Specialization or Generalization: A Study on Breadth-First Graph Traversal on GPUs

Wenyong Zhong, Yanxin Cao, Jiawen Li, Jianhua Sun, and Hao Chen
College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
Email: {hnuzwy,rachelc17,muzi,jhsun,haochen}@hnu.edu.cn

***Abstract*—GPUs (Graphics processing units) have been increasingly adopted for large-scale graph processing by exploiting the inherent parallelism. There have been many efforts in designing specialized graph analytics and generalized frameworks. The two classes of graph processing systems share some common design choices, and often make specific trade-offs. However, there is no characterization study that provides an in-depth understanding of both approaches. In this paper, we analyze two GPU-based graph processing systems (Enterprise and Gunrock) from the perspective of breadth-first graph traversal. We conduct both high-level performance comparison and low-level characteristic evaluation such as workload balancing, synchronization, and memory subsystem. We investigate the differences based on 10 real-world and synthetic graphs. Our results reveal some uncommon findings that would be beneficial to the research and development of large-scale graph processing on GPUs.**

***Index Terms*—GPU, Graph Processing, Breadth-First Traversal, BFS**

I. Introduction

With the rapid development of the Internet, processing large-scale graph structures with millions or billions of vertices and edges has become a hot research topic in both the academia and industry. Graph analytics are becoming increasingly important for many applications spanning areas from scientific computing, advertising, biological networks to social networks. For example, finding the shortest paths of on-line maps, the citation relationships among twitter forwarding, and the purchasing preference in E-commerce webs, are all typical scenarios that heavily rely on efficient graph computation.

As a result, in the last several years, we have witnessed a growing interest in distributed graph processing, such as Pregel, GraphLab, PowerGraph, GPS, and Mizan, which are purposely-built distributed graph computing systems with easy-to-use programming interfaces and reasonable performance under large-scale workloads. The prominent one is Pregel [7], which was firstly proposed as a programming model to address the challenges in parallel computing of large graphs. The high-level programming model of Pregel plays a significant role in abstracting architectural details of parallel computing from programmers. Specifically, the vertex-centric programming model proposed by Pregel greatly relieves the efforts of performing computation on large-scale data-intensive graphs, and provides high expressibility for a wide range of graph algorithms. Inspired by Pregel, a number of graph computing frameworks focusing on heterogeneous accelerators has emerged, such as Medusa [14] and TOTEM [5].

With the advancement of GPU hardware and the introduction of GPU programming frameworks such as CUDA and OpenCL, GPU has become a more generalized computing device. General purpose computing on GPU (GPGPU) has found its way into many fields as diverse as biology, linear algebra, cryptography, image processing, and so on, given the tremendous computational power provided by GPUs such as massive parallel threads and high memory bandwidth as compared to CPUs. In parallel to this trend, GPUs are increasingly leveraged to accelerate graph applications with either low-level hardwired implementations [3], [8], [6] or high-level programmable frameworks to hide the hardware intricacies [14], [15], [2], [11], [10] without sacrificing much performance.

Medusa [14] is a general purpose GPU-based graph processing framework that provides high-level APIs for easy programming and scales to multiple GPUs. CuSha [2] is a graph processing framework that enables users to write vertex-centric algorithms on GPU, proposing a new graph representation called G-Shards to minimize non-coalesced memory accesses. Gunrock [11] is a high-performance graph processing library targeting GPUs, and it implements a data-centric abstraction. At the same time, another line of work focuses on hardwired implementation of specific graph algorithms. A. Davidson et al. [3] present three parallel-friendly and work-efficient approaches to solve the Single-source Shortest Paths (SSSP) problem on GPUs. D. Merrill et al. [8] present a Breath-first search (BFS) parallelization focused on fine-grained task management. Enterprise [6] is the state-of-the-art implementation of BFS with specifically designed optimizations for GPUs.

These endeavors greatly advances the research of GPU-based graph processing. However, little is known about the particularities between the generalization and specialization of graph analytics on GPUs. To what extend, can graph frameworks approach the performance of hardwired implementations? Does specialization always outperform generalization across all types of graphs? How to identify the key metrics that are tightly related to the performance difference, given the common goal of optimizations in both camps. To answer these questions, in this paper we conduct a thorough evaluation on two state-of-the-art implementations from the perspective of generalization and specialization of graph processing; one is *Enterprise* [6], and the other is *Gunrock* [11]. We use Breadth-First Search as a case study because it is a building block for many graph applications including single source shortest path, betweenness centrality, and closeness centrality. The main contributions of

this paper include:

- We conduct an extensive evaluation on two stat-of-the-art GPU graph processing systems, which are representatives for generalized and specialized breath-first traversal systems. The experiments are performed from the perspective of both overall performance comparison and low-level crucial metric characterizations including load balancing, synchronization, and memory related issues, which helps understand the generalization and specialization of GPU graph analytics.
- Our evaluation is performed on a wide set of representative graph datasets that include real world scale-free graphs, road networks, and synthetic graphs. The results not only confirm observations made in existing work, but reveals some findings that will have important implications for future research on GPU-based graph processing.

II. Preliminaries and Background

In this section, we present the necessary background on the graph programming model and GPU architecture.

A. GPGPUs

Current generation of general-purpose GPUs typically has thousands of processing cores. For example, the NVIDIA Pascal GPU GTX 1080 consists of 4 Graphics Processing Clusters (GPCs), 20 Streaming Multiprocessors (SMs). Each GPC has 5 SMs, and each SM is equipped with up to 128 CUDA cores, 256 KB of register file capacity, a 96 KB shared memory unit, and 48 KB of total L1 cache. In addition to the L1 cache, the full GP104 chip used in GTX 1080 ships with a total of 64 ROPs and 2048 KB of L2 cache. with 20 SMs, the GTX 1080 GPU has a total of 2560 CUDA cores and 160 texture units. The off-chip global memory has a much larger size (typically in GB range) and longer access latency.

B. GPU-based Graph Analytics and Gunrock

Pregel [7] is the first programming model for large-scale graph processing that is inspired by the Bulk Synchronous Parallel (BSP) model. In this model, programmers express the parallelism of graph computation by a sequence of iterations called super-steps. The computation model of Pregel is vertex-centric and based on message passing.

Different from the previous GPU graph programming models that focus on sequential computation steps, Gunrock [11] proposes a data-centric parallel programming model for GPUs. The key abstraction of Gunrock's data-centric model is the *frontier*, which represents a subset of edges or vertices of the graph that is currently in active state. Gunrock manipulates the *frontier* with three operators. The *advance* operator produces a new frontier by visiting the neighbors of the current frontier. The *filter* operator generates a new frontier by selecting a subset of the current frontier based on certain policies provided by programmers. A *compute* operator defines an operation on all vertices or edges in the input frontier. A programmer-specified compute operator can be used together with the three operators.

C. Breadth-First Search and Enterprise

BFS algorithm is the building block for many graph applications, and the Graph 500 uses BFS to benchmark high performance hardware and software systems on power-law graphs. The traditional top-down BFS algorithm starts with a source vertex, and manipulates a data structure usually referred to *frontier queue* by adding unvisited neighboring vertices of the source to the queue. The vertices in the frontier queue are marked as visited, and will be used for expansion in the subsequent traversal. Iteratively, the BFS algorithm inspects the neighboring vertices of all the queued nodes and inserts new vertices into the frontier. The frontier produced by the preceding level will be used for expansion in the next level. In this way, the algorithm is executed repeatedly level by level until no vertex remains unvisited.

Enterprise [6] is a GPU-based system designed for breadth-first graph traversal, which leverages the massive threads and high memory bandwidth of GPUs to optimize the execution flow and data access pattern of BFS algorithm. Enterprise achieves high performance using three novel techniques. First, *streamlined GPU threads scheduling* is proposed to not only eliminate thread synchronization in frontier queue generation, but also remove duplications in the frontier queue to avoid useless work during the execution. Second, *GPU workload balancing via frontier classification* is used to mitigate inter-thread workload imbalance by classifying the frontiers based on the out-degrees of vertices. Third, *GPU-aware direction optimization* is used to determine when to switch from top-down to bottom-up BFS based on the ratio of hub vertices in the frontier queue.

III. Evaluation and Analysis

All experiments were conducted on a Linux workstation with 2 3.50 GHz Intel 8-core E5-2630 v3 Xeon CPUs, and a NVIDIA GTX 1070 GPU with 8 GB device memory. The operating system was Ubuntu 16.04 with CUDA 8.0 and GCC v4.9 installed. *Gunrock* and *Enterprise* were compiled with the default setting of their source code. We use NVIDIA's command-line tool *nvprof* to collect hardware metrics and events that are necessary to analyze performance differences. Because our goal is to analyze and understand the differences of generalized and specialized GPU implementations, we only consider on-device performance results and ignore data transfer times.

Table I summarizes the datasets used for benchmarking, which contain different types of graphs. For example, real-world road networks such as roadNet-CA, road_usa, and europe_osm have an average out-degree below 3; social networks (names with prefix 'soc') commonly exhibit scale-free degree distributions and small diameters (thus small depths). One kronecker dataset is included, and it is similar to social networks with a small traversal depth. The dataset also include movie actor network (hollywood_2009), web graphs collected by crawler (uk-2002) and from Google (web-Google). Overall, our datasets cover a wide range of topologies that can help characterize the performance different between *Gunrock* and

TABLE I: Graph datasets used in our evaluations. Avg. Degree indicates the average vertex degree for all vertices, and depth is the average traversal depth randomly sampled.

Dataset	Vertices	Edges	Avg. Degree	Depth
roadNet-CA	1.97M	5.53M	2.81	676
road_usa	23.9M	57.7M	2.41	5430
europe_osm	51.0M	108M	2.12	21462
hollywood_2009	1.14M	113M	98.9	9
soc-LiveJournal1	4.85M	85.7M	17.7	14
soc-orkut	3.00M	213M	71.0	8
soc-twitter2010	21.3M	530M	24.9	16
uk-2002	18.5M	292M	15.8	27
web-Google	0.92M	5.1M	5.54	17
kron_g500-logn21	2.10M	181M	86.8	6

TABLE II: Overall performance comparison, illustrating the throughput in millions of traversed edges per second (MTEPS).

Dataset	Enterprise (Million)	Gunrock (Million)
roadNet-CA	69	56
road_usa	142	63
europe_osm	50	36
hollywood_2009	17272	5912
soc-LiveJournal1	4457	1360
soc-orkut	8162	3333
soc-twitter2010	3125	1587
uk-2002	4011	6568
web-Google	1134	619
kron_g500-logn21	24156	1254

Enterprise. The datasets are from the University of Florida Sparse Matrix Collection [4].

In the following, we first evaluate the overall performance of the two systems, and analyze the impact of graph topologies on the performance of BFS traversal with the commonly used metric *throughput* (measured in millions of traversed edges per second (MTEPS)) and GPU hardware-level metrics to understand the underlying reasons. Except for overall performance, we further investigate three important factors that are crucial design choices when building high-performance GPU-based graph processing. The first is workload distribution that is crucial to balance load among GPU cores and dependent on data layout of graphs. Second, both graph frameworks and hardwired implementations rely on explicit or implicit synchronization to arrange data or distribute work among different steps. The third factor is memory access pattern that is important for efficient utilization of the global and shared memory.

A. Overall Performance

Table II shows the throughput in millions of traversed edges per second (MTEPS) for all datasets. Several observations can be made in Table II. First, specialized BFS engine (Enterprise) is faster than generalized system (Gunrock) across all datasets except for uk-2002. The speedup peaks at 19x for kron_g500-logn21. For dataset uk-2002, Gunrock is 1.6x faster than Enterprise. The speedup ratio between Enterprise and Gunrock is less than 3x for other workloads. Second, the topology of

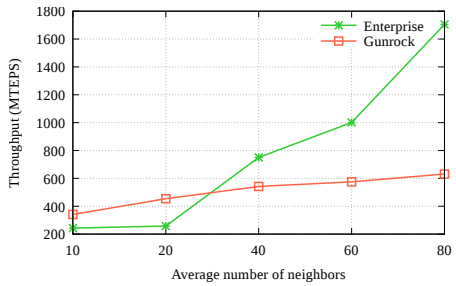

Fig. 1: Throughput comparison of graphs with different average degrees.

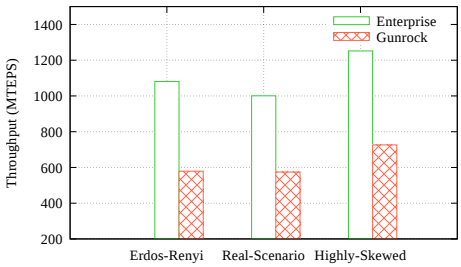

Fig. 2: Throughput on scale-free graphs with the same number of vertices and edges and different skewness.

input graphs has a significant impact on the overall performance. Scale-free graphs achieve much higher traversal rates than road networks by several orders of magnitude for both Enterprise and Gunrock. For example, under the workloads hollywood_2009 and kron_g500-logn21, the throughput of both systems is 345x/164x and 483x/35x better than that of the workload europe_osm respectively. Both Enterprise and Gunrock perform poorly on road networks, and the performance is close to each other. BFS is traversal-based, and there is only a subset of vertices or edges active in the frontier at any given time. So smaller frontiers means less parallelism, and the GPU can not reach it maximum throughput. Third, from Table I and Table II, we can also infer the performance impact of the average degree. Road networks have much smaller average degrees as compared to hollywood_2009 and kron_g500-logn21, which can directly explain the performance gap between these two types of workloads. More work per vertex indicates higher throughput, because this can keep the GPU fully occupied.

Besides the datasets in Table I, we conduct experiments to show the impact of the average degree on BFS performance using a set of synthetic scale-free graphs (with R-MAT [1]) that have the same number of vertices (10 millions) but differs in average neighbors (edges vary from 10 millions to 80 millions). As shown in Figure 1, both systems achieve higher throughput with increasing average degrees, but Enterprise climbs up much faster than Gunrock when the average degree becomes larger. For low average degrees, Gunrock outperforms Enterprise. Thus, combined with the above analysis, we can know that larger average degree is necessary to obtain higher performance.

In addition, the skewness of vertex distribution in scale-

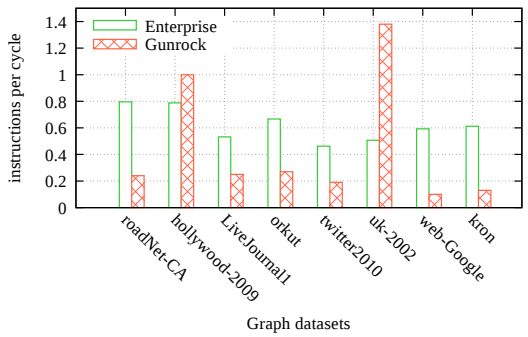

Fig. 3: Comparison of instructions per cycle (IPC) between Enterprise and Gunrock.

free graphs may also have implications for performance. Here, we further investigate the performance variance caused by skewness, by using GTgraph [1] to generate graphs with different behaviors from typical social graphs. We set the ratio of the parameters a, b, and c (choosing b = c for symmetry) to 8 (highly-skewed scale-free graph), 3 (closest to many real-world scenarios), and 1 (Erdos-Renyi model). The synthesized graphs have the same number of vertices and edges (10 millions vertices and 60 millions edges) but different skewness. Figure 2 shows the results. We can see that both Enterprise and Gunrock performs better with the highly-skewed workload because highly-skewed graphs have smaller traversal depths, although for the workloads Erdos-Renyi and real-Scenario, the difference is marginal. Enterprise still outperforms Gunrock under the three workloads.

In the following, we investigate the architectural metrics that characterize the observed differences in overall performance. Collecting low-level metrics needs to use *nvprof* to instrument GPU kernels, which incurs very high runtime overhead, and for road graphs, the applications often ran for more than a week until completion. So, we only present the results of roadNet-CA. As significant differences in IPC (instructions per cycle) could be indicative of correlated performance differences, we choose the metric *executed_ipc* for our analysis. Executed IPC is the average number of instructions executed (guaranteed to retire) per active cycle. An SM is considered active if at least one warp is allocated on the SM. As shown in Figure 3, the IPCs of Enterprise are much higher than that of Gunrock except for two exceptions. One is for dataset uk-2002, under which we obtain better performance with Gunrock as illustrated in Table II. The other exception is for the case of hollywood-2009, where Enterprise outperforms Gunrock but has a lower IPC. This is because IPC is not always directly connected to performance, cycles will also be influenced by a wide range of reasons that cause stalls, which we will discuss next.

Warp can be stalled for many reasons including *instruction fetch* (the next instruction is not yet available), *execution dependency* (the next instruction is waiting for one or more of its inputs to be computed by earlier instructions), *memory dependency* (the next instruction is waiting for a previous memory accesses to complete), *memory throttle* (a large number of outstanding memory requests prevents forward progress), *texture* (the texture sub-system is fully utilized or has too many outstanding requests), *sync* (the warp is waiting for all threads to synchronize after a barrier instruction), *constant memory dependency* (the warp is stalled on a miss in the cache for __constant__ memory), *pipe busy* (the warp is stalled because the functional unit is busy), *not selected* (the warp was ready but did not get a chance to issue as some other warp was selected for issue), and *other* (the warp is blocked for an uncommon reason like compiler or hardware reasons).

Table III, Table IV, Table V, and Table VI show the breakdown of stall reasons. Due to space constraints, we only present the results of two representative graphs hollywood_2009 and roadNet-CA, and other workloads exhibit similar behaviors. The first column represents the names of running kernels (abbreviation), and the percentage numbers in the parentheses are the ratio of kernel runtime to total execution time. We only include kernels that contribute the most to the running time (3 for Enterprise, 2 or 3 for Gunrock).

From the four tables, we can make the following observations. First, the kernels that consume the most of execution time are not the same for different workloads. For example, there is no overlap between the 3 kernels in Table III and Table IV. Table V and Table VI share one kernel 'CULL', but the top time-consuming kernel is not the same. This phenomenon implies that focusing on the kernels that are invoked under one kind of workload may veil the facts that different workloads demand different optimizations. Second, for hoolywood_2009 and other similar graphs in our datasets, the execution time is relatively evenly distributed to different kernels in Enterprise, while only one kernel dominates in Gunrock. This has important implication for performance, because one underlying factor may render the dominant kernel the bottleneck. Road networks that display poor performance in Table II behave similarly (comparing Table IV and Table VI), considering the runtime distribution. Third, stalls such as *texture, pipe, throttle, not_selected* have pretty low values, which means that both systems do not use specific resources such as texture memory on one hand, and do not saturate certain hardware components such as the functional unit (*pipe*) and issue a large number of outstanding memory requests (*throttle*) on the other hand. Forth, memory dependency (column *memory*) is the main contributor to stalls. However, other stalls should also be considered when reasoning the performance differences. For example, the values of memory dependency in Table III are larger than that in Table V, but the overall performance can not reflect this. Instead, other stalls such as *sync* and *inst_fetch* may also be major factors that influence performance. For the same system, stalls are not always identical. For example, the stalls *inst_fetch* and *sync* in Enterprise are more significant for roadNet-CA as compared to hollywood_2009. In summary, we speculate that single dominant kernel combined with multiple non-trivial stalls could explain the performance downside of Gunrock.

TABLE III: Breakdown of stalls for workload hollywood_2009 in Enterprise.

KERNELS (%)	inst_fetch	exec	memory	texture	sync	other	constant	pipe	throttle	not_selected
WAP (22%)	6.52%	7.47%	73.39%	0.01%	3.78%	5.55%	1.92%	0.50%	0.00%	0.85%
THD (15%)	3.22%	10.09%	80.02%	0.21%	1.70%	3.44%	0.63%	0.28%	0.01%	0.41%
SORT1 (14%)	0.27%	56.48%	31.54%	0.15%	0.00%	10.29%	0.05%	0.01%	1.20%	0.01%

TABLE IV: Breakdown of stalls for workload roadNet-CA in Enterprise.

KERNELS (%)	inst_fetch	exec	memory	texture	sync	other	constant	pipe	throttle	not_selected
SORT2 (49%)	2.82%	3.56%	88.74%	0.00%	2.90%	1.40%	0.40%	0.04%	0.00%	0.13%
PRE (14%)	14.69%	22.40%	19.78%	0.00%	29.25%	6.25%	5.40%	0.83%	0.11%	1.28%
POST (10%)	11.62%	14.41%	42.09%	0.00%	19.48%	4.16%	6.82%	0.51%	0.16%	0.75%

TABLE V: Breakdown of stalls for workload hollywood_2009 in Gunrock.

KERNELS (%)	inst_fetch	exec	memory	texture	sync	other	constant	pipe	throttle	not_selected
RELAX (80%)	1.83%	13.44%	66.89%	0.00%	12.77%	3.98%	0.14%	0.37%	0.01%	0.58%
CULL (18%)	23.21%	7.12%	16.98%	0.00%	36.07%	8.76%	7.15%	0.32%	0.02%	0.36%

TABLE VI: Breakdown of stalls for workload roadNet-CA in Gunrock.

KERNELS (%)	inst_fetch	exec	memory	texture	sync	other	constant	pipe	throttle	not_selected
FORWARD (50%)	25.36%	12.07%	40.37%	0.00%	12.65%	4.96%	4.50%	0.10%	0.00%	0.00%
CULL (16%)	33.54%	9.03%	3.51%	0.00%	33.08%	12.43%	7.05%	0.63%	0.05%	0.67%
COUNT (9%)	17.13%	9.42%	49.73%	0.00%	0.00%	1.54%	21.16%	0.32%	0.60%	0.10%

B. Load Balancing

It is very challenging to write efficient GPU programs for graph processing due to the inherent irregularity of graph structures. Due to large variance of vertex degrees, this irregularity would cause imbalanced workload distribution, which severely limits the utilization of GPU resources. Existing generalized or specialized graph engines put a lot of efforts in designing efficient data organization and workload-to-thread mapping strategies to alleviate this problem. For example, Enterprise introduces an approach of classifying frontiers based on out-degrees and assigning GPU threads dynamically at runtime. Gunrock proposed a set of methods to deal with load imbalance, such as static workload mapping strategy, dynamic grouping workload mapping strategy, and merge-based load-balanced partitioning workload mapping strategy.

For graph processing systems, imbalanced workload distribution among threads would result in warp divergence, which should be avoided in order to achieve ideal performance. Warp execution efficiency (WEE) is a low-level metric defining the ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor. It can be used to describe how evenly a system allocate its load to GPU threads. Table VII and Table VIII show the average WEE of Enterprise and Gunrock for all the kernels invoked during execution (the dash line in Table VIII indicates the absence of kernel invocation). As aforementioned, only the results of two datasets are presented due to space limits. In addition, the percentage of time consumption for each kernel is also include in the tables (the Time column).

From the tables, we can know that both systems achieve

TABLE VII: Warp execution efficiency of Enterprise.

KERNELS	hollywood_2009		roadNet-CA	
	Time	WEE	Time	WEE
WAP	22.1%	97.62%	0.1%	100.00%
THD	14.7%	63.70%	0.7%	86.80%
SORT1	13.8%	70.78%	3.0%	93.78%
REAPER	10.3%	92.17%	6.5%	86.32%
SORT2	6.9%	93.65%	49.5%	98.23%
SORT3	9.6%	95.35%	0.6%	93.17%
CTA1	6.5%	99.09%	4.5%	100.00%
PRE	5.2%	84.49%	13.6%	84.49%
WAPE	4.6%	98.04%	5.0%	100.00%
POST	3.2%	88.75%	10.1%	88.75%
CTA2	2.1%	99.98%	0.1%	100.00%
THDE	0.9%	98.52%	6.5%	98.41%

TABLE VIII: Warp execution efficiency of Gunrock.

KERNELS	hollywood_2009		roadNet-CA	
	Time	WEE	Time	WEE
RELAX	79.9%	99.13%	–	–
CULL	17.8%	82.76%	15.9%	83.70%
COUNTS	0.84%	94.67%	9.0%	97.07%
MAD	0.46%	100.00%	1.3%	100.00%
DOWNSWEEP	0.23%	100.00%	5.1%	99.98%
LIGHT	0.23%	94.31%	–	–
REDUCE	0.23%	97.30%	4.2%	97.47%
FORWARD	–	–	50.3%	82.10%
ACCUM	–	–	5.9%	3.12%
SCAN	0.0%	99.79%	6.1%	96.38%

TABLE IX: Comparison of BSP barrier count and kernel invocation count.

Dataset	BSP Barriers		Kernel Invocations	
	Enterprise	Gunrock	Enterprise	Gunrock
roadNet-CA	258	676	3366	3682
road_usa	258	5430	3369	39633
europe_osm	583	21462	7591	121790
hollywood_2009	9	9	134	61
soc-LiveJournal1	13	15	190	103
soc-orkut	8	41	116	169
soc-twitter2010	16	16	229	113
uk-2002	27	43	372	301
web-Google	17	40	232	197
kron_g500-logn21	6	6	90	44

very high warp execution efficiency because of the specifically designed optimizations. In Enterprise, the top time-consuming kernels for the two workloads (WAP and SORT2) achieves very high WEE. As for Gunrock, the WEE of the top kernel RELAX for hollywood_2009 is relatively higher than its counterpart for roadNet-CA (kernel FORWARD). Note that higher WEE does not necessarily indicate higher performance. For example, for workload hollywood_2009, the overall WEE of Enterprise is smaller than that of Gunrock (90% vs 96%), but the performance of the former is 3x better than the latter. The same situation can be observed between the two workloads in Enterprise. Thus, WEE should not be used for performance analysis individually, and more active threads do not directly lead to higher performance, because threads might be stalled for a variety of reasons and frequent context switching consumes processor time without getting real work done.

C. Synchronization

Graph applications require frequent synchronization. There are four types of synchronization that we will discuss in the following. First, in BSP model, a barrier is needed at the end of each super-step to make sure that all conditions are met before starting the next super-step. Second, within each super-step, GPU kernel invocations are alway necessary to perform implicit global synchronizations. Third, atomic instructions are generally adopted in graph primitives to manipulate irregular graph data structures. Forth, inter-thread communications within a warp (warp vote and shuffle) provide warp-synchronous programming style. Warp vote instructions such as *__all, __any, __ballot* are like predicated *__syncthreads_or, and* instructions except that results are only aggregated across a warp. Warp shuffle instructions are intrinsic functions that allow threads to directly access another threadâĂŹs registers. Statistics about warp vote and shuffle can be collected using the metric *inst_inter_thread_communication*.

Table IX summarizes the BSP barrier count (proportional to the iterations required for an operation to converge) and kernel invocation count. Barrier count shows positive correlation with performance for most graphs. However, for road networks, the significant differences of barrier count do not translate to performance boost. From kernel invocation count, we can not

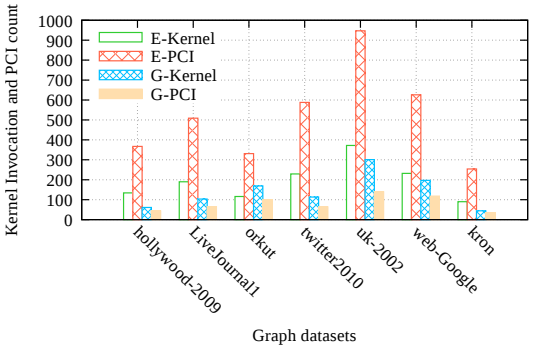

Fig. 4: Kernel invocation and PCI data transfer count.

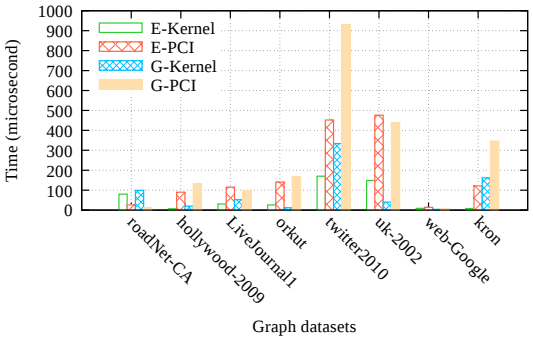

Fig. 5: Total kernel execution time and PCI data transfer time.

see such correlation. Enterprise achieves higher performance than Gunrock even it incurs more kernel invocations. Thus, although it is a common practice to reduce kernel invocations when building programmable generalized graph frameworks, it may not be the case for specialized graph processing systems. Table X shows the global memory atomic transactions. Relying on a specially-designed frontier queue generation method, Enterprise is able to not only eliminate the need of thread synchronization (via atomic instructions), but also remove duplicated frontiers from the queue that avoids potentially useless work down the road. We attribute this as a major performance improvement compared to Gunrock which incurs millions or more atomic operations across all datasets except for road networks. Finally, from Table XI we can infer that inter-thread communications exhibit no negative correlation with performance, as Enterprise invokes much more inter-thread communication instructions than Gunrock. The *vote* instruction evaluates a condition and broadcasts the result to all threads in the warp. The *shuffle* instruction enables data exchange between threads within a warp without staging the data through shared memory. By performing the exchange without both a read and a write, it can reduce shared memory usage.

D. Memory Subsystem

A negative side-effect of GPU-based applications is that CPU and GPU need to communicate frequently via PCI interface, which incurs high overhead. Our aforementioned analysis ignores the PCI data transfer overhead, considering only the kernel execution time. Here, we study the PCI data transfer

TABLE X: Global memory atomic transactions.

	roadNet-CA	hollywood_2009	soc-LiveJournal1	soc-orkut	soc-twitter2010	uk-2002	web-Google	kron_g500-logn21
Enterprise	0	0	0	0	0	0	0	0
Gunrock	0.004M	7.7M	5.6M	2.3M	34M	2.6M	0.07M	28M

TABLE XI: Number of inter-thread communication instructions executed by non-predicated threads.

	roadNet-CA	hollywood_2009	soc-LiveJournal1	soc-orkut	soc-twitter2010	uk-2002	web-Google	kron_g500-logn21
Enterprise	0	10M	10M	30M	9.3M	47M	0.54M	5.4M
Gunrock	0.01M	0.28M	0.6M	0.12M	2.2M	0.63M	0.06M	0.53M

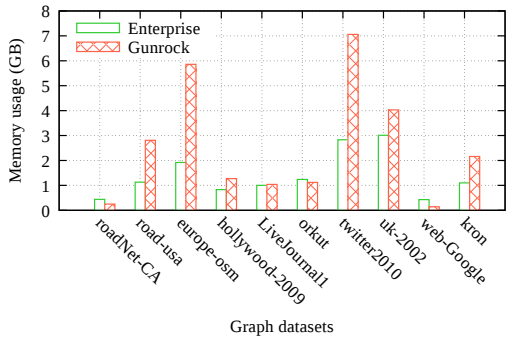

Fig. 6: Memory usage comparison.

patterns. Figure 4 compares the kernel invocation counts and *cudaMemcpy* function calls (PCI) that is used to transfer data between CPU and GPU. The results of road networks are not shown because the values are too large to fit in the same figure. One interesting phenomenon is that Enterprise interacts with CPU up to 9x more frequently than Gunrock, and the kernel invocations in Gunrock is nearly 2x more than the *cudaMemcpy* calls. This pattern normally implies higher performance, but the results in Table II show the opposite. We can explain this by examining the total kernel execution time and data transfer time as shown in Figure 5. In most cases, the total data transfer time in Enterprise is on par with Gunrock. In particular, for dataset twitter2010, although the number of calls to *cudaMemcpy* is small in Gunrock, the total transfer time is enormous. Our analysis indicate Enterprise's fine-grained data management is effective.

Both Enterprise and Gunrock use the Compressed Sparse Row (CSR) format to store graphs. The design choice of different CSR-based graph structures not only tightly correlates to efficient memory access that impacts performance, but also determines memory consumption. Figure 6 compares the memory usage. Enterprise is more memory efficient for most datasets except for roadNet-CA and web-Google, whose memory consumption is small. For the two largest datasets europe-osm and twitter2010, Gunrock allocates 3x and 2.5x more memory respectively. Besides memory usage, memory access throughput is an important indicator for overall performance. We use global memory load throughput (graph processing is read dominant) as the metric to perform comparison. The global memory throughput is the amount of data requested by instructions from the global address space, and it considers both L1 cache hits and all accesses to L2 cache. Table XII presents the results, which illustrates that higher memory throughput has a positive correlation with achieved performance. One outlier is the result from Enterprise under dataset roadNet-CA. Understanding the reason would require an in-depth investigation of Enterprise's implementation, and we leave this for future work.

IV. Discussion

Based on the above analysis, we summarize several important observations. First, although specialized system loses the flexibility of generalization, the specifically-designed optimizations may also be applicable to similar systems. Enterprise's strategies such as how to avoid atomic operations would be of great significance to Gunrock, given that both systems are based on the same concept of *frontier*. Second, understanding real performance bottlenecks requires a horizontal view of multiple factors instead of concentrating on a single metric. Some metrics are not directly connected to performance, such as the warp execution efficiency discussed in this paper. Third, different workloads may stress different components, and an insignificant kernel may dominant the execution when workload changes. Therefore, non-critical parts in an application also deserve attentions. Forth, counterintuitive patterns discovered during execution are not necessarily an indication of degrading performance, such as the kernel invocation count. Last but not least, current systems behave poorly for road networks, and new approaches are desired in future research.

V. Related Work

GPU based graph processing frameworks. Medusa [14] is an efficient implementation of the Pregel model for GPUs. Medusa provides a more fine-grained programming interface, exposing data parallelism on edges, vertices, and messages called EMV model. Even with the fine-grained interface, Medusa introduces load imbalance and non-coalesced memory access among threads on the GPU, which leads to underutilization of GPU computing resources. Furthermore, the EMV model is still complicated with too many details exposed to developers compared to the original vertex-centric model. In this paper, we present an Edge-Vertex model and two optimizations to address these issues. CuSha [2] is a graph processing framework, proposing new graph representations

TABLE XII: Global memory throughput.

	roadNet-CA	hollywood_2009	soc-LiveJournal1	soc-orkut	soc-twitter2010	uk-2002	web-Google	kron_g500-logn21
Enterprise	102.3GB/s	145.2GB/s	87.8GB/s	82.7GB/s	72.9GB/s	110.9GB/s	77.2GB/s	86.9GB/s
Gunrock	26.4GB/s	74.9GB/s	73.1GB/s	74.4GB/s	60.7GB/s	82.2GB/s	65.8GB/s	70.9GB/s

such as G-Shards and Concatenated Windows to minimize non-coalesced memory accesses and achieve higher GPU utilization for processing sparse graphs. However, the new graph representation in CuSha incurs larger memory space overhead than the conventional CSR representation. TOTEM [5] is a graph processing engine for heterogeneous many-core systems, which reduces development complexity and applies algorithm-agnostic optimizations to improve performance.

Gunrock [11] is a high-performance graph processing library targeting the GPU. Gunrock implements a data-centric abstraction, and strikes a balance between performance and expressiveness by coupling GPU computing primitives and optimization strategies with a high-level programming model. GraphReduce [10] is a scalable GPU-based framework that operates on graphs that exceed the GPU's memory capacity. GraphReduce adopts a combination of edge- and vertex-centric implementations and uses multiple GPU streams to exploit the high degree of parallelism of GPUs. Enterprise [6] is a new GPU-based BFS system that combines three techniques to remove potential performance bottlenecks: streamlined GPU threads scheduling, GPU workload balancing, and GPU based BFS direction optimization. The GPU workload balancing that classifies nodes based on different out-degrees is similar to our sorting-based load-balancing strategy, but using a different approach. GraphBIG [9] is a comprehensive benchmark suites for graph computing, which supports a wide selection of workloads for both CPU and GPU, and covers a broad scope of graph computing applications.

Characterization study on GPU-based graph processing. In [13], the authors used cycle accurate GPU simulator to analyze the impact of warp scheduler, performance bottlenecks and load imbalance across SMs, CTAs and warps for 12 specialized graph applications. [12] performs a comparison between 3 graph processing frameworks from several dimensions including efficient building block operators, synchronization, workload distribution, and data movement. Our work is different in that we focus on the characterization of a specific graph algorithm BFS to present a detailed analysis for generalization and specialization.

VI. Conclusions

In this paper, we present a characterization study of BFS on two GPU-based graph processing systems, which are representative state-of-the-art solutions in generalization and specialization. Our evaluation reveals some findings that have not been covered in related works, which we believe provides insights in how these systems run and how to improve existing both generalized and specialized systems.

Acknowledgment

This research was supported in part by the National Science Foundation of China under grants 61772183, 61272190, and 61572179.

References

[1] D. A. Bader and K. Madduri, "Gtgraph: A suite of synthetic graph generators," https://github.com/dhruvbird/GTgraph, 2006.

[2] F. K. K. V. R. G. L. N. Bhuyan, "Cusha: Vertex-centric graph processing on gpus," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing (HPDC 2014)*. ACM, 2014, pp. 239–252.

[3] A. Davidson, S. Baxter, M. Garland, and J. D. Owens, "Work-efficient parallel gpu methods for single-source shortest paths," in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, ser. IPDPS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 349–359.

[4] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, 2011.

[5] A. Gharaibeh, L. B. Costa, E. Santos-Neto, and M. Ripeanu, "A yoke of oxen and a thousand chickens for heavy lifting graph processing," in *Proceedings of the the 21st International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2012, pp. 345–354.

[6] H. Liu and H. H. Huang, "Enterprise: Breadth-first graph traversal on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2015)*. ACM, 2015, pp. 68:1–68:12.

[7] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[8] D. Merrill, M. Garland, and A. Grimshaw, "Scalable gpu graph traversal," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '12. New York, NY, USA: ACM, 2012, pp. 117–128.

[9] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "Graphbig: Understanding graph computing in the context of industrial solutions," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 69:1–69:12.

[10] D. Sengupta, S. Song, K. Agarwal, and K. Schwan, "Graphreduce: Processing large-scale graphs on accelerator-based systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*. ACM, 2015, pp. 28:1–28:12.

[11] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the gpu," in *Proceedings of the 21th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2016)*. ACM, 2016, pp. 11:1–11:12.

[12] Y. Wu, Y. Wang, Y. Pan, C. Yang, and J. D. Owens, "Performance characterization of high-level programming models for gpu graph analytics," in *Proceedings of 2015 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2015, pp. 66–75.

[13] Q. Xu, H. Jeon, and M. Annavaram, "Graph processing on gpus: Where are the bottlenecks?" in *Proceedings of 2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE Computer Society, 2014, pp. 140–149.

[14] 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, 2013.

[15] W. Zhong, J. Sun, H. Chen, J. Xiao, Z. Chen, C. Cheng, and X. Shi, "Optimizing graph processing on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1149–1162, April 2017.