GARDENIA: A Graph Processing Benchmark Suite for Next-generation Accelerators

ZHEN XU, XUHAO CHEN, JIE SHEN, YANG ZHANG, CHENG CHEN, and CANQUN YANG, National University of Defense Technology, China

This paper presents the Graph Algorithm Repository for Designing Next-generation Accelerators (GARDENIA), a benchmark suite for studying irregular graph algorithms on massively parallel accelerators. Applications with limited control and data irregularity are the main focus of existing generic benchmarks for accelerators, while available graph processing benchmarks do not apply state-of-the-art algorithms and/or optimization techniques. GARDENIA includes emerging graph processing workloads from graph analytics, sparse linear algebra and machine learning domains, which mimic massively multithreaded commercial programs running on modern large-scale datacenters. Our characterization shows that GARDENIA exhibits irregular microarchitectural behavior which is quite different from structured workloads and straightforward-implemented graph benchmarks.

CCS Concepts: • **Computer systems organization** → **Architectures**; • **Computing methodologies** → *Parallel computing methodologies*; • **Theory of computation** → Design and analysis of algorithms;

Additional Key Words and Phrases: Benchmark Suite, Graph Processing, Massive Multithreading, Irregular Workloads

ACM Reference Format:
Zhen Xu, Xuhao Chen, Jie Shen, Yang Zhang, Cheng Chen, and Canqun Yang. 2018. GARDENIA: A Graph Processing Benchmark Suite for Next-generation Accelerators. *ACM J. Emerg. Technol. Comput. Syst.* 1, 1, Article 1 (January 2018), 13 pages. https://doi.org/10.1145/3283450

1 INTRODUCTION

With the rise of graph analytics, sparse linear algebra and machine learning, graph processing applications have become increasingly important and pervasive. For these frequently executed algorithms, dedicated hardware accelerators [1, 13, 29] are an energy-efficient avenue to high performance. Meanwhile, general purpose accelerators (e.g., GPUs and MICs) which have been widely deployed in many HPC systems and datacenters, are also applied to speedup applications in these areas, such as graph analytics [27], sparse linear algebra [26] and machine learning [25]. However, these applications exhibit irregular runtime behavior which is quite different from that of conventional well-studied HPC applications. Therefore, it is important to set up a standard benchmark suite for architecture researchers to get deep understanding on these applications.

This work is partially supported by the National Natural Science Foundation of China 61502514, 61402488 and the National Key R&D Program of China 2017YFB0202003.

Authors' address: Zhen Xu, zhenx@nudt.edu.cn; Xuhao Chen, chenxuhao@nudt.edu.cn; Jie Shen, j.shen@nudt.edu.cn; Yang Zhang, zhangyang@nudt.edu.cn; Cheng Chen, chencheng@nudt.edu.cn; Canqun Yang, canqun@nudt.edu.cn, National University of Defense Technology, 109 Deya Rd, Changsha, Hunan, 410073, China.

1550-4832/2018/1-ART1 $15.00
https://doi.org/10.1145/3283450

The goal of GARDENIA[1] is to develop an irregular graph processing benchmark suite for future accelerator architecture research, including both general-purpose and domain-specific accelerator design. For general-purpose accelerators, GARDENIA is an important complement for generic benchmark suites (e.g.,Rodinia and Parboil) which have limited irregularity, and architectural support for irregular algorithms can be exploited by running the benchmarks and locating the performance bottlenecks. For domain-specific accelerators, GARDENIA can be used as a workload representative of real-world graph processing workloads, and in this way it enables the hardware specialization for these algorithms. Besides, GARDENIA benchmarks can be also used by algorithm, library and compiler designers as reference implementations for comparison.

As far as we know, GARDENIA is the first graph processing benchmark suite specifically targeting accelerator architecture reserach. It is different from previous GPU graph processing benchmark suites in many ways: 1) The benchmark implementations incorporate *state-of-the-art* optimization techniques for massively parallel accelerators; 2) The input graphs are selected appropriately for our target platforms, in terms of graph size, desity and topology; 3) The workloads are representatives of popular graph processing algorithms running on modern datacenters.

This paper makes the following three major contributions:

- It unveils the limitations of previously available benchmark suites and identifies the reason for which they are not good candidates to evaluate future accelerators.
- We present GARDENIA, a graph processing benchmark suite for accelerators that provides irregularity, diversity and state-of-the-art techniques.
- We characterize GARDENIA, illustrate its microarchitecture behavior, and analyze the performance bottlenecks to provide insights on how to design next-generation accelerators with energy-efficiency for irregular workloads.

The rest of the paper is organized as follows: Section 2 explains the motivation of this work. Section 3 describes the design of our benchmark suite. Then we characterize the benchmarks in Section 4. And section 5 concludes.

2 MOTIVATION

This work aims to develop an irregular graph processing benchmark suite that can help designing next-generation accelerators capable of accelerating real-world irregular (not only HPC) applications on modern datacenters. In section 2.1, we show what features such a benchmark suite should equip. Then in section 2.2, we discuss how existing benchmark suites cannot fulfill these conditions.

2.1 Benchmark Requirements

A benchmark suite of real-world irregular workloads on data parallel accelerators needs to meet the following requirements:

Massive Parallelism and Accelerator Coverage Massively parallel accelerators, such as GPUs and MICs (Xeon Phi coprocessors) have been already widely deployed in today's high performance servers and datacenters. Many important applications are highly dependent on accelerators to achieve required performance. Future accelerators' main feature is to provide tremendous performance using extreme throughput and memory bandwidth. Meanwhile, to take full advantage of the hardware, workloads should be well paralleled.

[1]The source code can be found at github.com/chenxuhao/gardenia

Workload Irregularity Regular applications have been intensively investigated on large-scale parallel systems in the past decade. Many scientific and commercial applications, have been successfully mapped onto accelerators. Despite this progress, data-parallel accelerators (DPAs) continue to be confined to structured parallelism. While structured parallelism maps directly to DPAs, much more unstructured applications in real-world cannot simply take advantage of them. GARDENIA is designed to represent irregular graph applications, and can be used to guide the design of future accelerators.

State-of-the-Art Optimization Techniques Researchers have made great efforts to map general purpose applications onto GPUs during the past decade, including parallel algorithm innovations and optimization techniques. A benchmark suite should include rising optimizations to fully represent real-world applications, which are rarely straight-forward implementations. This is important because straightforward-implemented workloads may exhibit quite different microarchitectural behaviors compared to optimized ones, which are unqualified to represent real-world applications and may mislead architecture design.

Workload and Dataset Diversity With the rise of heterogeneous computing, real-world workloads are increasingly diverse. They are written in different parallelization models, run on a variety of computing platforms, and cover a wide range of application areas. Besides, the behaviors of these irregular applications change dramatically with different types of input datasets. To cover different program characteristics and behaviors, a diverse collection of both workloads and input datasets is required, so that architects can thoroughly understand the application behaviors and make reasonable trade-offs when designing an accelerator.

2.2 Existing Benchmark Suites and their Limitations

We further discuss the drawbacks of previously available benchmark suites. Table 1 shows the comparison between existing benchmark suites and GARDENIA.

Generic Benchmark Suites for Multicore CPUs. *PARSEC* [5] is an application repository for shared-memory computers. It is composed of several shared-memory multithreaded workloads from emerging programs. This suite only includes CPU programs (e.g. Pthreads, OpenMP and TBB programs) and is not specifically designed for irregular workloads.

Generic Benchmark Suites for Accelerators. *Parboil* [33] and *Rodinia* [8] are widely used GPU benchmark suites. They provide CUDA, OpenCL and OpenMP implementations. Despite the popularity, most of their benchmarks are structured kernels from HPC applications with limited control and memory irregularity. Although they do include some irregular workloads such as BFS, their implementations are quite out-of-date. Thus, they are not good candidates for studying emerging irregular applications on accelerators.

Graph Benchmarks for CPUs. *GAPBS* [3] is a graph processing benchmark suite on CPUs. It is a portable high-performance graph baseline using only OpenMP for parallelism. Their implementations are representatives of up-to-date performance on multicore CPUs. However, GAPBS does not include any implementation for accelerators. Besides, it only includes six typical graph benchmarks, which is not diverse enough to represent emerging irregular applications on accelerators.

Graph Benchmarks for GPUs. *Pannotia* [7] assembles a set of graph algorithms implemented in OpenCL, and the irregularities of these kernels are characterized on GPUs. However, due to limited knowledge on how to optimize graph algorithms on GPUs at that time, no specific optimization is applied. We will show that workloads which lack of state-of-the-art optimization techniques behaves quite differently from optimized ones in Section 4. *GraphBIG* [**?**] includes workloads for both CPU and GPU sides, using OpenMP and CUDA respectively. Since it mainly focuses on CPU versions, most of its GPU workloads are also straightforward implemented, although some of the

	No. of workloads	Accelerator oriented	Irregular	Diverse	up-to-date
GARDENIA	9	√	√	√	√
PARSEC	12	×	×	√	√
Parboil	11	√	×	√	√
Rodinia	23	√	×	√	√
GAPBS	6	×	√	×	√
Pannotia	8	√	√	√	×
GraphBIG	6	√	√	√	×
LonestarGPU	7	√	√	×	√

Table 1. Comparison between GARDENIA and Previously Available Benchmark Suites. ”No. of wrokloads” shows how many workloads are included; ”Accelerator oriented” shows if the benchmark suite is built targeting accelerators; ”Irregular” shows if the suite focuses on irregular workloads; ”Diverse” shows if the suite includes diverse workloads and datasets; ”up-to-date” shows if the suite employs up-to-date optimization techniques.

GPU workloads achieve substantial speedup. *Lonestargpu* [6] includes a couple of irregular CUDA benchmarks with advanced optimizations, but it does not focus on emerging graph processing applications. Thus it is not an appropriate candidate for designing next-generation accelerators for these domains. Besides, the datasets it provides are not diverse enough for deeply understanding graph algorithms.

Graph Processing Frameworks. Many parallel CPU (Pregel [20], GraphMat [34], Ligra [32], Graphlab [18, 19], GraphReduce [31]) and GPU (Gunrock [35], Medusa [37], CuSha [15]) graph processing frameworks are developed to provide programmability and high performance simultaneously. Efforts have been made to generalize the optimization techniques that are previously proposed in specific graph algorithms. These frameworks provide workloads that employ state-of-the-art techniques, but they can not be efficiently used for architecture research due to the complexity of their infrastructures. Note that they are not benchmarks suites, but they provide us the insight that guides our design. We applies the key optimizations in these frameworks to our workloads, making sure that our benchmark suite represents real-world applications.

Our GARDENIA benchmark suite tackles the limitations of existing benchmark suites, applies the state-of-the-art optimization techniques, and covers diverse irregular workloads and datasets. Thus, GARDENIA is a good benchmark candidate for architecture research.

3 THE GARDENIA BENCHMARK SUITE

GARDENIA is intended to provide a graph processing workload collection that represents important irregular applications running on the accelerators of modern datacenters. And in this way it can help graph processing research and accelerator design by standardizing evaluations. GARDENIA fully fulfill the demands highlighted in Section 2:

- Each workload is parallelized using OpenMP for multicore CPUs, CUDA for GPUs and OpenMP target for MICs to exploit massive parallelism.
- GARDENIA is not designed for regular HPC programs which have been well studied on accelerators in the last decade. It focuses on emerging irregular workloads that are widely employed in graph analytics, sparse linear algebra and machine learning domains.
- All the benchmarks apply state-of-the-art optimization techniques, i.e., the benchmarks are reasonably optimized to get substantial speedups on GPUs and MICs.
- The workloads as well as their input datasets are multitudinous and are carefully selected from various application domains.

3.1 Optimization Techniques

We apply the state-of-the-art algorithm innovations and optimization techniques that have been widely employed in academic and industry libraries [10, 26, 35]. Benchmarks are neither over-optimized nor unoptimized (i.e. straightforward-implemented) for the target accelerator. Techniques bound to specific architectures are not included in our suite.

Mapping Strategies. Basically there are two parallelism strategies for graph algorithms: *quadratic* mapping and *linear* mapping [24]. Quadratic mapping checks every vertex of the graph in each iteration, and depending on if the vertex needs to be processed a thread may process its vertex or stay idle [21]. Quadratic mapping is intuitive and used for sequential-movement benchmarks. By contrast, a frontier queue is maintained in linear mapping stargegy to hold the to be processed vertices during each iteration. Then corresponding threads are initialized and the number of threads is due to amount of vertices to be processed. Same number of to be processed vertices are distributed to each thread and therefore all threads are working till they terminate at the same time. [21]. In this way, linear mapping is commonly more efficient than quadratic mapping, but extra overhead arises from maintaining the frontier. We use linear mapping for traversal-based benchmarks (see section 3.2 for details).

Load Balancing. Load imbalance problem is one of the key issues for irregular algorithms, and is particularly worse for scale-free (power-law) graph datasets. Hong *et al.* [14] proposed a warp-execution method for BFS to map warps instead of threads to vertices. This scheme is applied in most of the GARDENIA benchmarks. Based on Hong's scheme, Merrill *et al.* [21] proposed a multi-level load balancing scheme. According to the size of a vertex's neighbor list, the workload of a vertex may be distributed to a thread, a warp or a thread block. This strategy turns out to be quite efficient on GPUs, and is employed in traversal-based benchmarks in GARDENIA.

Push vs. Pull Beamer *et al.* [2] proposed direction-optimizing BFS which uses two different methods to drive BFS: push and pull. The push method detects the status of vertices in current frontier and scatters them to their outgoing neighbors to create new frontier. The push method is intuitive and commonly used. Conversely, the pull method checks unvisited vertices whose parents are in the current frontier and gathers status from the incoming neighbors. The pull method is beneficial if the unvisited vertices' number is less than the frontier size [35].

Reordering Queue. The order of vertices in the frontier queue decides the computation order, memory access order and load balance, and therefore affects performance. By ochestrating the order of vertices for processing, many graph algorithms can reduce overall computation workload or memory access latency. For example, we use the delta-stepping [22] implementation for OpenMP SSSP on CPUs and MICs. This approach organizes vertices in the output frontier into "bin" or "buckets" by their distances to the source vertex and those vertices with smaller distances get to be processed first.

Other Techniques. We also include architectural optimization techniques to take advantages of the underlying hardware. For example, we use the *texture cache* to hold the read-only data, i.e. the x vector in spmv, and thus the read-only data is forced to be cached in the L1 read-only cache which has much shorter latency than the off-chip DRAM. For MIC benchmarks, we enable *vectorization* to make full use of the compute capability. We also use *bit operations* to reduce computation and storage overhead, i.e., in linear mapping algorithms on the frontier queue, the current frontier is converted into a bitmap of vertices, and we get to know whether a vertex is active only by checking if they are valid in the bitmap. For many CUDA implementations, *kernel fusion* combines multiple kernels into a single one, and thus can keep the entire program on the accelerator. This is beneficial because it leverages producer-consumer locality between adjacent kernels [35].

3.2 Diverse Irregular Workloads

The GARDENIA suite currently includes the following 9 workloads(more on the way to release):

Breadth-First Search (BFS) is a key graph primitive which traverses a graph in breadth-first order [21]. It starts at the tree root and explores the neighbor nodes first before moving to the next level neighbours. We implement it using the linear mapping strategy along with Merrill's load balancing technique.

Single-Source Shortest Paths (SSSP) finds the shortest path from a specific source vertex to every other reachable vertices in a directed graph [11]. We implement delta-stepping algorithm for CPUs and Bellman-Ford algorithm for GPUs. SSSP is a traversal-based algorithm and we apply similar techniques as in BFS.

Betweenness Centrality (BC) is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph there exists at least one shortest path between them. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex. BC is also traversal-based.

PageRank (PR) ranks a website based on the score of its neighboring sites (i.e. the sites that link to it) [30]. Each vertex is given a initial score at the beginning. At each iteration, it scans each vertex and updates its score using the its neighbors' scores and degrees. It iterates until the score change is small enough. Unlike above traversal-based benchmarks, PR has sequential-movement access pattern and so do all the following benchmarks.

Connected Components (CC) All the vertices of a conncected component are assigned a unique label [4]. In each iteration, the label is propagated from a smaller-id vertex to its larger-id neighbors. The algorithm iterates until convergence (no propogation happens).

Triangle Counting (TC) counts how many triangles there are in an undirected graph [36]. It is essential in graph statistics applications (e.g. clustering coefficients). To find triangles, it computes the neighbor list intersection of each vertex and its neighbor.

Stochastic Gradient Descent (SGD) is a stochastic approximation of the gradient descent optimization. It decomposes the *ratings* matrix into two feature vectors (*users* and *items*), and it uses the dot product of *users* and *items* to make a rating prediction. It iterates until the error between prediction and the actual rating is small enough.

Sparse Matrix-Vector Multiplication (SpMV) is one of the most heavily used primitives in scitific computations [36]. It identifies the elements in a matrix that are non-zero and multiplies them in a vector.

Symmetric Gauss-Seidel smoother (SymGS) is an important primitive in High Performance Conjugate Gradients (HPCG) [10]. It carries out a forward and backward triangular solve whose operation is similar to SpMV but in a data-dependent order. We use vertex coloring [9] to find parallelism and arrange the order of tasks.

3.3 Diverse Input Datasets

A major characteristic of irregular applications is the input dependency. Research has shown that graph properties substantially affect performance [3]. Therefore, we should characterize graph workloads not only by the algorithms, but also by the properties of graph instances used.

Graph Size Due to the rapid increase of data size in modern datacenters, it has become problematic to manage and manipulate the large volume of data with relatively limited hardware resources. Our selected datasets should include large enough graph instances. On the other hand, to support architecture research, small but representative datasets are also needed to guarantee that the simulation finishes in reasonable time.

Graph Density/Sparsity The average degree of vertices is usually used to represent the density of a graph. This parameter affects the computation intensity, irregularity, and the amount of locality exists in graph algorithms. A denser graph is likely to have more data reuse since vertices may be visited or updated more times. High density also means more regular memory access behavior. However, denser graphs have more edges, which implies larger working set. In most case real-world graphs are sparse, which makes graph analytics irregular.

Graph Topology In [3], graphs are divided into two categories in terms of topology: meshes and social networks. Meshes usually have ralatively higher diameter and lower degree distribution because they are usually obtained from physical world, such as electric transmission network. By contrast, social networks are non-spatial, and they usually have a power-law degree distribution (scale-free). Due to degrees (number of neighbors) of different vertices may vary much in a social network, load balancing is a major issue for parallel execution [3].

Graph	# Vertices	# Edges	Avg. deg.	Description
cage14	1.5M	27M	18	DNA electrophoresis
Freescale	3.0M	23M	7.7	Circuit simulation
Flickr	2.3M	33M	14.4	Social relationship inside flickr.com
Web-Google	0.9M	5.1M	5.8	Webpage graph from Google
Youtube	1.1M	4.9M	4.3	Relationships of Youtube users
LiveJ	4.8M	69M	14.2	LiveJournal friendship network
Soc-Pokec	1.6M	31M	18.8	Pokec user relationship
Wikipedia	3.1M	39M	12.5	Wikipedia page link graph

Table 2. Several datasets of GARDENIA

In summary, we select our datasets from the UF Sparse Matrix Collection [12], the SNAP dataset Collection [17], and the Koblenz Network Collection [16] which are all publicly available. Size, density, topology and application domains vary among these graphs. For example, twitter is a large "social network" graph that can be used to evaluate real machines, while road is a relatively small-sized graph with "mesh" topology that can be used in simulation. We list part of the our datatsets in Table 2.

4 EVALUATION

In this section, we evaluate GARDENIA in terms of workload irregularity, optimization integration, workload and dataset diversity and accelerator coverage, showing the necessity of developing GARDENIA as an irregular workload benchmark suite and its main feature.

4.1 Experimental Setup

To briefly show the feature of GARDENIA, we test and compare 3 implementations in this Section including: (1) Serial: Serial implementation on CPU, (2) CUDA: CUDA implementation on GPU. (3) MIC: OpenMP target implementation on MIC(Xeon Phi).

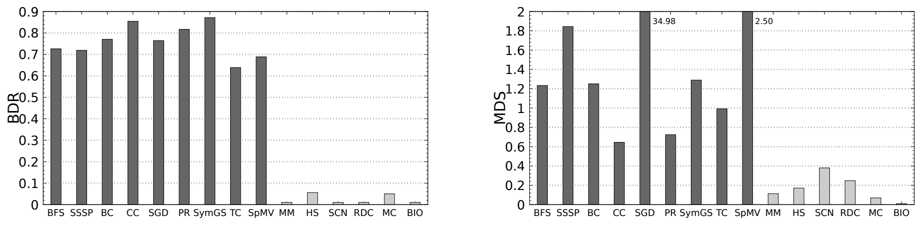
In Section 4.2 to Section 4.5: Serial experiments are executed on Intel Xeon E5-2620 2.10 GHz 6-core CPU. We perform the CUDA experiments on NVIDIA K40m GPU with CUDA Toolkit 8.0 release. For the MIC experiments, we launch 228 threads on MIC, 4 threads on each of 57 cores. Option -O3 and -arch=sm_35 are used respectively for gcc and nvcc to compile the CUDA implementations.

In Section 4.3 we further evaluate GARDENIA on a newer HW/SW platform. The CUDA experiments are performed on NVIDIA Geforce GTX 1080 GPU with CUDA 9.1.

All the benchmarks are executed for 10 times and the average execution result is collected. We only focus on the computation part of each program's timing. For the CUDA implementation test, we exclude the data in/out transfer time, which is about 10%-15% of the entire execution time. In section 4.2 and 4.3 we find that different datasets lead to the same conclusion, thus we only show experimental results with one dataset due to space limit.

4.2 Workload Irregularity

Branch divergence and memory divergence are two critical factors that reflect workload irregularity. We use two metrics, BDR and MDS, to measure them. BDR (Branch Divergence Rate) is the average ratio of inactive threads per warp, which reflects the amount of branch divergence[23]. MDS (Memory Divergence Scale) is the average number of executed instruction's replays due to memory issues. The replay will be repeated until all requested data are ready, therefore MDS may be greater than 1. Higher BDR and MDS mean higher branch divergency and memory divergency, respectively.



(a) BDR (b) MDS

Fig. 1. The difference between irregular and regular workloads.

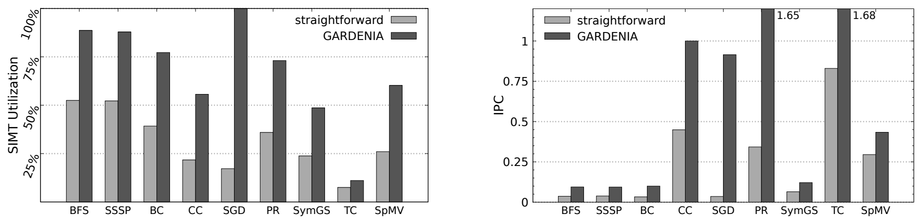
We select 9 workloads from GARDENIA as irregular workloads along with 6 workloads from NVIDIA SDK [28] and Rodinia [8] as regular ones for comparison. The regular workloads cover various algorithms in specific application domains: Matrix Multiplication (MM) is from the dense liner algebra area; Hotspot (HS) performs physical simulation; Scan (SCN) and Reduction(RDC) are widely used parallel computing primitives; MonteCarlo (MC) and Binomial Options (BiO) are financial applications. We run all the workloads with the same input dataset web-Google [17].

Figure 1 shows that GARDENIA workloads have much higher BDR and MDS compared to the regular ones, which demonstrates that irregular workloads have much higher branch and memory diversities. The high irregularity makes irregular workloads behave significantly different from structured benchmarks found in previous generic suites, which implies the necessity of constructing an irregular benchmark suite like GARDENIA.

4.3 The Necessity of Integrating Optimizations

To closely mimic real-world applications, we select and generalize state-of-the-art graph processing optimization techniques, and integrate these optimizations for each workload in GARDENIA. This is essential because straightforward implementations may show largely different microarchitectural behaviors and mislead architecture design. We compare GARDENIA and straightforward CUDA implementations in this section to show how optimization leads to different application behavior, certifying the necessity of integrating optimizations in GARDENIA as a benchmark suite.

GARDENIA includes different versions for each workload by applying incremental optimizations. In this section, the GARDENIA workloads we employ for comparison are the ones with best performance. We run all the workloads with the same input dataset soc-LiveJ [12].



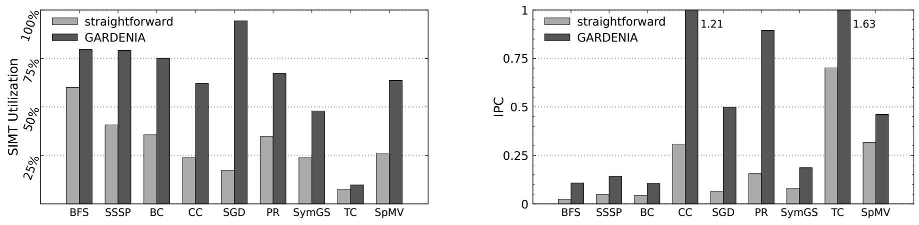
(a) SIMT utilization (b) IPC

Fig. 2. Optimization effects measured by the SIMT utilization and IPC.

Fig. 2 compares the SIMT utilization and IPC(Instructions per cycle) of the *straightforward* implementations (similar to those provided in Pannotia [7] and GraphBIG [23]) and the *GARDENIA* implementations with optimizations applied. We observe that the SIMT utilization as long as the executed IPC of the GPU are largely increased by applying optimizations, leading up to 3.8 × performance improvement.

This huge difference indicates that optimization techniques can substantially change program behavior, so it is necessary to include up-to-date optimization techniques in GARDENIA so as to closely mimic the behavior of real-world applications which are most likely built using these optimizations.

As a benchmark suite, GARDENIA will be used to evaluate different hardware/software platforms. So we retake the evaluation of Fig. 2 to illustrate that the optimizations still matter on a newer platforam (Nivida GTX 1080 with CUDA Toolkit 9.1). As shown in Fig. 3, when compared to Fig. 2, though the absolute values of some workloads vary due to different hardware and software generations, the difference between the straightforward version and GARDENIA remains the same trend. The optimizations employed in GARDENIA still lead to different program behavior on the newer platform. It implies that as a benchmark suite, GARDENIA is not sensitive to different HW/SW environments and able to mimic real-world applications no matter what test environment it is.

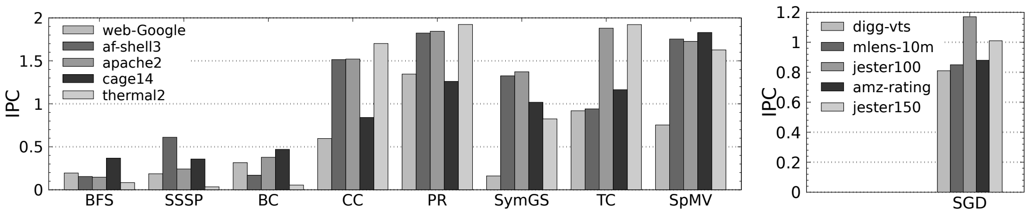


(a) SIMT utilization (b) IPC

Fig. 3. Optimization effects measured by the SIMT utilization and IPC on new testbed

4.4 Workload and Dataset Diversity

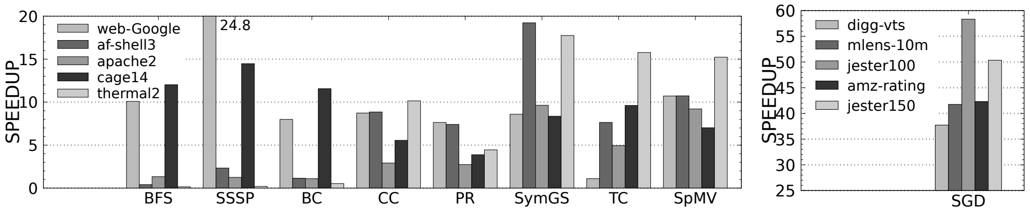
To represent emerging irregular workloads for next-generation accelerators, GARDENIA selects representative workloads from various application domains(as is described in Section 3.2). Furthermore, as graph processing performance depends not only on the workload but also on the dataset to run, a careful collection of graph datasets is of necessity. We show this with the CUDA implementations of 8 GARDENIA workloads with 5 different datasets [12]. SGD uses quite different datasets compared to the other 8 workloads, so we test SGD alone with 5 other compatible datasets [16].



(a) (b)

Fig. 4. The program IPC for different workloads and datasets.

Fig. 4(a) shows the IPC for 8 workloads running with 5 input datasets. In each workload cluster, IPC varies widely among different datasets. For the same dataset, different workloads may lead to entirely different IPCs . For example, for dataset thermal2, BFS, SSSP and BC have low IPCs which are below 0.1, while CC, PR, TC and SpMV have much higher IPCs which are above 1.5. Fig. 4(b) shows IPC of SGD also varies with different datasets.



(a) (b)

Fig. 5. The program speedup for different workloads and datasets.

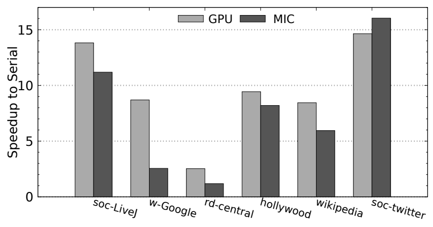
Fig. 5(a) shows the speedup of the 8 workloads in comparison with the CPU serial implementations with 5 input datasets. It shows similar behavior like Fig. 4(a). The speedups vary widely for the same workload with different datasets. For the same dataset, different workloads lead to entirely different speedups. Fig. 5(b) shows speedup of SGD varies with different input datasets.

Thus, architecture research needs a diverse set of both workloads (in terms of application domains and program behaviors) and input datasets (in terms of graph size, density, and topology) to make microarchitecture characteristics fully manifested, and GARDENIA is designed to meet this requirement.

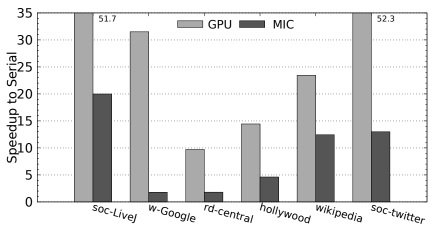
4.5 Accelerator Coverage

GARDENIA targets future accelerator architecture research, including GPU and MIC. It includes both GPU and MIC implementations of all workloads, along with CPU OpenMP implementation. To demonstrate the difference between MIC and GPU, we show their speedups of running BFS and SSSP with 6 input datasets [12].

Fig. 6 shows the speedup of GPU and MIC implementations to the serial CPU implementation of the BFS and SSSP with 6 different datasets(other workloads lead to similar results, and we use BFS and SSSP as examples due to space limit). Similar to GPU, graph workloads are also data sensitive on MIC. For example, BFS with input *soc-twitter* running on MIC can get speedup of 16.0 × while



(a) BFS



(b) SSSP

Fig. 6. Speedup comparison between GPU and MIC.

the corresponding speedup is only 1.2 × for input *road-central*. Due to MIC's lower core number and flexibility, speedup on MIC are relatively lower than that on GPU. For all the 6 datasets, SSSP gets significantly higher speedup running on GPU than on MIC.

In fact, due to different features of MIC and GPU, irregular workloads running on them have significantly different microarchitecture behaviors. Thus, it is important to include both GPU and MIC implementations in GARDENIA. We don't discuss in depth the microarchitectural difference details in this paper.

5 CONCLUSION

In this paper, we present GARDENIA, a graph processing benchmark suite for studying irregular algorithms on massively parallel accelerators. With workloads and datasets carefully selected as well as state-of-the-art techniques employed, GARDENIA is designed to represent emerging real-world applications on modern datacenters, and to facilitate accelerator architecture research. Our characterization illustrates that GARDENIA benchmarks exhibit quite different microarchitectural behavior from structured workloads and straightforward-implemented graph benchmarks, and thus demonstrating the necessity of constructing such a suite.

REFERENCES

[1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News* 43, 3, 105–117.

[2] Scott Beamer, Krste Asanović, and David Patterson. 2013. Direction-optimizing breadth-first search. *Scientific Programming* 21, 3-4, 137–148.

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

[4] Scott Beamer, Krste Asanovic, and David A. Patterson. 2015. The GAP Benchmark Suite. *CoRR* abs/1508.03619 (2015). http://arxiv.org/abs/1508.03619

[5] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques.* ACM, 72–81.

[6] M. Burtscher, R. Nasre, and K. Pingali. 2012. A quantitative study of irregular programs on GPUs. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC) (IISWC '12).* 141–151.

[7] Shuai Che, Bradford M Beckmann, Steven K Reinhardt, and Kevin Skadron. 2013. Pannotia: Understanding irregular GPGPU graph applications. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on.* IEEE, 185–195.

[8] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC 2009.* Ieee, 44–54.

[9] Xuhao Chen, Pingfan Li, Jianbin Fang, Tao Tang, Zhiying Wang, and Canqun Yang. 2017. Efficient and High-quality Sparse Graph Coloring on the GPU. *Concurrency and Computation: Practice and Experience* 29, 10 (April 2017).

[10] Steven Dalton, Nathan Bell, Luke Olson, and Michael Garland. 2014. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. (2014). http://cusplibrary.github.io/ Version 0.5.0.

[11] A. Davidson, S. Baxter, M. Garland, and J. D. Owens. 2014. Work-Efficient Parallel GPU Methods for Single-Source Shortest Paths. In *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS)*. 349–359. https://doi.org/10.1109/IPDPS.2014.45

[12] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1.

[13] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi. 2016. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13. https://doi.org/10.1109/MICRO.2016.7783759

[14] Sungpack Hong, Sang Kyun Kim, Tayo Oguntebi, and Kunle Olukotun. 2011. Accelerating CUDA graph algorithms at maximum warp. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 267–276.

[15] Farzad Khorasani, Keval Vora, Rajiv Gupta, and Laxmi N. Bhuyan. 2014. CuSha: Vertex-centric Graph Processing on GPUs. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing (HPDC) (HPDC '14)*. ACM, New York, NY, USA, 239–252. https://doi.org/10.1145/2600212.2600227

[16] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1343–1350.

[17] J. Leskovec. 2013. SNAP: Stanford Network Analysis Platform. (2013). http://snap.stanford.edu/data/index.html

[18] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *Proceedings of the VLDB Endowment* 5, 8 (April 2012), 716–727. https://doi.org/10.14778/2212351.2212354

[19] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. Graphlab: A new parallel framework for machine learning. In *Proceedings of the UAI*. 340–349.

[20] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 135–146.

[21] Duane Merrill, Michael Garland, and Andrew Grimshaw. 2012. Scalable GPU Graph Traversal. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) (PPoPP '12)*. ACM, New York, NY, USA, 117–128.

[22] Ulrich Meyer and Peter Sanders. 2003. Δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms* 49, 1 (2003), 114–152.

[23] Lifeng Nai, Yinglong Xia, Ilie G Tanase, Hyesoon Kim, and Ching-Yung Lin. 2015. GraphBIG: understanding graph computing in the context of industrial solutions. In *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*. IEEE, 1–12.

[24] R. Nasre, M. Burtscher, and K. Pingali. 2013. Data-Driven Versus Topology-driven Irregular Computations on GPUs. In *Proceedings of the 27th IEEE International Parallel Distributed Processing Symposium (IPDPS) (IPDPS '13)*. 463–474.

[25] NVIDIA. 2016. cuDNN Library. (2016). https://developer.nvidia.com/cudnn/

[26] NVIDIA. 2016. CUSPARSE Library. (2016). http://docs.nvidia.com/cuda/cusparse/

[27] NVIDIA. 2016. nvGRAPH Library. (2016). http://docs.nvidia.com/cuda/nvgraph/index.html

[28] GPU Nvidia. 2013. computing SDK. *Gpu computing sdk," Avalible at: https://developer. nvidia. com/gpu-computing-sdk* 22, 07 (2013), 2013.

[29] Muhammet Mustafa Ozdal, Serif Yesil, Taemin Kim, Andrey Ayupov, John Greth, Steven Burns, and Ozcan Ozturk. 2016. Energy efficient architecture for graph analytics accelerators. In *ISCA 2016*. IEEE, 166–177.

[30] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[31] Dipanjan Sengupta, Shuaiwen Leon Song, Kapil Agarwal, and Karsten Schwan. 2015. GraphReduce: processing large-scale graphs on accelerator-based systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 28.

[32] Julian Shun and Guy E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) (PPoPP '13)*. ACM, New York, NY, USA, 135–146. https://doi.org/10.1145/2442516.2442530

[33] John A. Stratton, Christopher Rodrigrues, I-Jui Sung, Nady Obeid, Liwen Chang, Geng Liu, and Wen-Mei W. Hwu. 2012. *Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing*. Technical Report IMPACT-12-01. UIUC, Urbana. http://impact.crhc.illinois.edu/Shared/Docs/impact-12-01.parboil.pdf

[34] Narayanan Sundaram, Nadathur Satish, Md Mostofa Ali Patwary, Subramanya R. Dulloor, Michael J. Anderson, Satya Gautam Vadlamudi, Dipankar Das, and Pradeep Dubey. 2015. GraphMat: High Performance Graph Analytics Made Productive. *Proc. VLDB Endow.* 8, 11 (July 2015), 1214–1225. https://doi.org/10.14778/2809974.2809983

[35] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D Owens. 2016. Gunrock: A high-performance graph processing library on the GPU. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 11.

[36] Xiangyao Yu, Christopher J Hughes, Nadathur Satish, and Srinivas Devadas. 2015. IMP: Indirect memory prefetcher. In *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 178–190.

[37] J. Zhong and B. He. 2014. Medusa: Simplified Graph Processing on GPUs. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (June 2014), 1543–1552. https://doi.org/10.1109/TPDS.2013.111

Received January 2018; revised May 2018; revised August 2018; accepted October 2018