible and better suited for handling complex parallel tasks. However, in the evaluation of Section 4, we observed that the impact of SIMT varies depending on the benchmark algorithm. For algorithms like BFS with relatively few branches, the optimization effect of SIMT appears redundant. For TC, whose complexity and branch count are moderate, the role of NVIDIA SIMT and the optimization points of AMD-like GPU interact, resulting in NVIDIA slightly outperforming AMD-like GPUs. For algorithms like ESBV, which have more complex branches, the optimization effect of SIMT is even more prominent than the other advantages of AMD-like GPU architecture. This evidence supports Hypothesis 3.

5.1.4 Hardware parameters. From Section 4, it can be seen that in both two groups, NVIDIA devices are superior to AMD-like ones in terms of RAM technology, bandwidth, and capacity. The superior performance of A100 regarding inst_executed_global _stores and gld_efficiency in Table 6 and Fig 7 confirm the advantage of A100 in on-device memory access speed during executing graph algorithms. It is important to note that as the scale of the graph increases, the I/O transmission of on-device memory becomes a significant bottleneck in performance. In this context, Nvidia GPUs have a substantial advantage over AMD-like GPUs, which helps them perform better with very large graphs. This observation aligns with the performance trends we discussed in Section 4 regarding the sensitivity to the scale of the graphs. But from Table 5, we can see that the advantage of A100 (or V100) in RAM technology did not make them outperform Z100L (or Z100) in all situations as expected. It is because some of the factors discussed earlier offset this one. Therefore, we do not have the evidence to support Hypothesis 5. Nonetheless, NVIDIA's advanced RAM technology contributes to the cases when nvGRAPH on A100 or V100 surpasses adGRAPH on Z100L or Z100.

5.2 Analysis Summary

Based on the characteristics of the benchmark algorithms and the two GPU architectures, our analysis can be summarized as follows. Graph algorithms have varying levels of complexity, which affects how much they benefit from SIMT. For example, algorithms with low branching complexity, like BFS, may not see a significant improvement in execution efficiency when using NVIDIA GPUs with SIMT. This is where AMD-like GPUs with larger warp sizes and independent shared memory can have an advantage. In such cases, the benefits of using NVIDIA's SIMT can be negated, making AMD-like GPUs a better choice. However, this advantage is not seen in the other algorithms, where out of the three variables – SIMT paradigm, independent shared memory, and warp size – SIMT paradigm has the most significant impact on performance.

5.3 Threats to Validity

Our experimental study of the two GPU architectures is subject to the following threats to validity:

1. The influence of computing platforms. NVIDIA and AMD-like GPUs use two different computing platforms, namely CUDA and ROCm, shown in Fig 3. nvGRAPH and adGRAPH rely on these two underlying platforms, which can impact their performance when executing specific graph algorithms. For the sake of this study, we assume in Section 3 that any differences between the platforms will not significantly affect the experimental outcomes. However, because CUDA is not open source, we cannot guarantee consistency between the two computing platforms by conducting code analysis. Therefore, the effect demonstrated in the results may not be completely attributed to the hardware architectures. One approach for accurately evaluating the impacts from different computing platforms is to conduct comparative experiments using the same hardware for both platforms. This can be achieved by porting one platform to the other's hardware. However, current porting tools demand an excessive amount of effort to do so. In order to make this approach more feasible, we need significant advancements in more efficient porting solutions in the future.

2. The difference in the calculation method of the metrics provided by the profiling tools. The profiling tools of the two GPUs could calculate the metrics differently, which makes it difficult to maintain complete consistency in the metrics we use for the study. In our results, we have selected metrics with comparable meanings in ROCm and CUDA for profiling, but we cannot completely rule out errors caused by the different calculation methods of metrics.

3. Generalization beyond algorithms and libraries. We used the mature graph analytics library nvGRAPH (and its ported version), which primarily employs classical graph algorithms, to investigate the impact of GPU architectures. However, with more recently emerging algorithms that incorporate special optimization strategies such as kernel fusion or optimized data layout, the impact of GPU architecture requires further research. There is a risk that the conclusion drawn in this paper may not apply to these advanced algorithms. For example, while most existing graph analysis libraries store graph data in CSR or CSC formats to reduce memory consumption, being accompanied by frequent irregular memory access, a hierarchical indicator has been developed in RealGraphGPU [8] to facilitate graph data storage with a greatly reduced incidence of irregular memory access. This could potentially lessen the validity of Hypothesis 2, which assumes that frequent irregular memory access is present.

6 CONCLUSION

In this paper, we investigate the performance impact of design choices of two main-stream GPU architectures on high-performance graph analytics. To enable a fair and pragmatic comparison of GPUs with two computing platforms(ROCm-like and CUDA), we implemented adGRAPH by porting nvGRAPH to the ROCm-like platform. Our experimental results show that the larger warp size and independent shared memory of AMD-like GPUs can indeed enhance the parallelism, leading to higher performance for graph algorithms with lower complexity, such as BFS. On the other hand, the adoption of SIMT by NVIDIA's GPU achieves better performance for algorithms with high complexity, such as ESBV.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China under Grant 62372333, the Fundamental Research Funds for the Central Universities under Grant 2042023kf0135, the Key Research and Development Program of Hubei Province under Grant 2024BAB044 and 2023BAB078, the Project funded by China Postdoctoral Science Foundation under Grant 2022M722459, and the Knowledge Innovation Program of Wuhan - Basic Research under Grant 2023010201010063.

REFERENCES

[1] Elwood Spencer Buffa. 1977. Graph Theory with Applications. *Journal of the Operational Research Society* (1977).

[2] John Douglas Owens, David P. Luebke, Naga K. Govindaraju, Mark J. Harris, Jens H. Krüger, Aaron E. Lefohn, and Timothy J. Purcell. 2007. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum* 26 (2007).

[3] Jonas Dann, Daniel Ritter, and Holger Fröning. 2020. Exploring Memory Access Patterns for Graph Processing Accelerators. In *Datenbanksysteme für Business, Technologie und Web.*

[4] Hongru Gao, Xiaofei Liao, Zhiyuan Shao, Kexin Li, Jiajie Chen, and Hai Jin. 2023. A survey on dynamic graph processing on GPUs: concepts, terminologies and systems. *Frontiers of Computer Science* 18 (2023), 1–23.

[5] Yunming Zhang, Mengjiao Yang, Riyadh Baghdadi, Shoaib Kamil, Julian Shun, and Saman P. Amarasinghe. 2018. GraphIt: a high-performance graph DSL. *Proceedings of the ACM on Programming Languages* 2 (2018), 1 – 30. https://api.semanticscholar.org/CorpusID:53088127

[6] Hang Liu and Howie Huang. 2018. SIMD-X: Programming and Processing of Graph Algorithms on GPUs. In *USENIX Annual Technical Conference.*

[7] Santosh Pandey, Lingda Li, Adolfy Hoisie, Xiaoye Sherry Li, and Hang Liu. 2020. C-SAW: A Framework for Graph Sampling and Random Walk on GPUs. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), 1–15.

[8] Myung-Hwan Jang, Yunyong Ko, DongKyun Jeong, Jeong-Min Park, and Sang-Wook Kim. 2022. RealGraphGPU: A High-Performance GPU-Based Graph Engine toward Large-Scale Real-World Network Analysis. *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (2022).

[9] Xuanhua Shi, Xuan Luo, Junling Liang, Peng Zhao, Sheng Di, Bingsheng He, and Hai Jin. 2018. Frog: Asynchronous Graph Processing on GPU with Hybrid Coloring Model. *IEEE Transactions on Knowledge and Data Engineering* 30 (2018), 29–42.

[10] Yangzihao Wang, Andrew A. Davidson, Yuechao Pan, Yuduo Wu, Andy T. Riffel, and John Douglas Owens. 2015. Gunrock: a high-performance graph processing library on the GPU. *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2015).

[11] Seung Won Min, Vikram Sharma Mailthody, Zaid Qureshi, Jinjun Xiong, Eiman Ebrahimi, and Wen mei W. Hwu. 2020. EMOGI: Efficient Memory-access for Out-of-memory Graph-traversal In GPUs. *Proc. VLDB Endow.* 14 (2020), 114–127.

[12] Yu Zhang, Xiaofei Liao, Hai Jin, Bingsheng He, Haikun Liu, and Lin Gu. 2019. DiGraph: An Efficient Path-based Iterative Directed Graph Processing System on Multiple GPUs. *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019).

[13] Himchan Park and Min-Soo Kim. 2018. EvoGraph: An Effective and Efficient Graph Upscaling Method for Preserving Graph Properties. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018).

[14] Lin Jiang, Ru Feng, Junjie Wang, and Junyong Deng. 2022. DBR: A Depth-Branch-Resorting Algorithm for Locality Exploration in Graph Processing. *2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)* (2022), 178–184. https://api.semanticscholar.org/CorpusID:254930494

[15] Nathan Otterness and James H. Anderson. 2021. Exploring AMD GPU scheduling details by experimenting with "worst practices". *Real-Time Systems* 58 (2021), 105 – 133.

[16] Nathan Otterness and James H. Anderson. 2020. AMD GPUs as an Alternative to NVIDIA for Supporting Real-Time Workloads. In *Euromicro Conference on Real-Time Systems.*

[17] Indrani Paul, Wei Huang, Manish Arora, and Sudhakar Yalamanchili. 2015. Harmonia: Balancing compute and memory power in high-performance GPUs. *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)* (2015), 54–65.

[18] Muhammed Emin Ozturk, Omid Asudeh, Gerald Sabin, P. Sadayappan, and Aravind Sukumaran-Rajam. 2023. A Performance Portability Study Using Tensor Contraction Benchmarks. *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2023), 591–600.

[19] Yuxin Wang, Qiang Wang, Shaohuai Shi, Xin He, Zhenheng Tang, Kaiyong Zhao, and Xiaowen Chu. 2019. Benchmarking the Performance and Energy Efficiency of AI Accelerators for AI Training. *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)* (2019), 744–751.

[20] Carlos Reaño and Federico Silla. 2016. Performance Evaluation of the NVIDIA Pascal GPU Architecture: Early Experiences. *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (2016), 1234–1235.

[21] Mike Mantor. 2012. AMD Radeon™ HD 7970 with graphics core next (GCN) architecture. In *2012 IEEE Hot Chips 24 Symposium (HCS).* 1–35.

[22] AMD. 2020. *"Vega" Instruction Set Architecture Reference Guide.* Advanced Micro Devices, Inc. Retrieved 27-January-2020 from https://gpuopen.com/amd-vega-7nm-instruction-set-architecture-documentation/

[23] AMD. 2021. *OpenCL Programming Guide.* Advanced Micro Devices, Inc. Retrieved October-2021 from https://cgmb-rocm-docs.readthedocs.io/en/latest/Programming_Guides/Opencl-programming-guide.html#memory-arch

[24] Xinxin Mei and Xiaowen Chu. 2015. Dissecting GPU Memory Hierarchy Through Microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems* 28 (2015), 72–86.

[25] Xuanhua Shi, Zhigao Zheng, Yongluan Zhou, Haici Jin, Ligang He, Bo Liu, and Qiangsheng Hua. 2018. Graph Processing on GPUs. *ACM Computing Surveys (CSUR)* 50 (2018), 1 – 35.

[26] Peter Zhang, Marcin Zalewski, Andrew Lumsdaine, Samantha Misurda, and Scott McMillan. 2016. GBTL-CUDA: Graph Algorithms and Primitives for GPUs. *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2016), 912–920.

[27] NVIDIA. 2018. *"nvGRAPH".* Retrieved October 30, 2018 from https://developer.nvidia.com/nvgraph

[28] Carl Yang, Aydın Buluç, and John Douglas Owens. 2019. GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU. *ACM Transactions on Mathematical Software (TOMS)* 48 (2019), 1 – 51.

[29] Farzad Khorasani, Keval Vora, Rajiv Gupta, and Laxmi N. Bhuyan. 2014. CuSha: vertex-centric graph processing on GPUs. In *IEEE International Symposium on High-Performance Parallel Distributed Computing*.

[30] Todd Eavis and Ahmad Taleb. 2007. Mapgraph: efficient methods for complex olap hierarchies. In *International Conference on Information and Knowledge Management*.

[31] Yu Zhang, Da Peng, Xiaofei Liao, Hai Jin, Haikun Liu, Lin Gu, and Bingsheng He. 2021. LargeGraph. *ACM Transactions on Architecture and Code Optimization (TACO)* 18 (2021), 1 – 24.

[32] Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, and John McPherson. 2013. From "Think Like a Vertex" to "Think Like a Graph". *Proc. VLDB Endow.* 7 (2013), 193–204.

[33] Seunghwa Kang, Chuck Hastings, Joe Eaton, and Brad Rees. 2023. cuGraph C++ primitives: vertex/edge-centric building blocks for parallel graph computing. *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2023), 226–229.

[34] Yujiang Bi, Yi Xiao, Weiyi Guo, Ming Gong, Peng Sun, Shun Xu, and Yi-Bo Yang. 2020. Lattice QCD GPU Inverters on ROCm Platform. *EPJ Web of Conferences* (2020).

[35] Yannan Zhang and Hongyan Qian. 2020. Porting and Optimizing G-BLASTN to the ROCm-based Supercomputer. *2020 International Conference on Computer Science and Management Technology (ICCSMT)* (2020), 73–77.

[36] Jisheng Zhao, Colleen Bertoni, Jeffrey Young, Kevin Harms, Vivek Sarkar, and Brice Videau. 2023. HIPLZ: Enabling performance portability for exascale systems. *Concurrency and Computation: Practice and Experience* 35 (2023).

[37] Jack Choquette and Wishwesh Gandhi. 2020. NVIDIA A100 GPU: Performance & Innovation for GPU Computing. *2020 IEEE Hot Chips 32 Symposium (HCS)* (2020), 1–43.

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

[39] Ryan A. Rossi and Nesreen Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI Conference on Artificial Intelligence*. https://networkrepository.com

[40] H. Martin Bucker and Christian Sohr. 2014. Reformulating a Breadth-First Search Algorithm on an Undirected Graph in the Language of Linear Algebra. *2014 International Conference on Mathematics and Computers in Sciences and in Industry* (2014), 33–35.

[41] En-Ming Huang, Bo Wun Cheng, Meng-Hsien Lin, Chun-Yi Lee, and Tsung-Tai Yeh. 2024. WER: Maximizing Parallelism of Irregular Graph Applications Through GPU Warp EqualizeR. *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)* (2024), 201–206.

[42] Mauro Bisson and Massimiliano Fatica. 2017. High Performance Exact Triangle Counting on GPUs. *IEEE Transactions on Parallel and Distributed Systems* 28 (2017), 3501–3510.

[43] Zhigao Zheng, Guojia Wan, Jiawei Jiang, Chuang Hu, Hao Liu, Shahid Mumtaz, and Bo Du. 2024. Lock-free Triangle Counting on GPU. *IEEE Trans. Comput.* (2024).

[44] Jie Chen and Yousef Saad. 2012. Dense Subgraph Extraction with Application to Community Detection. *IEEE Transactions on Knowledge and Data Engineering* 24 (2012), 1216–1230.

[45] Zihao Yu, Ningyi Liao, and Siqiang Luo. 2024. GENTI: GPU-powered Walk-based Subgraph Extraction for Scalable Representation Learning on Dynamic Graphs. *Proc. VLDB Endow.* 17 (2024), 2269–2278.

[46] Kai Siong Yow, Ningyi Liao, Siqiang Luo, and Reynold Cheng. 2023. Machine Learning for Subgraph Extraction: Methods, Applications and Challenges. *Proc. VLDB Endow.* 16 (2023), 3864–3867.

[47] Bingchao Li, Jizeng Wei, Ji zhou Sun, Murali Annavaram, and Nam Sung Kim. 2019. An Efficient GPU Cache Architecture for Applications with Irregular Memory Access Patterns. *ACM Transactions on Architecture and Code Optimization (TACO)* 16 (2019), 1 – 24.