2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

Architectural Implications in Graph Processing of Accelerator with Gardenia Benchmark Suite

Yang Zhang*, Jie Shen*, Zhen Xu*, Shikai Qiu*, Xuhao Chen†
*Department of Computer Science and Technology, National University of Defense Technology, Changsha, China
†Institute for Computational Engineering and Sciences, University of Texas at Austin, USA

***Abstract*—Existing generic benchmarks for accelerators (e.g. Parboil and Rodinia) have focused on high performance computing (HPC) applications which have limited control flows and data irregularity. Previous available graph analytics benchmark suites include straightforward implemented workloads which do not employ up-to-date optimization techniques and thus have quite different behaviors from real-world applications. This paper first briefly presents and characterizes the Graph Analytics Repository for Designing Next-generation Accelerators (GARDENIA)[1], which is a benchmark suite for studies of irregular algorithms on various massively parallel accelerators. It includes emerging irregular big-data and machine learning applications, in which mimic massively multithreaded programs deployed on not only datacenters but also hand-on devices. Then we characterize Nvidia GPU with GARDENIA, covering a wide spectrum of metrics such as parallelization, cache locality, off-chip traffic and irregularity. Based on the characterization on Nvidia GPU, we unveil the performance bottlenecks of the current mainstream accelerator and give architectural insights for building high performance and energy-efficient domain-specific accelerators for graph applications.**

***Index Terms*—benchmark suite, performance measurement, massive multithreading, graph analytics**

I. Introduction

When we plan to design a custom accelerator for graph analytics, the first thing is to acquire a real-world graph benchmark suite to evaluate the architecture of the accelerator. As the evolvement of the algorithms, many good optimization methods are applied to the programs. However, current graph analysis workloads are much naive and not well optimized. So we want to provide a collection of high performance workloads to more accurately reflect the real codes' behaviors and to help us reveal the real architectural implications. On the other hand, the up-to-date accelerators are much various in their micro-architecture, different accelerators may have different characteristics. To make our research results more general, we adopt a widely used Kepler GPU to research on the architectural implication of accelerator, with multicore CPU as performance baseline. We aim to offer comprehensive evaluations on accelerators for real world applications.

Knowledge extraction and analytics on graph data structures have become a hot spot in today's large-scale datacenters, such as web search engines, social networks and recommender systems. For some algorithms which are frequently executed on hardware , dedicated hardware accelerators are more energy-efficient choices compared with CPU [1], [2]. On the other hand, general-purpose accelerators (e.g. GPUs and MICs) are trying to expand their application areas such as graph analytics [3], machine learning [4] and sparse linear algebra [5]. Given the above trends and implications, it is quite necessary to build a standardized benchmark suite for characterizing and measuring the hardware accelerator design.

The goal of GARDENIA is to create a suite of emerging *irregular* workloads that can drive researches on accelerator architecture. For general-purpose accelerators, GARDENIA is an important complement for generic benchmark suites (e.g. Rodinia and Parboil) as its workloads contain much more irregularities than those structured benchmarks, and architectural implications for irregular workloads can be found by running them and analyzing the performance outputs. For accelerators used in specific domain [6], [7], [8], [9], however, GARDENIA offers a collection of workloads, which represent graph analytics, and the underlying hardware design should be specialized for these algorithms.

GARDENIA is a graph analytics benchmark suite specifically targeting researches on accelerator architecture. GARDENIA is quite different from other benchmark suites for graph analytics. The main difference is the benchmarks incorporate new optimization methods for massively parallel accelerators, which behave quite differently from those straightforward implemented benchmarks in previous available benchmark suites to facilitate architectural research. Our experiments on GPU reveal some new and different insights from prior knowledge about graph algorithms on many core coprocessors. Based on the observations, we give suggestions to researchers who want to design many core coprocessors with higher performance.

This paper makes three contributions:

- Based on analysis on previous benchmark suites, we find they are not suitable to evaluate future accelerators for graph processing.
- We briefly introduce GARDENIA, a domain-specific benchmark suite which provides irregularity and diversity to allow architectural exploration for future accelerators.
- We use GARDENIA to evaluate several aspects of a commonly used Nvidia's GPU, including parallelization, cache locality, off-chip traffic and irregularity. We give an analysis on its performance bottlenecks and give advice on coprocessor designers.

The rest of the paper is organized as follows: Section II shows what motivates our work. The design of the benchmark suite is described in Section III. Section IV explains the

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

978-1-7281-4328-6/19/$31.00 ©2019 IEEE
DOI 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00191

experimental methodology. We evaluate the performance of GPU with the benchmarks in Section V. Section VI concludes.

II. Motivation

This work is aimed to introduce a benchmark suite to be used to evaluate the design of next-generation accelerators which are used to accelerate real-world irregular applications. In the motivation section, we first present why we need such a benchmark suite. Then we will show why the existing suites are not suitable for our evaluations.

A. Requirements for a Benchmark Suite

The following five requirements call for a domain-specific benchmark suite targeting future accelerators:

Massively Parallel Applications Massively parallel accelerators, such as GPUs and MICs (Xeon Phi coprocessors) are commonly used in current high performance servers and datacenters. Those essential big-data analysis and machine learning engines that drive many important applications often use accelerators for good performance. For example, Google Brain is driven by an ocean of GPUs. The trend for future accelerators is to deliver higher performance through extreme throughput and memory bandwidth. Consequently, applications with massive parallelization can utilize the additional processing power for high performance.

Emerging Irregular Workloads Regular applications have been intensively investigated on GPGPUs in the past decades. Many applications with structured compute and memory access patterns have been successfully mapped to GPUs, including image processing, signal processing, physics simulation, finance analysis, computational biology, machine learning etc. Despite this progress, GPU makes its best when applications have structured parallelism with little irregularity. Structured parallelism matches well with data-parallel accelerators, but more unstructured applications cannot simply take advantage of them. These irregular applications which widely exists in real-world behave quite differently from structured applications [10]. Future accelerators will be designed to meet the demands of emerging irregular applications and a good benchmark suite should have the ability to represent them.

Employ State-of-the-Art Techniques Plenty of efforts have been made to map applications onto accelerators during the past decades. These work include both parallel algorithm innovations and optimization techniques. A benchmark should be able to incorporate state-of-the-art techniques. Because different microarchitectural behaviors may exhibit between straightforward implementation and optimized one, misleading the workload users.

Diverse Heterogeneous computing applications become increasingly diverse, written in different parallelization models, running on various platforms and accommodating different usage models. The trend of recognition, mining and synthesis [11] has become dominant with the rapid development of big-data and machine learning applications in recent years. For designing domain-specific accelerators, we can use specialized collections of benchmarks to have a detailed study of these areas, while decisions about general-purpose acclelerators, such as GPUs and MICs, can be based on a diverse set of applications.

Support Research Different to the benchmark suite only for benchmarking real machines, a benchmark suite supporting research has more requirements. It often provides not only scoring systems but infrastructure to instrument, manipulate, and simulate the included programs efficiently.

B. Limitations of Existing Benchmark Suites

In this part we will analyze how existing benchmark suites can not meet the above requirements and thus considered unable to evaluate the performance of future accelerators.

Generic Benchmark Suites for Accelerators. *Rodinia* [12] and *Parboil* [13] are popular benchmark suites for GPU. CUDA, OpenCL and OpenMP implementations are provided in them. Although they are widely used, their benchmarks are mainly regular applications with limited control and memory divergency. They are not suitable for studying the up-to-date irregular workloads on accelerators.

Graph Benchmarks for CPUs. *GAPBS* [14] is a CPU based graph processing benchmark suite. It is a collection of high-performance implementations written in OpenMP. Their implementations are representative of graph performance on multicore CPUs. However, no implementations for accelerators are included in GAPBS.

Graph Benchmarks for GPUs. *Pannotia* [15] includes a set of graph applications with OpenCL implementation on GPU. But no specific optimization is applied in the benchmark. We will show that applicaitons without high optimizations may have different behaviors from optimized ones. *GraphBIG* [16] includes both OpenMP implementations for multicore CPUs and CUDA implementations for GPUs. Mainly focused on CPU versions, the GPU implementations are mostly straightforward. *Lonestargpu* [10] assembles a set of irregular CUDA benchmarks with high optimizations, and some of them are graph workloads. Although it is a good candidate for studying irregular applications on GPUs, they do not focus on graph analytics and thus cannot be used for designing domain-specific accelerators.

Graph Processing Frameworks Pregel [17], GraphMat [18], Ligra [19], Graphlab [20], [21] and GraphReduce [22] are parallel graph processing frameworks designed for CPU. Gunrock [23], Medusa [24], CuSha [25]) are frameworks designed for graph processing on GPU. The two frameworks target for high performance and high programmability, and many new techniques are provided. However, they are impractical for architecture research because of the implementation complexity.

III. The GARDENIA Benchmark Suite

One of the GARDENIA suite's targets was to provide a set of workloads that represent those important irregular applications running on massively parallel accelerators of modern datacenters for scientific studies. 10 irregular applications, which were chosen from big-data analysis and

	Programs	Acc-oriented	Irregular Wokloads	Diverse	State-of-the-art	Arch Reserach
GARDENIA	9	√	√	√	√	√
Parboil	11	√	×	√	√	√
Rodinia	9	√	×	√	√	√
Pannotia	8	√	√	√	×	×
GraphBIG	6	√	√	×	×	√
LonestarGPU	7	√	√	×	√	×
GAPBS	6	×	√	×	√	√
PARSEC	12	×	×	√	√	√
Gunrock	-	√	√	-	√	×

TABLE I
COMPARISON BETWEEN GARDENIA AND PREVIOUSLY AVAILABLE BENCHMARK SUITES

machine learning domains, are included for their popularity and representativeness.

GARDENIA meets all the requirements outlined in Section II:

- Each of the applications has been parallelized using OpenMP for multicore CPUs and CUDA for GPUs.
- GARDENIA benchmark suite focuses on emerging irregular workloads that is representive of big-data applications on modern datacenters equipped with massively parallel accelerators.
- Each of the workloads applies state-of-the-art optimization techniques in its area, i.e., the benchmarks are reasonably optimized to get substantial speedups on GPUs and MICs.
- The workloads as well as their datasets are diverse and were chosen from many different areas. The workloads are representative programs found in domains of big-data analysis, machine learning and sparse linear algebra. The datasets covers different characteristics such as size, density and topology.
- GARDENIA efficiently supports accelerator architecture research. GARDENIA is not a framework, so common abstractions are not forced onto all implementations as libraries do, but freed to do whatever is appropriate for a given program. Thus it facilitates code instrumentation and manipulation, as well as detailed architectural simulations.

A. Optimization Techniques

We present the optimization techniques for irregular applications that have been widely applied in academic and industry libraries [26], [23], [5], [27], [28]. Note that the benchmarks should not either be over-optimized or unoptimized (straightforward) for the accelerator architecture. Those techniques that are bound to specific architectures are not included in our suite. Meanwhile, although irregular algorithms are difficult to efficiently parallelize on data parallel accelerators due to their irregularity [10], recent works [29], [30], [31], [32], [33], [34], [35] have demonstrated that GPUs are capable to substantially accelerate graph algorithms if the algorithms are carefully designed and optimized. Those straightforward implementations that are not able to achieve reasonable speedup on accelerators should not be included since they can not mimic the behavior of real-world applications. Section V will show that straightforward implementations found in previous available benchmark suites behave differently from our optimized workloads. To avoid over-optimization and include state-of-the-art techniques at the same time, our strategy is to choose the common optimization techniques that have been generalized in libraries.

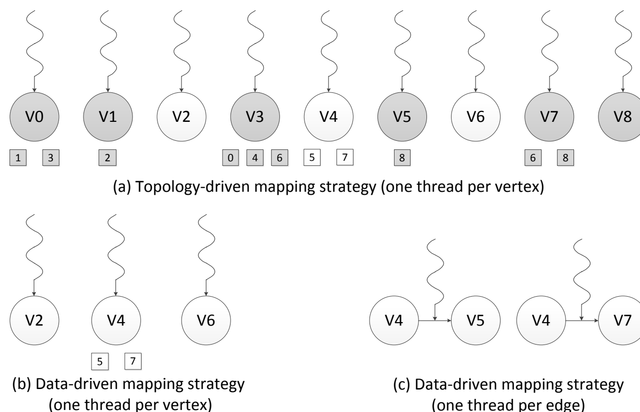

Fig. 1. Topology-driven vs. data-driven mapping strategies

Mapping Strategies. *topology-driven* or *data-driven* mappings [36] are utilized as two basic parallelism strategies. Take graph analytics as an example, in the naive topology-driven implementation, one thread is mapped to one vertex. During an iteration, the thread remains idle or is assigned to process the corresponding vertex. This strategy is a naive way to implement the topology-driven algorithm on the accelerator. In contrast to the topology method, the data-driven implementation uses an extra data structure named frontier queue to hold the unprocessed vertices. Threads are created in proportion to the size of the frontier for each iteration. Each thread is assigned to process some vertices in the frontier, without idles among the threads. Therefore, the data-driven implementation utilizes the threads more efficiently than the topology-driven implementation, with the shortcoming of maintaining a frontier. However, as threads have various number of edges to process, load imbalance problem still exists. The two strategies are shown in Fig.1.

Load Balancing. Irregular algorithms have load imbalance

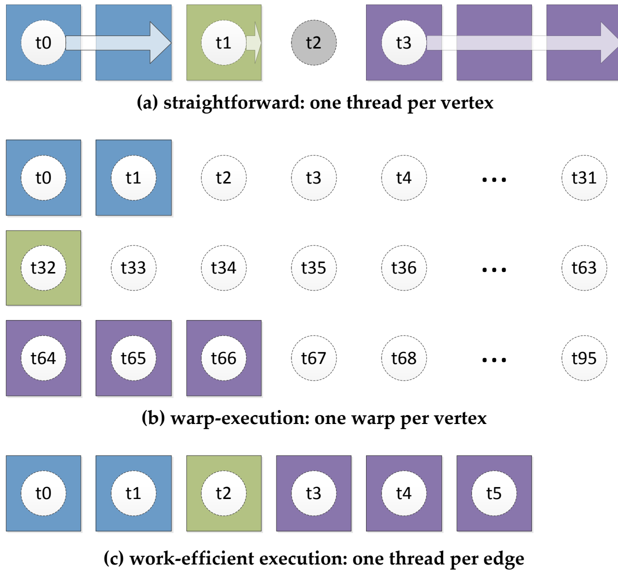

Fig. 2. Load balancing strategies

problem when mapped to accelerators, this problem becomes worse for scale-free graph datasets. A *warp-execution* instead of *thread-execution* method was propose by Hong *et al.* [37] to optimize BFS through mapping warp to vertices (shown in Fig.2). Merrill *et al.* [29] made a step further by proposing a *hierarchical* load balancing strategy. In this method, a vertex is mapped to a thread, a warp, or a thread block, depending on the scale of its neighbor list. For MIC benchmarks, we use OpenMP dynamic schedule scheme to automatically improve load balance.

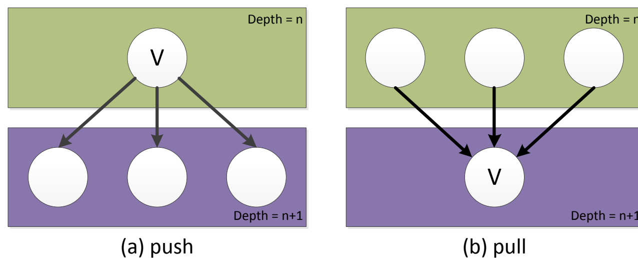

Fig. 3. Push vs. pull traversal

Push vs. Pull. Generally graph algorithms support two types of vertex operations on an implicit frontier: `push` and `pull`. The push-style operation means the active status is pushed by the current frontier of active vertices to its outgoing neighbors so as to create the new frontier. `push` is intuitive and commonly used. However, compared to the push-style method, which starts with a frontier of active vertices, the pull-style advance starts with a frontier of unvisited vertices (shown in Fig.3). So the new frontier is generated by filtering the unvisited frontier for vertices that have incoming neighbors in the current frontier. Beamer *et al.* [38] proposed direction-optimizing BFS which uses both methods to drive BFS. `pull` is beneficial when the number of unvisited vertices drops below the size of the current frontier.

Reordering Queue. The load balance and performance are largely decided by the computation order of the vertices in the frontier queue. Many graph primitives benefit from prioritizing the computation of certain vertices, as some work can be saved by computing those vertices first. For example, we use the delta-stepping implementation for SSSP [39] on MIC. This approach allows user-defined priority functions to organize an output frontier into bins or buckets. Similarly, sorting the frontier queue by degree can improve load balance in some cases.

Other Techniques. Apart from the above techniques, the suite also applies architectural optimization techniques to make full use of the hardware. For example, the read-only data is stored in the texture cache which is a kind of fast on-chip memory. By this way, the read-only data is put in the texture cache initially and the data-access latency is saved, since the long access to the DRAM is not needed.

B. Input Datasets

Input dependency is a major feature of irregular applications. Therefore, graph workloads are characterized both by the algorithms and the structure of the graphs that are used. Real-world sparse graphs are picked from the University of Florida Sparse Matrix Collection [40], the SNAP database [41] and the Koblenz Network Collection [42]. The number of vertices and edges for the graphs, as well as the algorithms that have used a certain graph are shown in Table II. We choose these graphs because they are different in size, degree distribution, density of local subgraphs and so on.

We consider several characteristics of graph datasets: 1) the graph size in terms of number of vertices and edges. 2) the graph density/sparsity in terms of average degree. The degree of a vertex in a graph is the number of connections it has to other vertices. 3) the graph topology in terms of degree distribution and diameter.

Graph Size Since the data size in modern datacenters increases rapidly, common researchers find it hard to use low-end servers to store and manipulate the large volume of data. We select datasets large enough for those researches on big data analysis with limited hardware budget. In addition, to support architecture research, we need small but representative datasets to ensure the programs finish in controllable time.

Graph Density The graph density is usually represented by the average degree of all vertices in the graph. This parameter is important because it is directly related to the amount of locality in the graph workloads. A graph with higher density will probably enjoy more data locality since more spatial locality can be gained between adjacent vertices. However, graphs with higher density have more edges, resulting in larger working set. When the size of the working set is larger than the cache capacity, cache thrashing problem becomes a major issue [43]. Note that real-world graphs are usually sparse, making graph analytics irregular.

Graph Topology There are two categories of graphs in terms of graph topology, mesh and social network. Commonly used graphs can be divided into two categories in terms of graph topology: meshes and social networks. Mesh topology is originated from physically spatial sources and social network topology comes from non-spatial sources. Meshes are structured and they are free from the problems caused by small-world and scale-free properties. However, social networks have a low diameter (small-world) and a power-law degree distribution (scale-free), making them hard to partition and to load balance.

We selected real-world graph instances used in our evaluation to be topologically diverse. For example, `twitter`, `soc-LiveJ` both have the "social network" topology and `road` has a "mesh" topology.

C. Workloads

The following workloads are part of the GARDENIA suite:

Breadth-First Search (BFS) is a fundamental graph primitive. BFS on GPUs incorporates Merrill's *hierarchical* load balancing technique and implement both topology-driven and data-driven mapping strategies.

Single-Source Shortest Paths (SSSP) [30] computes the distance from a given source vertex to all reachable vertices. SSSP on GPUs employs similar load balancing strategies as BFS does.

Betweenness Centrality (BC) can compute the influence of a vertex on a graph [31]. The betweenness centrality score of a vertex is computed as the proportion of shortest paths between all vertices that pass through the vertex. BC on GPUs employs similar load balancing strategies as BFS does.

PageRank (PR) is an iterative algorithm which uses the rank of the linked sites to rank a website [44]; at each iteration, it uses the weighted sum of its neighbors' scores and degrees to update the score of each vertex. PageRank is implemented in both `pull` and `push` fashion, while the `pull` version achieves better performance as we evaluated.

Connected Components (CC) attaches the same label to all vertices in the same connected component [45]. These connected components are of the weak variety, which is in contrast to strongly connected components. `warp-execution` is applied for CC on GPUs.

Triangle Counting (TC) counts the number of triangles in an undirected graph. It is key to graph statistics such as clustering coefficients [45]. To find triangles, it intersects each vertex's neighbor list with its neighbor's neighbor lists. TC on GPUs also employs the `warp-execution` technique.

Vertex Coloring (VC) assigns colors to vertices so that no two neighboring (connected) vertices are assigned the same color. We include recently proposed GPU implementations [46] which yield better coloring quality and performance than the CUSPARSE library [5].

Sparse Matrix-Vector Multiplication (SpMV) is perhaps the most important sparse linear algebra primitive. Our GPU code is from the CUSP library [27] and employs `warp/vector-execution` for memory coalescing. The non-zero elements in the matrix of the workload are used to multiply with the corresponding elements in the vector.

Symmetric Gauss-Seidel smoother (SymGS) is an important program in the multigrid sparse solver from HPCG. Similar to SpMV, SymGS also finds out non-zero elements in the matrix. The elements are multiplied with the corresponding elements in the vector. `warp/vector-execution` is also applied on GPUs.

Stochastic Gradient Descent (SGD) is a key method to solve the matrix factorization problem [47] in collaborative filters of recommender systems. It decomposes the ratings matrix into two smaller matrices, a $(user \times features)$ matrix, and a $(features \times item)$ matrix, and learn these iteratively.We implement `warp/vector-execution` for memory coalescing and `shuffle` operations on GPUs [48].

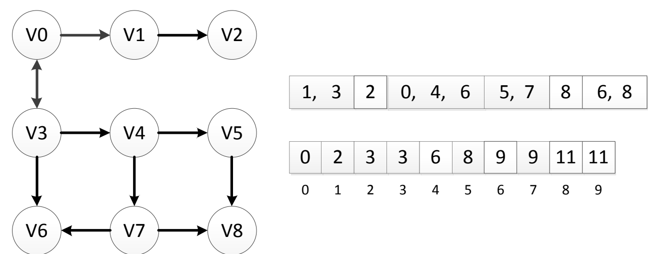

Fig. 4. An example of the compressed sparse row (CSR) format.

IV. Methodology and Experiment Setup

We use the University of Florida Sparse Matrix Collection [40], the SNAP database [41] and the Koblenz Network Collection [42], which are real-world sparse graphs. The matrices with the respective number of vertices and edges are shown in Table II and we use the compressed sparse row (CSR) format to store the sparse matrices (shown in Fig.4). In summary, we use 19 real-world graphs for our evaluation. The graphs are much different in size, degree distribution, density of local subgraphs and application domain.

We compare 2 platforms including (1) `Intel multicore CPU`: OpenMP implementation, (2) `Nvidia GPU`: CUDA implementation, We conduct the experiments on the NVIDIA K80 GPU with CUDA Toolkit 8.0 release. `OpenMP` is executed on Intel Xeon E5 2620 2.40 GHz CPU with 12 cores. We launch 12 threads for `OpenMP` since this is the best performing configuration as we evaluated. We bind the 12 threads to the 12 cores in one CPU for the stability of performance. We use `gcc` and `nvcc` with the `-O3` optimization option for compilation along with `-arch=sm_37` when compiling for the GPU. Hardware information is acquired by *nvprof* on real machines.

The benchmarks are executed 10 times and the average execution time is collected to avoid system noise. Only the computation time of each benchmark is used for timing.

The experiment is designed as 4 parts, including parallelization, cache localtiy, off-chip traffic and irregularity. The 4 parts are independent and irregularity may affect cache

locality and off-chip traffic. After evaluating on the 4 parts, we analyse performance bottlenecks within the 4 aspects and give architectural insights for building high performance and energy-efficient domain-specific accelerators for graph applications.

V. Evaluation

In this section, we will give a comprehensive evaluation on our benchmark suite. We will focus on the aspects of parallelization, cache locality, off-chip traffic and irregularity of the benchmark suite, and give an analysis on its performance bottlenecks.

A. Parallelization

Parallelization has significant effects on GPU performance. More parallelism means more hardware resources are utilized and the parallel execution model is met, leading to high performance. In this section, we will show and analyse the parallelism for our workloads with different datasets.

Figure 5 shows the speedups of our workloads on GPU over multicore CPU with different datasets. The baseline CPU programs are highly optimized openMP programs which run efficiently on multicore CPU. Quite diversed acceleration effects are acquired by CUDA programs on GPU over openMP programs, ranging from 0.36 (BFS) to 7.87 (SpMV) times. There are two main reasons for the low speedups. The first reason is the openMP programs run on muticore CPU are highly efficient. For example, the openMP program version for BFS is a highly optimized program developed by Beamer [38]. The other reason is that the much irregularity and complexity in programs (such as BFS and BC) can't meet with GPU's parallel computing model. For programs with high speedups, the speedups are mainly from careful optimization of memory accesses and high utilization of hardware resources (including compute units and memory bandwidth).

In general, SpMV and SGD have better execution effects across its whole datasets on GPU over CPU, while BFS and BC have worse performance on GPU. For different datasets in the same workload, speedups vary greatly, ranging from 0.48 to 2.76 (for CC). That indicates different datasets have diverse behaviors and have significant effects on workloads' performance. So we can get the Gardenia benchmark suite is very irregular and input sensitive. As we will see in the following sections, different datasets have more important effects on speedups than on irregularity and other metrics.

The main reason that causes low parallelism and inefficient GPU computing is under-utilization of SIMT lanes. Figure 6 shows the utilization of GPU SIMT lanes which is acquired by the hardware counter. The counter records the average active threads per warp to the maximum number number of threads per warp supported on a multiprocessor, which shows the utilization of SIMT lanes. The results show the SIMT lane utilization varies greatly in different workloads with different datasets, with a range from 11% to 100%, because different workloads with different datasets show different data structures and different parallelism. In average, BC, PR, SpMV, SymGS and SGD have relatively higer SIMT lane utilization, and TC has lower utilization (resulting from inefficient data reduction). For SGD and PR, different datasets have similar results, implying they are both insensitive to datasets. But for VC, the highest value is about 3 times of the lowest one, implying it is sensitive to datasets.

We make analysis on the relationship between SIMT utilization and speedup, and find SIMT utilization has relationship with speedup. SGD and SpMV both have high SIMT utilizations, and their speedups are high. BFS and TC have relatively low SIMT utilization, and their speedups are low. However, the lowest utilization will not always cause the lowest speedup (such as TC), because it can improve performance through optimizations in other aspects, leading to a medium speedup. BFS and SSSP have nearly the same utilization, but their speedups vary. This is because other aspects take effects.

B. Cache Locality

In Kepler GPU, L2 data cache is the last level cache (LLC). Memory access that fails in LLC will induce long latency access to the off-chip memory, which will degrade GPU performance. So data locality performance in L2 cache has great importance on GPU performance. In figure 7, workloads show diverse behaviors in L2 hit rate. For TC and SGD, they both have high average L2 cache hit rate, which is as high as 98% and an average of about 80%. Large graphs tend to perform badly in the figure, because the limited L2 cache capacity can not reserve so much data. Particularly, we find the dataset of Wikipedia gets the worst performance in 5 workloads, that are in BFS, SSSP, BC, PR and CC. Its worst hit rate is only 19% in PR. This is because Wikipedia's graph characteristic is not matched with these workloads and LLC fails to capture data locality. So there is room for optimization, which we leave for the future work. Moreover, for a single workload, hit rates vary clearly across different datasets, ranging from 72% to 19% (in PR). This shows datasets also have great impact on workload performance. However, there are three exceptions, namely BC, TC and VC. Especially for VC, its hit rate ranges from 68% to 63% across 5 datasets, showing a fairly flat distribution of hit rate.

We also find GPU speedups have tight relationship with L2 cache hit rate. SGD has higher L2 cache hit rate, and its speedup is high. Although TC has bad SIMT utilization, it achieves medium speedup by the high hit rate in L2 cache. SymGS have low L2 cache hit rate and its speedup is also low. This is because L2 cache is the last level cache on-chip and its performance has great effects on GPU performance.

C. Off-chip Traffic

For the bandwidth restricted programs, performance will increase when off-chip bandwidth increases. So optimization on off-chip accesses will have significant benefits. For the programs not restricted by bandwidth, increased off-chip bandwidth takes no effects.

Figure 8 shows the read and write bandwidth of GPU kernels, and figure 9 shows their corresponding ipc performance.

Graph	# Vertices	# Edges	Algorithms	Brief Description
Road-central	14,081,816	16,933,413	BFS,SSSP,BC,PR,CC,	Road Central
Holly	1,139,905	57,515,616	BFS,SSSP,BC,PR,CC	Hollywood
Twitter	21,297,772	265,025,809	BFS,SSSP,BC,PR,CC	Connection of Twitter users
Amazon	2,146,057 1,230,915	5,838,041	SGD	Amazon ratings network dataset
Thermal	1,228,045	4,904,179	SymGS,VC	Thermal Problem
Nlp	8,345,600	118,931,856	SymGS,VC	Symmetric indefinite KKT matrices
Ldoor	952,203	23,737,339	SymGS	Indeed test matrix
Rgg	1,048,576	6,891,620	TC	Random Geometric Graph
Jester	64,000 150	1,761,439	SGD	Jester Ratings
Netflix	480,189 users 17,770 movies	99,072,112 ratings	SGD	Netflix Prize
Ratings	138,493 27,278	20,000,263	SGD	Ratings
Road-CA	1,965,206	2,766,607	VC	Road network of California
Delaunay	1,048,576	3,145,686	TC,VC	Delaunay Graph
Web-wiki	1,864,433	4,507,315	TC,VC	Web of wikipedia
Af-shell	504,855	9,046,865	SymGS	Sheet metal forming
Cage15	5,154,859	99,199,551	SymGS	DNA electrophoresis
LiveJ	4847571	68993773	BFS,SSSP,BC,PR,CC,TC	LiveJournal online social network
Pokec	1,632,803	22,301,964	TC	Pokec online social network
Wikipedia	3,566,907	45,030,389	BFS,SSSP,BC,PR,CC	Links in Wikipedia pages

TABLE II
SUITE OF BENCHMARK GRAPHS

Fig. 5. Speedup of GPU over multicore CPU

Fig. 6. SIMT lane utilization of different workoads

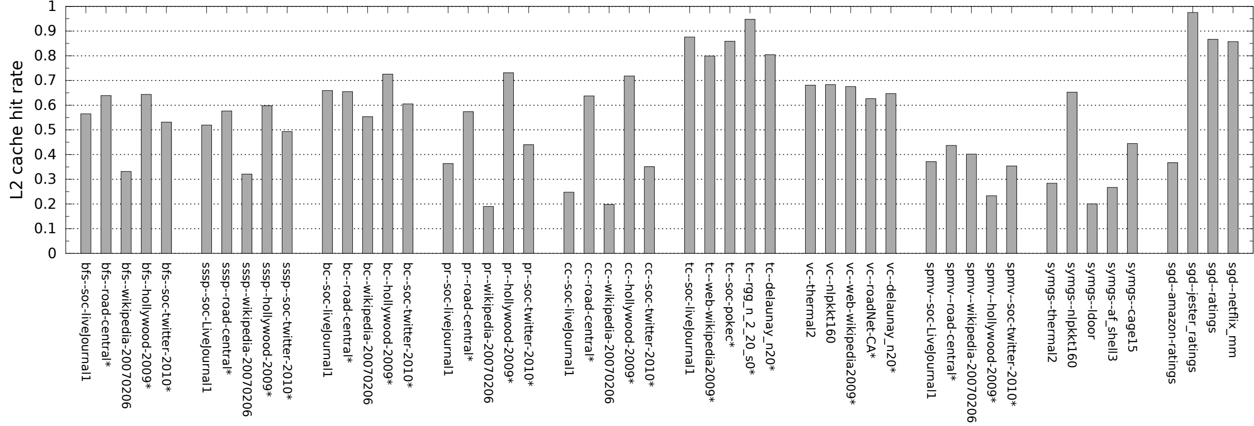

Fig. 7. L2 cache hit rate of different workloads across different datasets

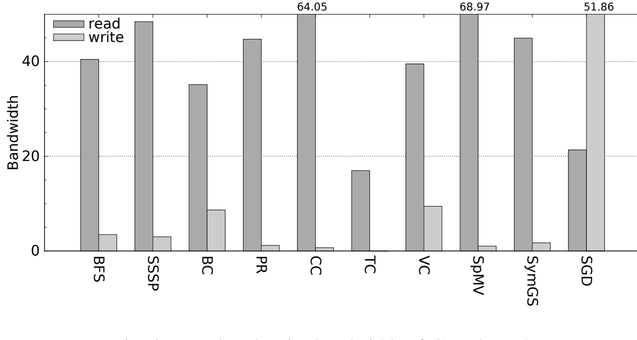

Fig. 8. Read and write bandwidth of GPU kernels

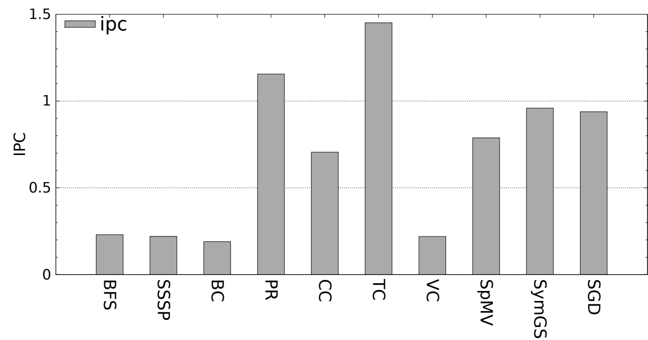

Fig. 9. Ipc performance of GPU kernels

The values of each workload in the two figures are acquired by using geometric means across various datasets. We can see read bandwidth is much higher than write bandwidth in most workloads. This is because there are many read operations but few memory write operations in our algorithms. Memory bandwidth (take read bandwidth as memory bandwidth) has no clear relationship with the ipc performance. For example, CC and SpMV have high bandwidth and their ipc are not high. TC has the lowest bandwidth but its ipc is the highest.

In our experiment, GPU can get an peak memory bandwidth of 170GB/s, which is obtained with bandwidth test in CUDA SDK. We can see the bandwidth values of our workloads are quite lower than the peak bandwidth value, indicating off-chip traffic is not constrained. It is hard for the irregular workloads to utilize the off-chip bandwidth and many of the compute operations are finished on-chip. We will further analyse the problem of off-chip traffic in the following sections.

D. Irregularity

Graph algorithms are a kind of highly irregular programs. Irregularity problem causes non-coalesced memory accesses and under-utilization of bandwidth. In this section, we will make evaluation on irregularity problem in GPU.

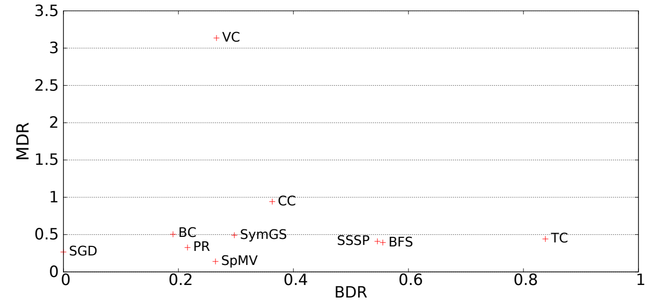

Fig. 10. BDR and MDR of different workloads

As we know, irregularity has bad effects on SIMT execution efficiency and causes replay overhead. We use BDR (Branch Divergence Rate) and MDR (Memory Divergence Rate) to evaluate branch divergence and memory replays, respectively. The higher BDR and MDR values indicate more irregularity, thus computing resources and memory bandwidth are under-utilized. Figure 10 shows the irregularity distribution of different workloads. Each point in the figure represents a workload in Gardenia suite. It shows a scattered distribution in whole.

But most points distribute less than 1.0 in MDR and 0.6 in BDR, except for VC and TC. So most workloads are neither branch-bound nor memory-bound. TC has moderate MDR value, but its BDR value is as high as 0.84. This means TC has many branch divergences and its warp execution efficiency is low (due to its reduction operations). VC's BDR value is low, but its MDR value is high as 3.2. This indicates the memory system works inefficiently for VC and lots of issued memory access are replayed.

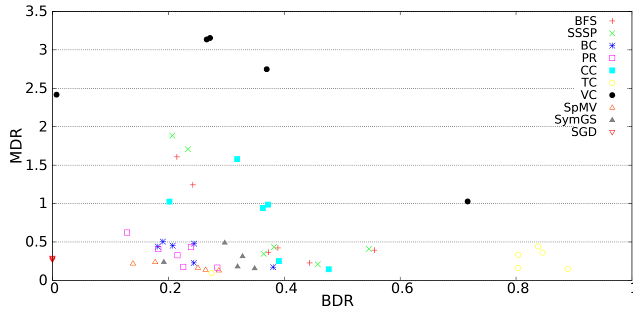

Fig. 11. BDR and MDR of different workloads with different datasets

To give a more comprehensive illustration the irregularity of our benchmarks, we illustrate BDR and MDR of different workloads over different datasets in figure 11. Similar to figure 10, most points in figure 11 are located in the region of less than 0.5 in BDR and less than 1.0 in MDR. This proves most workloads of Gardenia benchmark suite have low irregularity, which is partly due to state-of-the-art optimization techniques. Furthermore, we find most datasets from the same workload distribute locally. This shows irregularity is mainly determined by the workloads and it is insensitive to the input datasets.

E. Performance Bottleneck Analysis

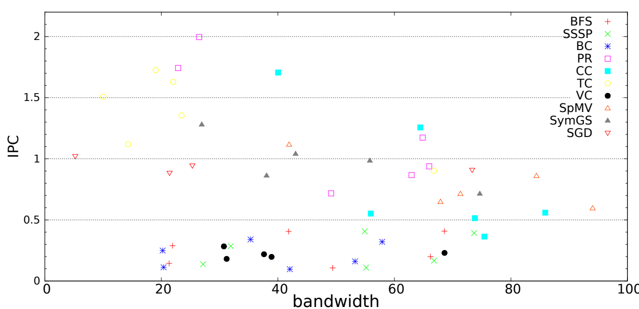

Fig. 12. Ipc and bandwidth of different workloads across different datasets

In this section, we will make performance bottleneck analysis on Gardenia benchmark suite, in the aspects of parallelization, bandwidth, cache peformance and irregularity.

1) parallelization: More compute units will have positive effects on parallelism, but it may not be energy efficient. Because SIMT utilization of Gardenia benchmark is in an average of about 60%, more compute units will bring more power consumption but have limited effects on graph workloads' performance, which have different degrees of parallelism by nature. So the next-generation accelerator should be able to balance computating power and energy efficiency, with a moderate number of compute units.

2) bandwidth: Scientific computing has much parallelism and long memory latencies can be well hidden. Because of its large number of off-chip memory access, scientific applications are mainly memory bandwidth bound. To reveal the effective bandwidth effects on performance in our workloads, we measure ipc and the effective bandwidth of the workloads with different datasets, which is shown in figure 12.

For the highly optimized programs in Gardenia benchmark suite, such as vector-centric or warp-centric optimized programs, they achieve high memory bandwidth and high ipc at the same time. However, the points in figure 12 are still not close to peak value of the bandwidth, implying bandwidth is not fully utilize. Points from the same workload distribute dispersedly along x-axis and y-axis, which means the ipc and bandwidth are sensitive to the datasets. Most dots locate in the left higher and middle lower region. Less effective bandwidth indicates more compute operations, leading to higher ipc. However, For BFS, SSSP and VC, their points distribute in the left lower region, that is because the programs fail to utilize compute units and memory bandwidth, presumably due to memory latency [14]. Our workloads don't have enough in flight memory accesses to hide memory latency, which becomes a bottleneck in performance. So Gardenia suite is not bandwidth limited and is dataset sensitive. The attention of the accelerator designers should be focused on latency, not on off-chip bandwidth, since no potential performance gains can be expected through higher bandwidth.

3) cache performance: Cache performance is effective to reduce memory latency. As analysed in section V-B, L2 cache hit rate has significant effects on GPU performance. Most workloads' L2 cache hit rates are below 50% and have room for optimization. Accelerator designers should design better cache architecture dedicated for graph algorithms and graph datasets to find an optimum point balancing compute and memory access, and reducing memory latency. Optimizing cache performance can not only improve cache hit rate, but also reduce MDR values, indirectly reducing irregularity.

4) irregularity: Irregular issues are mainly determined by algorithms themselves, and parallelism and cache performance also have effects on irregularity in architectural aspect. So we should find effect ways to relieve the previous bottlenecks to solve the irregularity issues.

VI. Conclusion

In this paper, we use GARDENIA, a domain-specific benchmark suite for accelerators, to evaluate architectural characteristics of a commonly used Nvidia's GPU. We focus on GPU's parallelization, cache locality, off-chip traffic and irregularity. A detailed analysis on its performance bottlenecks is given and useful advices are given to coprocessor designers. We conclude, (1) it is not the case that the more compute units

the accelerator has, the better performance it can get. We have to take both parallelization and energy efficiency into consideration; (2) the attention of the accelerator designers should be focused on latency, not on off-chip bandwidth, since no potential performance gains can be expected through higher bandwidth; (3) accelerator designers should design better cache architecture dedicated for graph algorithms and graph datasets to find an optimum point balancing compute and memory access, and reducing memory latency.

REFERENCES

[1] R. Hameed *et al.*, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*. New York, NY, USA: ACM, 2010, pp. 37–47.

[2] W. Qadeer *et al.*, "Convolution engine: Balancing efficiency & flexibility in specialized computing," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*. New York, NY, USA: ACM, 2013, pp. 24–35.

[3] NVIDIA, "nvGRAPH Library," 2016. [Online]. Available: http://docs.nvidia.com/cuda/nvgraph/index.html

[4] ——, "cuDNN Library," 2016. [Online]. Available: https://developer.nvidia.com/cudnn/

[5] ——, "CUSPARSE Library," 2016. [Online]. Available: http://docs.nvidia.com/cuda/cusparse/

[6] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2015, pp. 105–117.

[7] M. M. Ozdal *et al.*, "Energy efficient architecture for graph analytics accelerators," in *Proceedings of the 43rd International Symposium on Computer Architecture*. Piscataway, NJ, USA: IEEE Press, 2016, pp. 166–177.

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

[9] L. Nai *et al.*, "Graphpim: Enabling instruction-level pim offloading in graph computing frameworks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 457–468.

[10] M. Burtscher *et al.*, "A quantitative study of irregular programs on gpus," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, Nov 2012, pp. 141–151.

[11] C. Bienia *et al.*, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. New York, NY, USA: ACM, 2008, pp. 72–81.

[12] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the IEEE International Symposium on Workload Characterization*, 2009, pp. 44–54.

[13] J. A. Stratton *et al.*, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," UIUC, Urbana, Tech. Rep. IMPACT-12-01, Mar. 2012. [Online]. Available: http://impact.crhc.illinois.edu/Shared/Docs/impact-12-01.parboil.pdf

[14] S. Beamer *et al.*, "Locality exists in graph processing: Workload characterization on an ivy bridge server," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2015, pp. 56–65.

[15] S. Che *et al.*, "Pannotia: Understanding irregular gpgpu graph applications," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, Sept 2013, pp. 185–195.

[16] L. Nai *et al.*, "Graphbig: Understanding graph computing in the context of industrial solutions," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: ACM, 2015, pp. 69:1–69:12.

[17] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2010, pp. 135–146.

[18] N. Sundaram *et al.*, "Graphmat: High performance graph analytics made productive," *Proc. VLDB Endow.*, vol. 8, no. 11, pp. 1214–1225, Jul. 2015.

[19] J. Shun and G. E. Blelloch, "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)*. New York, NY, USA: ACM, 2013, pp. 135–146.

[20] Y. Low *et al.*, "Graphlab: A new parallel framework for machine learning," in *Proceedings of the UAI*, 2010, pp. 340–349.

[21] ——, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, Apr. 2012.

[22] D. Sengupta *et al.*, "Graphreduce: Processing large-scale graphs on accelerator-based systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: ACM, 2015, pp. 28:1–28:12.

[23] Y. Wang *et al.*, "Gunrock: A high-performance graph processing library on the GPU," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Mar. 2016.

[24] J. Zhong and B. He, "Medusa: Simplified graph processing on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1543–1552, June 2014.

[25] F. Khorasani *et al.*, "Cusha: Vertex-centric graph processing on gpus," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing (HPDC)*. New York, NY, USA: ACM, 2014, pp. 239–252.

[26] D. Merrill, "CUB," NVIDIA Research, 2015. [Online]. Available: http://nvlabs.github.io/cub/

[27] S. Dalton *et al.*, "Cusp: Generic parallel algorithms for sparse matrix and graph computations," 2014, version 0.5.0. [Online]. Available: http://cusplibrary.github.io/

[28] S. Baxter, "moderngpu 2.0," 2016, version 2.0. [Online]. Available: https://github.com/moderngpu/moderngpu/

[29] D. Merrill *et al.*, "Scalable gpu graph traversal," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. New York, NY, USA: ACM, 2012, pp. 117–128.

[30] A. Davidson *et al.*, "Work-efficient parallel gpu methods for single-source shortest paths," in *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS)*, May 2014, pp. 349–359.

[31] A. McLaughlin and D. A. Bader, "Scalable and high performance betweenness centrality on the gpu," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 572–583.

[32] G. M. Slota *et al.*, "High-performance graph analytics on manycore processors," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2015, pp. 17–27.

[33] R. Nasre *et al.*, "Morph algorithms on gpus," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. New York, NY, USA: ACM, 2013, pp. 147–156.

[34] S. Nobari *et al.*, "Scalable parallel minimum spanning forest computation," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. New York, NY, USA: ACM, 2012, pp. 205–214.

[35] G. M. Slota *et al.*, "Parallel graph coloring for manycore architectures," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 1–10.

[36] R. Nasre *et al.*, "Data-driven versus topology-driven irregular computations on gpus," in *Proceedings of the 27th IEEE International Parallel Distributed Processing Symposium (IPDPS)*, May 2013, pp. 463–474.

[37] S. Hong *et al.*, "Accelerating cuda graph algorithms at maximum warp," in *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP)*. New York, NY, USA: ACM, 2011, pp. 267–276.

[38] S. Beamer *et al.*, "Direction-optimizing breadth-first search," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 12:1–12:10.

[39] U. Meyer and P. Sanders, "Δ-stepping: A parallelizable shortest path algorithm," *J. Algorithms*, vol. 49, no. 1, pp. 114–152, Oct. 2003.

[40] "The University of Florida Sparse Matrix Collection," 2011. [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices/
[41] J. Leskovec, "Snap: Stanford network analysis platform," 2013. [Online]. Available: http://snap.stanford.edu/data/index.html
[42] "Koblenz network collection," 2013. [Online]. Available: http://konect.uni-koblenz.de
[43] X. Chen *et al.*, "Adaptive cache management for energy-efficient gpu computing," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Washington, DC, USA: IEEE Computer Society, 2014, pp. 343–355.
[44] L. Page *et al.*, "The pagerank citation ranking: Bringing order to the web," 1998.
[45] S. Beamer *et al.*, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, 2015.
[46] P. Li *et al.*, "High performance parallel graph coloring on gpgpus," in *Proceedings of the 30th IPDPS Workshop*, 2016, pp. 1–10.
[47] Y. Koren *et al.*, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009.
[48] X. Xie *et al.*, "CuMFSGD: Parallelized stochastic gradient descent for matrix factorization on gpus," in *International Symposium on High-Performance Parallel and Distributed Computing*, 2017.